

---

# podcastparser Documentation

*Release 0.6.8*

**gPodder Team**

**Jun 23, 2022**



---

## Contents

---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Example</b>                               | <b>3</b>  |
| <b>2</b> | <b>Supported XML Elements and Attributes</b> | <b>5</b>  |
| 2.1      | RSS . . . . .                                | 5         |
| 2.2      | Atom . . . . .                               | 6         |
| <b>3</b> | <b>The <code>podcastparser</code> module</b> | <b>9</b>  |
| <b>4</b> | <b>Unsupported Namespaces</b>                | <b>13</b> |
| 4.1      | Chapter Marks . . . . .                      | 13        |
| 4.2      | Others . . . . .                             | 13        |
| <b>5</b> | <b>Indices and tables</b>                    | <b>15</b> |
|          | <b>Python Module Index</b>                   | <b>17</b> |
|          | <b>Index</b>                                 | <b>19</b> |



*podcastparser* is a simple and fast podcast feed parser library in Python. The two primary users of the library are the [gPodder Podcast Client](#) and the [gpodder.net web service](#).

The following feed types are supported:

- Really Simple Syndication ([RSS 2.0](#))
- Atom Syndication Format ([RFC 4287](#))

The following specifications are supported:

- [Paged Feeds \(RFC 5005\)](#)
- [Podlove Simple Chapters](#)

These formats only specify the possible markup elements and attributes. We recommend that you also read the [Podcast Feed Best Practice](#) guide if you want to optimize your feeds for best display in podcast clients.

Where times and durations are used, the values are expected to be formatted either as seconds or as [RFC 2326](#) Normal Play Time (NPT).



# CHAPTER 1

---

## Example

---

```
import podcastparser
import urllib.request

feedurl = 'http://example.com/feed.xml'

parsed = podcastparser.parse(feedurl, urllib.request.urlopen(feedurl))

# parsed is a dict
import pprint
pprint.pprint(parsed)
```





---

### Supported XML Elements and Attributes

---

For both RSS and Atom feeds, only a subset of elements (those that are relevant to podcast client applications) is parsed. This section describes which elements and attributes are parsed and how the contents are interpreted/used.

#### 2.1 RSS

**rss+xml:base** Base URL for all relative links in the RSS file.

**rss/channel** Podcast.

**rss/channel/title** Podcast title (whitespace is squashed).

**rss/channel/link** Podcast website.

**rss/channel/description** Podcast description (whitespace is squashed).

**rss/channel/image/url** Podcast cover art.

**rss/channel/itunes:image** Podcast cover art (alternative).

**rss/channel/itunes:type** Podcast type (whitespace is squashed). One of ‘episodic’ or ‘serial’.

**rss/channel/atom:link@rel=payment** Podcast payment URL (e.g. Flattr).

**rss/channel/generator** A string indicating the program used to generate the channel. (e.g. MightyInHouse Content System v2.3).

**rss/channel/language** Podcast language.

**rss/channel/itunes:author** The group responsible for creating the show.

**rss/channel/itunes:owner** The podcast owner contact information. The <itunes:owner> tag information is for administrative communication about the podcast and isn’t displayed in Apple Podcasts

**rss/channel/itunes:explicit** Indicates whether podcast contains explicit material.

**rss/channel/itunes:new-feed-url** The new podcast RSS Feed URL.

**rss/redirect/newLocation** The new podcast RSS Feed URL.

**rss/channel/item** Episode.

**rss/channel/item/guid** Episode unique identifier (GUID), mandatory.

**rss/channel/item/title** Episode title (whitespace is squashed).

**rss/channel/item/link** Episode website.

**rss/channel/item/description** Episode description. If it contains html, it's returned as `description_html`. Otherwise it's returned as `description` (whitespace is squashed). See Mozilla's article *Why RSS Content Module is Popular*

**rss/channel/item/itunes:summary** Episode description (whitespace is squashed).

**rss/channel/item/itunes:subtitle** Episode subtitled / one-line description (whitespace is squashed).

**rss/channel/item/content:encoded** Episode description in HTML. Best source for `description_html`.

**rss/channel/item/itunes:duration** Episode duration.

**rss/channel/item/pubDate** Episode publication date.

**rss/channel/item/atom:link@rel=payment** Episode payment URL (e.g. Flattr).

**rss/channel/item/atom:link@rel=enclosure** File download URL (@href), size (@length) and mime type (@type).

**rss/channel/item/itunes:image** Episode art URL.

**rss/channel/item/media:thumbnail** Episode art URL.

**rss/channel/item/media:group/media:thumbnail** Episode art URL.

**rss/channel/item/media:content** File download URL (@url), size (@fileSize) and mime type (@type).

**rss/channel/item/media:group/media:content** File download URL (@url), size (@fileSize) and mime type (@type).

**rss/channel/item/enclosure** File download URL (@url), size (@length) and mime type (@type).

**rss/channel/item/psc:chapters** Podlove Simple Chapters, version 1.1 and 1.2.

**rss/channel/item/psc:chapters/psc:chapter** Chapter entry (@start, @title, @href and @image).

**rss/channel/item/itunes:explicit** Indicates whether episode contains explicit material.

**rss/channel/item/itunes:author** The group responsible for creating the episode.

**rss/channel/item/itunes:episode** An episode number.

**rss/channel/item/itunes:episodeType** The episode type. This flag is used if an episode is a trailer or bonus content.

## 2.2 Atom

For Atom feeds, *podcastparser* will handle the following elements and attributes:

**atom:feed** Podcast.

**atom:feed/atom:title** Podcast title (whitespace is squashed).

**atom:feed/atom:subtitle** Podcast description (whitespace is squashed).

**atom:feed/atom:icon** Podcast cover art.

**atom:feed/atom:link@href** Podcast website.

**atom:feed/atom:entry** Episode.

**atom:feed/atom:entry/atom:id** Episode unique identifier (GUID), mandatory.

**atom:feed/atom:entry/atom:title** Episode title (whitespace is squashed).

**atom:feed/atom:entry/atom:link@rel=enclosure** File download URL (@href), size (@length) and mime type (@type).

**atom:feed/atom:entry/atom:link@rel=(self|alternate)** Episode website.

**atom:feed/atom:entry/atom:link@rel=payment** Episode payment URL (e.g. Flattr).

**atom:feed/atom:entry/atom:content** Episode description (in HTML or plaintext).

**atom:feed/atom:entry/atom:published** Episode publication date.

**atom:feed/atom:entry/media:thumbnail** Episode art URL.

**atom:feed/atom:entry/media:group/media:thumbnail** Episode art URL.

**atom:feed/atom:entry/psc:chapters** Podlove Simple Chapters, version 1.1 and 1.2.

**atom:feed/atom:entry/psc:chapters/psc:chapter** Chapter entry (@start, @title, @href and @image).



## CHAPTER 3

---

### The podcastparser module

---

Simplified, fast RSS parser

**exception** `podcastparser.FeedParseError` (*msg, exception, locator*)

Exception raised when asked to parse an invalid feed

This exception allows users of this library to catch exceptions without having to import the XML parsing library themselves.

**class** `podcastparser.RSSItemDescription`

RSS 2.0 almost encourages to put html content in item/description but content:encoded is the better source of html content and itunes:summary is known to contain the short textual description of the item. So use a heuristic to attribute text to either description or description\_html, without overriding existing values.

`podcastparser.file_basename_no_extension` (*filename*)

Returns filename without extension

```
>>> file_basename_no_extension('/home/me/file.txt')
'file'
```

```
>>> file_basename_no_extension('file')
'file'
```

`podcastparser.is_html` (*text*)

Heuristically tell if text is HTML

By looking for an open tag (more or less:) >>> `is_html('<h1>HELLO</h1>')` True >>> `is_html('a < b < c')` False

`podcastparser.normalize_feed_url` (*url*)

Normalize and convert a URL. If the URL cannot be converted (invalid or unknown scheme), None is returned.

This will also normalize `feed://` and `itpc://` to `http://`.

```
>>> normalize_feed_url('itpc://example.org/podcast.rss')
'http://example.org/podcast.rss'
```

If no URL scheme is defined (e.g. “curry.com”), we will simply assume the user intends to add a <http://> feed.

```
>>> normalize_feed_url('curry.com')
'http://curry.com/'
```

It will also take care of converting the domain name to all-lowercase (because domains are not case sensitive):

```
>>> normalize_feed_url('http://Example.COM/')
'http://example.com/'
```

Some other minimalistic changes are also taken care of, e.g. a ? with an empty query is removed:

```
>>> normalize_feed_url('http://example.org/test?')
'http://example.org/test'
```

Leading and trailing whitespace is removed

```
>>> normalize_feed_url(' http://example.com/podcast.rss ')
'http://example.com/podcast.rss'
```

Incomplete (too short) URLs are not accepted

```
>>> normalize_feed_url('http://') is None
True
```

Unknown protocols are not accepted

```
>>> normalize_feed_url('gopher://gopher.hprc.utoronto.ca/file.txt') is None
True
```

`podcastparser.parse(url, stream, max_episodes=0)`

Parse a podcast feed from the given URL and stream

#### Parameters

- **url** – the URL of the feed. Will be used to resolve relative links
- **stream** – file-like object containing the feed content
- **max\_episodes** – maximum number of episodes to return. 0 (default) means no limit

**Returns** a dict with the parsed contents of the feed

`podcastparser.parse_length(text)`

Parses a file length

```
>>> parse_length(None)
-1
```

```
>>> parse_length('0')
-1
```

```
>>> parse_length('unknown')
-1
```

```
>>> parse_length('100')
100
```

podcastparser.**parse\_pubdate**(*text*)  
Parse a date string into a Unix timestamp

```
>>> parse_pubdate('Fri, 21 Nov 1997 09:55:06 -0600')
880127706
```

```
>>> parse_pubdate('2003-12-13T00:00:00+02:00')
1071266400
```

```
>>> parse_pubdate('2003-12-13T18:30:02Z')
1071340202
```

```
>>> parse_pubdate('Mon, 02 May 1960 09:05:01 +0100')
-305049299
```

```
>>> parse_pubdate('')
0
```

```
>>> parse_pubdate('unknown')
0
```

podcastparser.**parse\_time**(*value*)  
Parse a time string into seconds  
See RFC2326, 3.6 “Normal Play Time” (HH:MM:SS.FRACT)

```
>>> parse_time('0')
0
>>> parse_time('128')
128
>>> parse_time('00:00')
0
>>> parse_time('00:00:00')
0
>>> parse_time('00:20')
20
>>> parse_time('00:00:20')
20
>>> parse_time('01:00:00')
3600
>>> parse_time(' 03:02:01')
10921
>>> parse_time('61:08')
3668
>>> parse_time('25:03:30 ')
90210
>>> parse_time('25:3:30')
90210
>>> parse_time('61.08')
61
>>> parse_time('01:02:03.500')
3723
>>> parse_time(' ')
0
```

podcastparser.**parse\_type**(*text*)  
“normalize” a mime type

```
>>> parse_type('text/plain')
'text/plain'
```

```
>>> parse_type('text')
'application/octet-stream'
```

```
>>> parse_type('')
'application/octet-stream'
```

```
>>> parse_type(None)
'application/octet-stream'
```

`podcastparser.remove_html_tags(html)`

Remove HTML tags from a string and replace numeric and named entities with the corresponding character, so the HTML text can be displayed in a simple text view.

`podcastparser.squash_whitespace(text)`

Combine multiple whitespaces into one, trim trailing/leading spaces

```
>>> squash_whitespace(' some          text  with a    lot of   spaces ')
'some text with a lot of spaces'
```

`podcastparser.squash_whitespace_not_nl(text)`

Like `squash_whitespace`, but don't squash linefeeds and carriage returns

```
>>> squash_whitespace_not_nl(' linefeeds\ncarriage\r returns')
'linefeeds\ncarriage\r returns'
```



---

## Unsupported Namespaces

---

This is a list of podcast-related XML namespaces that are not yet supported by podcastparser, but might be in the future.

### 4.1 Chapter Marks

- [rawvoice RSS](#): Rating, Frequency, Poster, WebM, MP4, Metamark (kind of chapter-like markers)
- [IGOR](#): Chapter Marks

### 4.2 Others

- [libSYN RSS Extensions](#): contactPhone, contactEmail, contactTwitter, contactWebsite, wallpaper, pdf, background
- [Comment API](#): Comments to a given item (readable via RSS)
- [MVCB](#): Error Reports To Field (usually a mailto: link)
- [Syndication Module](#): Update period, frequency and base (for skipping updates)
- [Creative Commons RSS](#): Creative commons license for the content
- [Pheedo](#): Original link to website and original link to enclosure (without going through pheedo redirect)
- [WGS84](#): Geo-Coordinates per item
- [Conversations Network](#): Intro duration in milliseconds (for skipping the intro), ratings
- [purl DC Elements](#): dc:creator (author / creator of the podcast, possibly with e-mail address)
- [Tristana](#): tristana:self (canonical URL to feed)
- [Blip](#): Show name, show page, picture, username, language, rating, thumbnail\_src, license



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**p**

podcastparser, 9



### F

`FeedParseError`, 9  
`file_basename_no_extension()` (*in module `podcastparser`*), 9

### I

`is_html()` (*in module `podcastparser`*), 9

### N

`normalize_feed_url()` (*in module `podcastparser`*), 9

### P

`parse()` (*in module `podcastparser`*), 10  
`parse_length()` (*in module `podcastparser`*), 10  
`parse_pubdate()` (*in module `podcastparser`*), 10  
`parse_time()` (*in module `podcastparser`*), 11  
`parse_type()` (*in module `podcastparser`*), 11  
`podcastparser` (*module*), 9

### R

`remove_html_tags()` (*in module `podcastparser`*), 12  
`RSSItemDescription` (*class in `podcastparser`*), 9

### S

`squash_whitespace()` (*in module `podcastparser`*), 12  
`squash_whitespace_not_nl()` (*in module `podcastparser`*), 12