gpodder.net Documentation

gpodder.net Team

Dec 22, 2022

Contents

gpodder.net is a podcasting webservice. It connects your podcast players and lets you discover new and interesting podcasts.

This documentation is intended for developers. Please refer to the *API Documentation* if you are developing podcast applications and want to integrate functionality from gpodder.net. Please read the *Developer Documentation* if you want to contribute to gpodder.net itself.

CHAPTER 1

Contents

1.1 User Documentation

gpodder.net is a web service to manage your podcast subscriptions via the web. You can synchronize your devices, view status information and discover new interesting podcasts online.

1.1.1 Supported Clients

A list of supported clients is available in *Clients*.

To configure gPodder to connect to gpodder.net, open the my.gpodder.org / gpodder.net configuration dialog from the Subscriptions menu. Enter username and password that you've used during the registration on the webservice. The device ID is automatically generated from the hostname and will be used to identify this specific device in the webservice. Next enable the synchronisation of your subscription list. For the start, upload your subscription list to the webservice. Subsequent changes will be transmitted automatically.

1.1.2 Devices

Each device connected to the webservice will be identified by its *Device ID* which should therefore be unique (at least for your user account). *Do not try to synchronize devices by using the same Device ID.*

A list of your devices can be found on the devices page.

Synchronizing Devices

If you have at least two devices connected to the webservice, you can synchronize some of them. Open the device page for one of them, click synchronize and select the device to synchronize with. After synchronizing, the subscriptions of the devices will be merged. Adding a subscription at one device will automatically add it to the others. Same for deletions.

The device list groups devices that are synchronized with each other.

1.1.3 Episode States

Episode states (such as played, downloaded, etc) are synchronized across all devices.

1.1.4 Privacy Settings

By default we include information about your subscriptions in our toplist and podcast suggestions . *We will never associate your username and/or email address with your subscriptions on public pages.* You can even opt-out from our anonymized statistics at the privacy page. If you mark some podcasts as private, they will also not show up in your sharable subscription list.

1.1.5 Sharing your Subscriptions

If you want to let others know about your which podcasts you are listening to, go to your sharing page . You can either share your private URL with your friends or make it public and share your subscriptions with the whole world.

Clients

Android

Name	Status	gpodder.net features	License	Documentation
AntennaPod	Active	Directory, Search, Subscription sync, Episode Action sync	GPL 3	Synchronisation between devices
Escapepod	Active	Search	MIT	
Listen Up Free	Active	sync, subscription, search		
Podcast Addict	Active	none		Podcast Addict export script
Podkicker	Active	none		
Podax	Abandoned	Subscription sync	BSD	
Volksempfänger	Abandoned	Podcast search	ISC	
SwallowCatcher	Abandoned	Planned	AGPL	
Detlef Gpodderson	Abandoned	Subscription and Episode Action sync	GPL	Detlef Repository
gpodroid	Abandoned	Subscriptions only	EPL	gpodroid Repository
Podstars	Abandoned	Podcast search		
Podcatcher Deluxe	Deprecated	Sync of subscriptions and actions	GPL 3	
SoundWaves	Abandoned	Subscriptions sync	GPL 3	

Mac OS X

Name	Status	gpodder.net features	License	Documentation
gPodder	Active	All	GPL	gPodder Docs
Clementine	Active	Directory, Search, Subscription sync	GPL	
Mopidy-Podcast-GPodder	Deprecated	Directory, Search	Apache 2.0	

Windows

Name	Status	gpodder.net features	License	Documentation
gPodder	Active	All	GPL	gPodder Docs
Clementine	Active	Directory, Search, Subscription sync (since 1.1)	GPL	

Web

Name	Status	gpodder.net features	License	Documentation
Cloud Caster	Active	Directory, Search, Subscription sync		

Linux

Name	Status	gpodder.net features	License	Documentation
gPodder	Active	All	GPL	gPodder Docs
Amarok	Active	Directory, Search, Subscription-Sync (under development)	GPL	
BashPodder		Subscription sync via “bpsync” included in mygpoclient	GPL	
Clementine	Active	Directory, Search, Subscription sync	GPL	
Mopidy-Podcast-GPodder	Development	Directory, Search	Apache 2.0	

Windows Phone 7 / 8

Name	Status	gpodder.net features	License	Documentation
WPodder	??	Directory, subscription import	??	
Podcatcher	Active	Directory search, subscription import	GPL	
Carboncast	Active	Podcast search, Subscription import	??	

Maemo

Name	Status	gpodder.net features	License	Documentation
gPodder	Active	All	GPL	gPodder Docs

MeeGo

Name	Status	gpodder.net features	License	Documentation
gPodder	Active	All	GPL	gPodder Docs
Podcasts	Active	Unknown	Proprietary	
Podcatcher for Nokia N9	Active	Directory search, Subscription import	GPL	

webOS

Name	Status	gpodder.net features	License	Documentation
drPodder	Active	Directory search via patch	GPL	

Symbian

Name	Status	gpodder.net features	License	Documentation
Nokia Podcasting	•	Subscription service	GPL	Podcasting Docs
Podmaster	Active	Directory search		
Poddi	Active	Directory Search		Poddi in the Nokia Store

1.2 Podcast Publisher Documentation

The page contains useful information for podcast publishers.

1.2.1 Contents

Frequently Asked Questions

How can I add my podcast?

If your podcast is missing from the gpodder.net directory, go to https://gpodder.net/missing/ and enter the URL of your podcast feed.

How can I add a logo / description / etc?

All information of a podcast is retrieved from the podcast feed. To add some information on gpodder.net, add it to the podcast feed and it will be shown on gpodder.net after the next update.

How often is my podcast updated?

This depends on how often your podcast publishes new episodes. Basically we use the average time between episodes as the update interval.

How can I ensure my podcast is up-to-date?

There are several ways

- Use Pubsubhubbub. Your hub will then trigger the podcast update whenever a new episode is released.
- Request a Publisher account. You can then trigger an update of your podcast on gpodder.net manually.
- Donate. If we have more resources available for updating podcast feeds, all podcasts will be updated quicker.

1.3 API Documentation

This is the specification of Version 2 of the public API for the gpodder.net Web Services.

Please consult the *Integration Guide* before integrating the gpodder.net API in your application.

There are two different APIs for different target audiences:

- The **Simple API** targets developers who want to write quick integration into their existing applications
- The **Advanced API** targets developers who want tight integration into their applications (with more features)

The API is versioned so that changes in the major version number indicate backwards incompatible changes. All other changes preserve backwards compatibility. See *API Changes* for a list of changes. The current version is 2.11. This versioning scheme has been introduced in bug 1273.

The API is available as a machine-readable file in OpenAPI format at https://raw.githubusercontent.com/gpodder/mygpo/master/mygpo/api/openapi.yaml

1.3.1 Contents

Integration Guide

This guide describes how the gpodder.net API can be integrated in podcast applications. It describes good practice and points out caveats.

General

- The Mailing List is the right place to ask questions
- Consult the *API Reference* for available functionality.
- Add your client to the clients list when you're ready

- Please use the name *gpodder.net* (all lowercase, .net suffix) to refer to the webservice. *gPodder* (uppercase P, no suffix) refers to the client application.

Implementation

- If possible/available use an existing library.
- If you have to implement your own client, please consider releasing it as a library.
- Try to keep the requests to the API to a sensible limit. There are no hard limit, please judge for yourself what is necessary in your case. Please ask on the mailing list if unsure. Your client might get blocked if it misbehaves.
- Your client should send a useful User-Agent header. We might block clients with generic/missing User-Agent headers.

Integration

The following contains useful information for integrating gpodder.net into a podcast application.

Device

Many API endpoints refer to a *device*. A device is an instance of a client accessing the API. The ID of a device must be unique per user. Therefore clients should generate a device ID such that it is unique for the user, even he uses the same application on multiple devices. A common strategy is to include the applications name and the hostname

A user might use several clients for playing podcasts, which could generate device Id like the following

- gPodder on his N9 (*gpodder-n9*)
- gPodder on his notebook (*gpodder-netbook*)
- Amarok on his PC (*amarok-mypc*)
- a web based player (*mywebservice-myusername*)

When a previously unknown device Id is used in some API request, a device is automatically created. Refer to the *Device API* on how to provide some information about the device. Users can manage their devices online.

Podcast Directory

The most basic *passive* integration with gpodder.net is to access some of its public data. Refer to the *Directory API* for available endpoints.

Subscription Management

The most common form of *active* integration is subscription management. Clients can upload the podcast subscriptions using their device Id and receive subscription changes (for their device) that were made online. Refer to the *Subscriptions API* for additional information.

Episode Actions Synchronization

Clients can upload and download certain actions (episode downloaded, played, deleted) to/from gpodder.net. This gives the user a central overview of where and when he accessed certain podcast episodes, and allows clients to synchronise states between applications. Refer to the *Episode Actions API* for further information.

API Reference

This is the reference documentation for the gpodder.net API 2.

The *Integration Guide* contains additional non-normative information for integrating gpodder.net into podcasting applications.

Contents

General Information

Protocol

The API is provided via https. Requests via http will redirect to the corresponding URL via https.

CORS

All endpoints send the Access-Control-Allow-Origin: * header which allows web application to access the API.

Identifying Podcasts and Episodes

Podcast is identified by its feed URL, episode is identified by its media URL.

Date Format

Date format: ISO 8601 / RFC 3339: `YYYY-MM-DDTHH:MM:SSZ`

Formats

All data is exchanged as JSON. All resources are represented as JSON objects, and requests are expected as also expected to contain JSON objects.

JSONP Callbacks

You can pass a `json=<function-name>` parameter to any GET call to have the results wrapped in a JSON function. This is typically used when browsers want to embed content received from the API in web pages by getting around cross domain issues. The response includes the same data output as the regular API, plus the relevant HTTP Header information.

API Parametrization

Since 2.7

Clients should retrieve and process clientconfig.json (see *Client Parametrization*) before making requests to the web-service. If a client can not process the configuration, it can assume the default configuration given in the clientconfig.json documentation.

Devices

Devices are used throughout the API to identify a device / a client application. A device ID can be any string matching the regular expression `[\w.-]+`. The client application MUST generate a string to be used as its device ID, and SHOULD ensure that it is unique within the user account. A good approach is to combine the application name and the name of the host it is running on.

If two applications share a device ID, this might cause subscriptions to be overwritten on the server side. While it is possible to retrieve a list of devices and their IDs from the server, this SHOULD NOT be used to let a user select an existing device ID.

Formats

Most of the resources are offered in several different formats

- OPML
- JSON
- JSONP with an option function name that wraps the result (since 2.8)
- plain text with one URL per line
- XML a custom XML format (see example, since 2.9)

JSON

```
[
 {
   "website": "http://sixgun.org",
   "description": "The hardest-hitting Linux podcast around",
   "title": "Linux Outlaws",
   "author": "Sixgun Productions",
   "url": "http://feeds.feedburner.com/linuxoutlaws",
   "position_last_week": 1,
   "subscribers": 1954,
   "mygpo_link": "http://gpodder.net/podcast/11092",
   "logo_url": "http://sixgun.org/files/linuxoutlaws.jpg",
   "scaled_logo_url": "http://gpodder.net/logo/64/
↪fa9fd87a4f9e488096e52839450afe0b120684b4.jpg"
 },
]
```

XML

```
<podcasts>
 <podcast>
  <title>Linux Outlaws</title>
  <url>http://feeds.feedburner.com/linuxoutlaws</url>
  <website>http://sixgun.org</website>
  <mygpo_link>http://gpodder.net/podcast/11092</mygpo_link>
  <author>Sixgun Productions</author>
  <description>The hardest-hitting Linux podcast around</description>
  <subscribers>1954</subscribers>
  <logo_url>http://sixgun.org/files/linuxoutlaws.jpg</logo_url>
  <scaled_logo_url>http://gpodder.net/logo/64/
↪fa9fd87a4f9e488096e52839450afe0b120684b4.jpg</scaled_logo_url>
 </podcast>
</podcasts>
```

API Variants

Simple API

The Simple API provides a way to upload and download subscription lists in bulk. This allows developers of podcast-related applications to quickly integrate support for the web service, as the only

- Synchronization of episode status fields is not supported
- This API uses more bandwith than the advanced API
- The client can be stateless
- The client can be low-powered - subscribe/unsubscribe events are calculated on the server-side

Advanced API

The Advanced API provides more flexibility and enhanced functionality for applications that want a tighter integration with the web service. A reference implementation will be provided as part of the gPodder source code (and gPodder will make use of that reference implementation).

- The client has to persist the synchronization state locally
- Only changes to subscriptions are uploaded and downloaded
- Synchronization of episode status fields is supported in this API
- Only JSON is used as the data format to ease development

Authentication API

Login / Verify Login

POST /api/2/auth/(*username*)**/login.json**

- since 2.10

Log in the given user for the given device via HTTP Basic Auth.

Parameters

- **`username`** – the username which should be logged in

Status Codes

- 401 Unauthorized – If the URL is accessed without login credentials provided
- 400 Bad Request – If the client provides a cookie, but for a different username than the one given
- 200 OK – the response headers have a `sessionid` cookie set.

The client can use this URL with the cookie in the request header to check if the cookie is still valid.

Logout

`POST /api/2/auth/`(*username*)**`/logout.json`**

- since 2.10

Log out the given user. Removes the session ID from the database.

Parameters

- **`username`** – the username which should be logged out

Status Codes

- 200 OK – if the client didn't send a cookie, or the user was successfully logged out
- 400 Bad Request – if the client provides a cookie, but for a different username than the one given

Directory API

Retrieve Top Tags

`GET /api/2/tags/`(**int:** *count*)**`.json`**

- Does not require authentication
- Since 2.2

Example response:

```
HTTP/1.1 200 OK

[
  {
   "title": "Technology",
   "tag": "technology",
   "usage": 530
  },
  {
   "title": "Society & Culture",
   "tag": "society-culture",
   "usage": 420
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "title": "Arts",
    "tag": "arts",
    "usage": 400
   },
   {
    "title": "News & Politics",
    "tag": "News & Politics",
    "usage": 320
   }
]
```

Parameters

- **count** – number of tags to return

Retrieve Podcasts for Tag

GET /api/2/tag/(*tag*)**/**
int: *count*.**json**

- Does not require authentication
- Since 2.2

Example response:

```
HTTP/1.1 200 OK

[
 {"url": "http://leo.am/podcasts/floss",
  "title": "FLOSS Weekly",
  "author": "Leo Laporte",
  "description": "Each Thursday we talk about Free Libre and Open Source Software
↪with the people who are writing it. Part of the TWiT Netcast Network.",
  "subscribers": 1138,
  "logo_url: "http://leoville.tv/podcasts/coverart/floss144audio.jpg",
  "website": "http://twit.tv/",
  "mygpo_link": "http://gpodder.net/podcast/12925"},

 {"url": "http://leo.am/podcasts/twit",
  "title": "this WEEK in TECH - MP3 Edition",
  "author": "Leo Laporte",
  "description": "Your first podcast of the week is the last word in tech. [...]",
  "subscribers": 895,
  "logo_url": "http://leoville.tv/podcasts/coverart/twit144audio.jpg",
  "website": "http://thisweekintech.com/",
  "mygpo_link": "http://thisweekintech.com/"}
]
```

Parameters

- **tag** – URL-encoded tag
- **count** – maximum number of podcasts to return

Retrieve Podcast Data

GET /api/2/data/podcast.json

Returns information for the podcast with the given URL or 404 if there is no podcast with this URL.

- No authentication required
- Since 2.2

```
HTTP/1.1 200 OK

{
 "website": "http://coverville.com",
 "mygpo_link": "http://www.gpodder.net/podcast/16124",
 "description": "The best cover songs, delivered to your ears two to three times␣
↪a week!",
 "subscribers": 19,
 "title": "Coverville",
 "author": "Brian Ibbott",
 "url": "http://feeds.feedburner.com/coverville",
 "logo_url": "http://www.coverville.com/art/coverville_iTunes300.jpg"
}
```

Query Parameters

- **`url`** – the feed URL of the podcast

Retrieve Episode Data

GET /api/2/data/episode.json

Returns information for the episode with the given {url} that belongs to the podcast with the {podcast}

- Does not require authentication
- Since 2.2 (added released in 2.6)

Example response:

```
HTTP/1.1 200 OK

{
 "title": "TWiT 245: No Hitler For You",
 "url": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.com/twit/
↪twit0245.mp3",
 "podcast_title": "this WEEK in TECH - MP3 Edition",
 "podcast_url": "http://leo.am/podcasts/twit",
 "description": "[...]",
 "website": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.com/
↪twit/twit0245.mp3",
 "released": "2010-12-25T00:30:00",
 "mygpo_link": "http://gpodder.net/episode/1046492"
}
```

Query Parameters

- **`podcast`** – feed URL of the podcast to which the episode belongs
- **`url`** – media URL of the episode

Podcast Toplist

GET /toplist/(**int:** *number*)**.***format*

- Does not require authentication (public content)
- Since 1.0

Example request:

```
GET /toplist/50.json
```

Example response:

```
HTTP/1.1 200 OK

[
 {
   "website": "http://linuxoutlaws.com/podcast",
   "description": "Open source talk with a serious attitude",
   "title": "Linux Outlaws",
   "author": "Sixgun Productions",
   "url": "http://feeds.feedburner.com/linuxoutlaws",
   "position_last_week": 0,
   "subscribers": 1736,
   "mygpo_link": "http://www.gpodder.net/podcast/11092",
   "logo_url": "http://linuxoutlaws.com/files/albumart-itunes.jpg"
 },
 {
   "website": "http://syndication.mediafly.com/redirect/show/
↪d581e9b773784df7a56f37e1138c037c",
   "description": "We're not talking dentistry here; FLOSS all about Free Libre␣
↪Open Source Software. Join hosts Randal Schwartz and Leo Laporte every Saturday␣
↪as they talk with the most interesting and important people in the Open Source␣
↪and Free Software community.",
   "title": "FLOSS Weekly Video (large)",
   "author": "Leo Laporte",
   "url": "http://feeds.twit.tv/floss_video_large",
   "position_last_week": 0,
   "subscribers": 50,
   "mygpo_link": "http://www.gpodder.net/podcast/31991",
   "logo_url": "http://static.mediafly.com/publisher/images/
↪06cecab60c784f9d9866f5dcb73227c3/icon-150x150.png"
 }]
```

Query Parameters

- **jsonp** – a functionname on which the response is wrapped (only valid for format `jsonp`; since 2.8)
- **scale_logo** – returns logo URLs to scaled images, see below.

Parameters

- **number** – maximum number of podcasts to return
- **format** – see *Formats*

The number field might be any value in the range 1..100 (inclusive both boundaries).

For the JSON and XML formats, an optional paramter scale_logo={size} can be passed, which provides a link to a scaled logo (scaled_logo_url) for each podcast. size has to be a positive number up to 256 and defaults to 64.

The OPML and TXT formats do not add any information about the (absolute and relative) popularity for each podcast, only the ordering can be considered. The JSON format includes a more detailed list, usable for clients that want to display a detailed toplist or post-process the toplist:

All shown keys must be provided by the server. The description field may be set to the empty string in case a description is not available. The title field may be set to the URL in case a title is not available.

Podcast Search

GET /search.(*format*)

Carries out a service-wide search for podcasts that match the given query. Returns a list of podcasts. See *Formats* for details on the response formats.

- Does not require authentication (public content)
- Since 2.0

Query Parameters

- **q** – search query
- **jsonp** – used to wrap the JSON results in a function call (JSONP); the value of this parameter is the name of the function; since 2.8
- **scale_logo** – when set, the results (only JSON and XML formats) include links to the podcast logos that are scaled to the requested size. The links are provided in the scaled_logo_url field; since 2.9

Parameters

- **format** – see *Formats*

Suggestions API

Retrieve Suggested Podcasts

GET /suggestions/(**int:** *number*)**.** *format*

- Requires HTTP authentication
- Since 1.0

Example request:

```
GET /suggestions/10.opml
```

Example response:

```
HTTP/1.1 200 OK

[
  {
   "website": "http://www.linuxgeekdom.com",
```

(continues on next page)

(continued from previous page)

```
  "mygpo_link": "http://gpodder.net/podcast/64439",
  "description": "Linux Geekdom",
  "subscribers": 0,
  "title": "Linux Geekdom",
  "author": "aj@linuxgeekdom.com (A.J. Stringham)",
  "url": "http://www.linuxgeekdom.com/rssmp3.xml",
  "logo_url": null
},
{
  "website": "http://goinglinux.com",
  "mygpo_link": "http://gpodder.net/podcast/11171",
  "description": "Going Linux",
  "subscribers": 571,
  "title": "Going Linux",
  "author": "Larry Bushey",
  "url": "http://goinglinux.com/mp3podcast.xml",
  "logo_url": "http://goinglinux.com/images/GoingLinux80.png"
}]
```

Parameters

- **number** – the maximum number of podcasts to return
- **format** – see *Formats*

Query Parameters

- **jsonp** – function name for the JSONP format (since 2.8)

Download a list of podcasts that the user has not yet subscribed to (by checking all server-side subscription lists) and that might be interesting to the user based on existing subscriptions (again on all server-side subscription lists).

The TXT format is a simple URL list (one URL per line), and the OPML file is a "standard" OPML feed. The JSON format looks as follows:

The server does not specify the "relevance" for the podcast suggestion, and the client application SHOULD filter out any podcasts that are already added to the client application but that the server does not know about yet (although this is just a suggestion for a good client-side UX).

Device API

Update Device Data

POST /api/2/devices/(*username*)**/***deviceid***.json**

- Requires HTTP authentication
- Since 2.0

The device ID is generated by the client application to identify itself in API requests. The name is used to display a human readable identifier to the user on the webservice. Only the keys that are supplied will be updated.

Example request:

```
POST /api/2/devices/a-user/somedevice-123.json

{
    "caption": "gPodder on my Lappy",
    "type": "laptop"
}
```

Parameters

- **`username`** – the username for which device data should be updated
- **`deviceid`** – see *Devices*

Request JSON Object

- **`caption`** (*`string`*) – The new human readable label for the device
- **`type`** (*`string`*) – he type of the device. Possible values: desktop, laptop, mobile, server, other

List Devices

`GET /api/2/devices/`(*username*)**`.json`**

- Requires HTTP authentication
- Since 2.0

Returns the list of devices that belong to a user. This can be used by the client to let the user select a device from which to retrieve subscriptions, etc..

Example response:

```
HTTP/1.1 200 OK

[
  {
   "id": "abcdef",
   "caption": "gPodder on my Lappy",
   "type": "laptop",
   "subscriptions": 27
  },
  {
   "id": "09jc23caf",
   "caption": "",
   "type": "other",
   "subscriptions": 30
  },
  {
   "id": "phone-au90f923023.203f9j23f",
   "caption": "My Phone",
   "type": "mobile",
   "subscriptions": 5
  }
]
```

Parameters

- **`username`** – the username for which the devices should be returned

Get Device Updates

GET /api/2/updates/(*username*)**/** *deviceid*.**json**

- Requires Authentication
- Since 2.3

Example response:

```
HTTP/1.1 200 OK

{
    "add":     [
    {
       "title": "PaulDotCom Security Weekly",
       "url": "http://pauldotcom.com/podcast/psw.xml",
       "description": "PaulDotCom Security Weekly Podcast with Paul, Larry, Mick,␣
↪Carlos, and special guests!",
       "subscribers": 93,
       "logo_url": "http://pauldotcom.com/images/psw-logo-sm.png"
       "website": "http://pauldotcom.com/",
       "mygpo_link": "http://gpodder.net/podcast/11194",
    }
  ],

  "remove":  ["<URL3>"],

  "updates": [
    {
      "title": "TWiT 245: No Hitler For You",
      "url": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.com/
↪twit/twit0245.mp3",
      "podcast_title": "this WEEK in TECH - MP3 Edition",
      "podcast_url": "http://leo.am/podcasts/twit",
      "description": "[...]",
      "website": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.
↪com/twit/twit0245.mp3",
      "mygpo_link": "http://gpodder.net/episode/1046492"
      "released":   """2009-12-12T09:00:00"
      "status":        "(new|play|download|delete)"
     }
    ],

   "timestamp":   <timestamp>
}
```

Query Parameters

- **since** – `timestamp` when updates have last been retrieved
- **include_actions** (*bool*) – Default: false, since 2.10

The response will have the following form and will contain

- a list of subscriptions to be added, with URL, title and descriptions
- a list of URLs to be unsubscribed

- a list of updated episodes
- the current timestamp; for retrieving changes since the last query

If include_actions is set to true, each updated episode (with a state other than new) will contain an additional property action which includes the user's latest episode action reported for this episode. The actions have the same format as in *Episode Action Types*.

Subscriptions API

Get Subscriptions of Device

GET /subscriptions/(*username*)**/***deviceid*.*format*

- Requires HTTP authentication
- Since 1.0

Example request:

```
GET /subscriptions/bob/asdf.opml
```

Parameters

- **username** – username for which subscriptions should be returned
- **deviceid** – see *Devices*
- **format** – see *Formats*

Query Parameters

- **jsonp** – function name for the JSONP format (since 2.8)

Status Codes

- 200 OK – the subscriptions are returned in the requested format
- 401 Unauthorized – Invalid user
- 404 Not Found – Invalid device ID
- 400 Bad Request – Invalid format

Get All Subscriptions

GET /subscriptions/(*username*)**.***format*

- Requires HTTP authentication
- Since 2.11

Example request:

```
GET /subscriptions/bob.opml
```

This can be used to present the user a list of podcasts when the application starts for the first time.

Parameters

- **username** – username for which subscriptions should be returned
- **deviceid** – see *Devices*
- **format** – see *Formats*

Query Parameters

- **jsonp** – function name for the JSONP format (since 2.8)

Status Codes

- 200 OK – the subscriptions are returned in the requested format
- 401 Unauthorized – Invalid user
- 400 Bad Request – Invalid format

Upload Subscriptions of Device

PUT /subscriptions/(*username*)**/***deviceid*.*format*

- Requires HTTP authentication
- Since 1.0

Upload the current subscription list of the given user to the server. The data should be provided either in OPML, JSON or plaintext (one URL per line) format, and should be uploaded just like a normal PUT request (i.e. in the body of the request).

For successful updates, the implementation always returns the status code 200 and the empty string (i.e. an empty HTTP body) as a result, any other string should be interpreted by the client as an (undefined) error.

Example request:

```
PUT /subscriptions/john/e9c4ea4ae004efac40.txt
```

Parameters

- **username** – username for which subscriptions should be uploaded
- **deviceid** – see *Devices*
- **format** – see *Formats*

Status Codes

- 200 OK – the subscriptions have been updated
- 401 Unauthorized – Invalid user
- 400 Bad Request – Invalid format

In case the device does not exist for the given user, it is automatically created. If clients want to determine if a device exists, you have to to a GET request on the same URL first and check for a the 404 status code (see above).

Upload Subscription Changes

POST /api/2/subscriptions/(*username*)**/** *deviceid***.json**

- Requires HTTP authentication
- Since 2.0

Only deltas are supported here. Timestamps are not supported, and are issued by the server.

Example request:

```
{
    "add": ["http://example.com/feed.rss", "http://example.org/podcast.php"],
    "remove": ["http://example.net/foo.xml"]
}
```

Parameters

- **username** – username for which subscriptions should be returned
- **deviceid** – see *Devices*

Status Codes

- 400 Bad Request – the same feed has been added and removed in the same request
- 200 OK – the subscriptions have been updated

In positive responses the server returns a timestamp/ID that can be used for requesting changes since this upload in a subsequent API call. In addition, the server sends a list of URLs that have been rewritten (sanitized, see bug:747) as a list of tuples with the key "update_urls". The client SHOULD parse this list and update the local subscription list accordingly (the server only sanitizes the URL, so the semantic "content" should stay the same and therefore the client can simply update the URL value locally and use it for future updates.

Example response:

```
{
  "timestamp": 1337,
  "update_urls":
   [
    [
     "http://feeds2.feedburner.com/LinuxOutlaws?format=xml",
     "http://feeds.feedburner.com/LinuxOutlaws"
    ],
    [
     "http://example.org/podcast.rss ",
     "http://example.org/podcast.rss"
    ]
   ]
}
```

URLs that are not allowed (currently all URLs that don't start with either http or https) are rewritten to the empty string and are ignored by the Webservice.

Get Subscription Changes

GET /api/2/subscriptions/(*username*)**/**
*deviceid***.json**

- Requires HTTP authentication
- Since 2.0

This API call retrieves the subscription changes since the timestamp provided in the since parameter. Its value SHOULD be timestamp value from the previous call to this API endpoint. If there has been no previous call, the cliend SHOULD use 0.

The response format is the same as the upload format: A dictionary with two keys "add" and "remove" where the value for each key is a list of URLs that should be added or removed. The timestamp SHOULD be stored by the client in order to provide it in the since parameter in the next request.

Example response:

In case nothing has changed, the server returns something like the following JSON content.

```
{
   "add": [],
   "remove": [],
   "timestamp": 12347
}
```

Parameters

- **username** – username for which subscriptions should be returned
- **deviceid** – see *Devices*

Query Parameters

- **since** – the `timestamp` value of the last response

Episode Actions API

The episode actions API is used to synchronize episode-related events between individual devices. Clients can send and store events on the webservice which makes it available to other clients. The following types of actions are currently accepted by the API: download, play, delete, new. Additional types can be requested on the Mailing List.

Example use cases

- Clients can send `download` and `delete` events so that other clients know where a file has already been downloaded.
- Clients can send `play` events with `position` information so that other clients know where to start playback.
- Clients can send `new` states to reset previous events. This state needs to be interpreted by receiving clients and does not delete any information on the webservice.

Episode Action Types

- download
- delete
- play

- new
- flattr

Upload Episode Actions

POST /api/2/episodes/(*username*)**.json**

- Requires HTTP authentication
- Since 2.0

Upload changed episode actions. As actions are saved on a per-user basis (not per-device), the API endpoint is the same for every device. For logging purposes, the client can send the device ID to the server, so it appears in the episode action log on the website.

Example request:

```
POST /api/2/episodes/some-user.json

[
  {
   "podcast": "http://example.com/feed.rss",
   "episode": "http://example.com/files/s01e20.mp3",
   "device": "gpodder_abcdef123",
   "action": "download",
   "timestamp": "2009-12-12T09:00:00"
  },
  {
   "podcast": "http://example.org/podcast.php",
   "episode": "http://ftp.example.org/foo.ogg",
   "action": "play",
   "started": 15,
   "position": 120,
   "total":  500
  }
]
```

Request JSON Object

- **podcast** – The feed URL to the podcast feed the episode belongs to (required)
- **episode** – The media URL of the episode (required)
- **device** – The device ID on which the action has taken place (see *Devices*)
- **action** – One of: download, play, delete, new (required)
- **timestamp** – A UTC timestamp when the action took place, in ISO 8601 format
- **started** – Only valid for “play”. the position (in seconds) at which the client started playback. Requires position and total to be set.
- **position** – Only valid for “play”. the position (in seconds) at which the client stopped playback
- **total** – Only valid for “play”. the total length of the file in seconds. Requires position and started to be set.

Example response:

The return value is a JSON dictionary containing the timestamp and a list of URLs that have been rewritten (sanitized, see bug:747 and bug:862) as a list of tuples with the key "update_urls". The client SHOULD parse this list and update the local subscription and episode list accordingly (the server only sanitizes the URL, so the semantic "content" should stay the same and therefore the client can simply update the URL value locally and use it for future updates. An example result with update_urls:

```
HTTP/1.1 200 OK

{
    "timestamp": 1337,
    "update_urls": [
        ["http://feeds2.feedburner.com/LinuxOutlaws?format=xml",
         "http://feeds.feedburner.com/LinuxOutlaws"],
        ["http://example.org/episode.mp3 ",
         "http://example.org/episode.mp3"]
    ]
}
```

URLs that are not allowed (currently all URLs that contain non-ASCII characters or don't start with either http or https) are rewritten to the empty string and are ignored by the Webservice.

Get Episode Actions

GET /api/2/episodes/(*username*)**.json**

- Requires HTTP authentication
- Since 2.0

Timestamps: The result is a list of all episode actions that were uploaded since the timestamp given in the since parameter (regardless of the action timestamp itself). The timestamp SHOULD be the value returned by the previous episode retrieve request. If no since value is given, ALL episode actions for the given user are returned. Please note that this could be a potentially long list of episode actions, so clients SHOULD provide a since value whenever possible (e.g. when uploads have been taken place before).

Example response:

The format of the action list is the same as with the action upload request, but the format is a bit different so that the server can send the new timestamp (that the client SHOULD save and use for subsequent requests):

```
HTTP/1.1 200 OK

{
    "actions": [],
    "timestamp": 12345
}
```

Client implementation notes: A client can make use of the device variant of this request when it is assigned a single device id. When adding a podcast to the client (without synching the subscription list straight away), the variant with the podcast URL can be used. The first variant (no parameters at all) can be used as a kind of "burst" download of all episode actions, but should be used as little as possible (e.g. after a re-install, although even then, the device-id parameter could be more useful).

Query Parameters

- **podcast** (*string*) – The URL of a Podcast feed; if set, only actions for episodes of the given podcast are returned
- **device** (*string*) – A Device ID; if set, only actions for the given device are returned

- **since** (*int*) – Only episode actions since the given timestamp are returned
- **aggregated** (*bool*) – If true, only the latest actions is returned for each episode (added in 2.1)

Podcast Lists API

Podcast Lists are used to collect podcasts about one topic. On the website, podcast lists are available at https://gpodder.net/lists/

Create Podcast List

POST /api/2/lists/(*username*)**/create.**
format

- requires authenticaton
- since 2.10

Query Parameters

- **title** – url-encoded title

Parameters

- **username** – username for which a new podcast list should be created
- **format** – see *Formats*

The list content is sent in the request body, in the format indicates by the format extension

The server then generates a short name for the list from the title given in the Request. For example, from the title "My Python Podcasts" the name "my-python-podcasts" would be generated.

Status Codes

- 409 Conflict – if the the user already has a podcast list with the (generated) name
- 303 See Other – the podcast list has been created at the URL given in the Location header

Get User's Lists

GET /api/2/lists/(*username*)**.json**

- since 2.10

Example response:

```
HTTP/1.1 200 OK

[
    {
        "title": "My Python Podcasts",
        "name": "my-python-podcasts",
        "web": "http://gpodder.net/user/a-user/lists/my-python-podcasts"
    }
]
```

Status Codes

- 200 OK – the list of lists is returned
- 404 Not Found – the user was not found

Get a Podcast List

GET /api/2/lists/(*username*)**/list/**
listname.*format*

- since 2.10

Parameters

- **`username`** – username to which the list belongs
- **`listname`** – name of the requested podcast list
- **`format`** – see *Formats*

Status Codes

- 200 OK – the podcast list is returned in in the requested format
- 404 Not Found – if the user or the list do not exist

Update a Podcast List

PUT /api/2/lists/(*username*)**/list/**
listname.*format*

- requires authentication
- since 2.10

Parameters

- **`username`** – username to which the list belongs
- **`listname`** – name of the requested podcast list
- **`format`** – see *Formats*

Status Codes

- 404 Not Found – if the user or the list do not exist
- 204 No Content – if the podcast list has been created / updated

Delete a Podcast List

DELETE /api/2/lists/(*username*)**/list/**
listname.*format*

- requires authentication
- since 2.10

Parameters

- **username** – username to which the list belongs
- **listname** – name of the requested podcast list
- **format** – see *Formats*

Status Codes

- 404 Not Found – if the user or the list do not exist
- 204 No Content – if the podcast list has been deleted

Settings API

Clients can store settings and retrieve settings as key-value-pairs, which are attached to either account, device, podcast or episode.

Keys are the names of the settings and are supposed to be strings. Values can be any valid JSON objects.

Known Settings

Although settings are primarily used to exchange settings between clients, some of them also trigger some behavior on the website.

Account

- `public_profile`: when set to False, sets all podcasts to private (as on http://gpodder.net/account/privacy, currently deactivated via API)
- `store_user_agent`: allow gpodder.net to store the User-Agent for each device (default: true)
- `public_subscriptions`: default "public" value for subscriptions (default: true)

Episode

- `is_favorite`: flags the episode as favorite (can be done on any episode-page)

Podcast

- `public_subscription`: when set to False, sets the subscription to this podcast to private (as on http://gpodder.net/account/privacy or any podcast-page, currently deactivated via API)

Save Settings

POST /api/2/settings/(*username*)**/**
scope.**json**

- Requires Authentication
- Since 2.4

Example request:

```
{
    "set": {"setting1": "value1", "setting2": "value2"},
    "remove": ["setting3", "setting4"]
}
```

Parameters

- **`scope`** – one of account, device, podcast, episode

Query Parameters

- **`podcast`** (*`string`*) – Feed URL of a podcast (required for scope podcast and episode)
- **`device`** – Device id (see *Devices*, required for scope device)
- **`episode`** – media URL of the episode (required for scope episode)

set is a dictionary of settings to add or update; remove is a list of keys that shall be removed from the scope.

Example response:

The response contains all settings that the scope has after the update has been carried out.

Get Settings

`GET /api/2/settings/`(*username*)**`/`**
scope.**`json`**

- Requires Authentication
- Since 2.4

Parameters

- **`scope`** – one of account, device, podcast, episode

Query Parameters

- **`podcast`** (*`string`*) – Feed URL of a podcast (required for scope podcast and episode)
- **`device`** – Device id (see *Devices*, required for scope device)
- **`episode`** – media URL of the episode (required for scope episode)

Example response:

The response contains all settings that the scope currently has

```
{
    "setting1": "value1",
    "setting2": "value2"
}
```

Favorites API

Get Favorite Episodes

`GET /api/2/favorites/`(*username*)**`.json`**

- Requires Authentication
- Since 2.4 (added released in 2.6)

The response is a list of all favorite episodes, as they can be seen on http://gpodder.net/favorites/

Example response:

```
HTTP/1.1 200 OK

[
   {
     "title": "TWiT 245: No Hitler For You",
     "url": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.com/
↪twit/twit0245.mp3",
     "podcast_title": "this WEEK in TECH - MP3 Edition",
     "podcast_url": "http://leo.am/podcasts/twit",
     "description": "[...]",
     "website": "http://www.podtrac.com/pts/redirect.mp3/aolradio.podcast.aol.com/
↪twit/twit0245.mp3",
     "released": "2010-12-25T00:30:00",
     "mygpo_link": "http://gpodder.net/episode/1046492"
    }
]
```

Parameters

- **`username`** – username for which the favorites should be returned

Device Synchronization API

Get Sync Status

`GET /api/2/sync-devices/`(*username*)**`.json`**

- requires authentication
- since 2.10

Example response:

```
{
  "synchronized": [
       ["notebook", "n900"],
       ["pc-home", "pc-work"],
    ],
  "not-synchronized": [
       "test-pc", "netbook"
    ]
}
```

Parameters

- **`username`** – username for which the sync status is requested

Start / Stop Sync

POST /api/2/sync-devices/(*username*)**.json**

- requires authentication
- since 2.10

Example request:

```
{
  "synchronize": [
       ["notebook", "netbook"]
     ],
  "stop-synchronize": ["pc-work"]
}
```

Sets up / stops synchronization between devices. The synchronization status is sent as a response

Example status:

```
{
  "synchronized": [
       ["notebook", "netbook", "n900"]
   ],
  "not-synchronized": [
       "test-pc", "pc-work", "pc-home"
    ]
}
```

Parameters

- **username** – username for which the sync status is requested

Client Parametrization

The client configuration file is located at http://gpodder.net/clientconfig.json and contains information that clients should retrieve before making requests to the APIs.

If a client cannot retrieve and process this file (either temporarily or permanently), it can assume the default values provided below. However, the URLs in the file might reflect changed URLs and/or mirror servers. If a client decides to permanently ignore this file, it might hit an outdated URL or an overloaded server.

Commented Example

```
{
    "mygpo":  {
        "baseurl": "http://gpodder.net/"
    }

    "mygpo-feedservice": {
        "baseurl": "http://mygpo-feedservice.appspot.com/"
    }

    "update_timeout": 604800,
}
```

- `mygpo/baseurl`: URL to which the gpodder.net API Endpoints should be appended
- `mygpo-feedservice/baseurl`: Base URL of the gpodder.net feed service
- `update_timeout`: Time in seconds for which the values in this file can be considered valid.

API 1 (Deprecated)

This page describes version 1 of the gpodder.net API which is deprecated in favor of version 2.

There are two different APIs for different target audiences:

- The Simple API targets developers who want to write quick integration into their existing applications
- The Advanced API targets developers who want tight integration into their applications (with more features)

Reference Implementation

https://github.com/gpodder/mygpoclient

Legacy API

This API is for support of old clients (gPodder 2.1 and earlier) and should never be used in new code. It's just here for reference of what the server implementation should provide.

upload

- Request URI: `/upload` * Parameters: `username` (the e-mail address of the user), `password` (the user password in plaintext), `action` (always set to `update-subscriptions`), `protocol` (always set to `0`), `opml` (a HTTP file upload field that contains the subscription list as OPML file)

Parses the given OPML file and compares the user's default subscription list with the entries from the OPML file. For every new subscription, the server automatically creates a "subscribe" event, and for every removed subscription, the server automatically generates a "unsubscribe" event. The user's default subscription list on the server will match the uploaded OPML file after the request returns `@SUCCESS`.

Possible response values (these are potentially contained within surrounding text, e.g. HTML):

- `@GOTOMYGPODDER`: The website it opened in the web browser and the user gets the message "Please have a look at the website for more information."
- `@SUCCESS`: The subscription has been successfully uploaded.
- `@AUTHFAIL`: The supplied username and password combination is wrong.
- `@PROTOERROR`: There has been an error in the request format (wrong OPML format, wrong parameters, etc..).
- ''None of the above": This is an "unknown" response, and the client displays an error message to the user.

getlist

- Request URI: `/getlist`
- Parameters: `username` (the e-mail address of the user), `password` (the user password in plaintext)

This returns the main subscription list of the user as OPML content.

register

- Request URI: `/register`
- Parameters: None

This web page is to be opened in a web browser if the user choses to create a new user account.

toplist.opml

- Request URI: `/toplist.opml`
- Parameters: None

This should return an OPML file with the top 50 podcasts. This is the same as the Simple API endpoint `/toplist/50.opml` and can (obviously) utilize the same code.

Simple API

The Simple API provides a way to upload and download subscription lists in bulk. This allows developers of podcast-related applications to quickly integrate support for the web service, as the only

- Synchronization of episode status fields is `not` supported
- This API uses more bandwith than the advanced API
- The client can be stateless
- The client can be low-powered - subscribe/unsubscribe events are calculated on the server-side

Downloading subscription lists

- Request: `GET /subscriptions/''{username}''/''{device_id}''.opml`
- Request: `GET /subscriptions/''{username}''/''{device_id}''.json`
- Request: `GET /subscriptions/''{username}''/''{device_id}''.txt`
- Requires HTTP authentication

Get a list of subscribed podcasts for the given user. The first variant returns the content as OPML feed, the second variant as list of feed URLs in JSON format. The third variant returns the list of URLs (one per line) as simple plaintext.

- Example: `GET /subscriptions/bob/asdf.opml` (Download bob's list for device ID `asdf` as OPML)

In case of errors, the following HTTP status codes are used:

- `401` Invalid user
- `404` Invalid device ID
- `400` Invalid format (e.g. broken OPML)

Uploading subscription lists

- Request: `PUT /subscriptions/''{username}''/''{device_id}''.opml`
- Request: `PUT /subscriptions/''{username}''/''{device_id}''.json`
- Request: `PUT /subscriptions/''{username}''/''{device_id}''.txt`
- Requires HTTP authentication

Upload the current subscription list of the given user to the server. The data should be provided either in OPML, JSON or plaintext (one URL per line) format, and should be uploaded just like a normal PUT request (i.e. in the body of the request).

For successful updates, the implementation `always` returns the status code `200` and the ''empty string" (i.e. an empty HTTP body) as a result, any other string should be interpreted by the client as an (undefined) error.

Defined errors are as follows (in this case, the body that is received from the server ''might" be a user-friendly description of the error):

- `401` Invalid user
- `400` Invalid format (cannot parse OPML or JSON)

In case the device does not exist for the given user, it is automatically created. If clients want to determine if a device exists, you have to to a GET request on the same URL first and check for a the 404 status code (see above).

- Example: `PUT /subscriptions/john/e9c4ea4ae004efac40.txt` (Upload john's list for that device as text file)

Downloading podcast toplists

- Request: `GET /toplist/''{number}''.opml`
- Request: `GET /toplist/''{number}''.json`
- Request: `GET /toplist/''{number}''.txt`
- Does ''not" require authentication (''public content")

The `number` field might be any value in the range 1..100 (inclusive both boundaries). An example request looks like:

- `GET /toplist/50.json` - Get the top 50 list in JSON format

Download a list of podcasts, sorted in descending order (more popular podcasts first) in different formats. The OPML and TXT formats do not add any information about the (absolute and relative) popularity for each podcast, only the ordering can be considered. The JSON format includes a more detailed list, usable for clients that want to display a detailed toplist or post-process the toplist:

```
[{"url": "http://twit.tv/node/4350/feed",
  "title": "FLOSS Weekly",
  "description": "Free, Libre and Open Source Software with Leo.",
  "subscribers": 4711,
 },
 {"url": "http://feeds.feedburner.com/LinuxOutlaws",
  "title": "The Linux Outlaws",
  "description": "A podcast about Linux with Dan and Fab.",
  "subscribers": 1337,
 }]
```

All shown keys must be provided by the server. The `description` field may be set to the empty string in case a description is not available. The `title` field may be set to the URL in case a title is not available.

Downloading podcast suggestions

- Request: `GET /suggestions/''{number}''.opml`
- Request: `GET /suggestions/''{number}''.json`
- Request: `GET /suggestions/''{number}''.txt`
- Requires HTTP authentication

The `number` field might be any value in the range 1..100 (inclusive both boundaries). An example request looks like:

- `GET /suggestions/10.opml` - Get 10 suggestions in OPML format

Download a list of podcasts that the user has not yet subscribed to (by checking ''all" server-side subscription lists) and that might be interesting to the user based on existing subscriptions (again on ''all" server-side subscription lists).

The TXT format is a simple URL list (one URL per line), and the OPML file is a "standard" OPML feed. The JSON format looks as follows:

```
[{"url": "http://twit.tv/node/4350/feed",
  "title": "FLOSS Weekly",
  "description": "Free, Libre and Open Source Software with Leo."
 },
 {"url": "http://feeds.feedburner.com/LinuxOutlaws",
  "title": "The Linux Outlaws",
  "description": "A podcast about Linux with Dan and Fab."
 }]
```

The server does not specify the "relevance" for the podcast suggestion, and the client application ''SHOULD" filter out any podcasts that are already added to the client application but that the server does not know about yet (although this is just a suggestion for a good client-side UX).

Searching for podcasts

- Request: `GET /search.opml?q=''{query}`"
- Request: `GET /search.json?q=''{query}`"
- Request: `GET /search.txt?q=''{query}`"
- Does ''not" require authentication (''public content")

Carries out a service-wide search for podcasts that match the given query. Returns a list of podcasts.

The format of the search results is the same as for podcast suggestions. See there for the exact format.

Advanced API

The Advanced API provides more flexibility and enhanced functionality for applications that want a tighter integration with the web service. A reference implementation will be provided as part of the gPodder source code (and gPodder will make use of that reference implementation).

- The client has to persist the synchronization state locally
- Only changes to subscriptions are uploaded and downloaded
- Synchronization of episode status fields is supported in this API
- Only JSON is used as the data format to ease development

Add/remove subscriptions

- Request: `POST /api/1/subscriptions/''{username}''/''{device_id}''.json`
- Requires HTTP authentication

Update the subscription list for a given device. Only deltas are supported here. Timestamps are not supported, and are issued by the server.

Example JSON upload data:

```
{"add": ["http://example.com/feed.rss", "http://example.org/podcast.php"],
 "remove": ["http://example.net/foo.xml"]}
```

The server returns a timestamp/ID that can be used for requesting changes since this upload in a subsequent API call (see below):

```
{"timestamp": 12345, "update_urls": []}
```

`Update 2010-01-07`: In addition, the server MUST send any URLs that have been rewritten (sanitized, see [[bug:747]]) as a list of tuples with the key "update_urls". The client SHOULD parse this list and update the local subscription list accordingly (the server only sanitizes the URL, so the semantic "content" should stay the same and therefore the client can simply update the URL value locally and use it for future updates. An example result with update_urls:

```
{"timestamp": 1337,
 "update_urls": [
  ["http://feeds2.feedburner.com/LinuxOutlaws?format=xml",
   "http://feeds.feedburner.com/LinuxOutlaws"],
  ["http://example.org/podcast.rss ",
   "http://example.org/podcast.rss"]]}
```

`Update 2010-01-17`: URLs that are not allowed (currently all URLs that don't start with either http or https) are rewritten to the empty string and are ignored by the Webservice.

Retrieving subscription changes

- Request: `GET /api/1/subscriptions/''{username}''/''{device_id}''.json?since=''{timestamp}`"
- Requires HTTP authentication

This API call retrieves only the changes since the last upload (the last upload is determined by the "since" parameter, which usually is taken from the return value of a previous update call). The response format is the same as the upload format, i.e. JSON: A dictionary with two keys "add" and "remove" where the value for each key is a list of URLs that should be added or removed. There is one additional key ("timestamp") that is provided by the server that will tell the client the next value for the "since" parameter in case the client wants to issue another GET request in the future without uploading data.

In case nothing has changed, the server returns something like the following JSON content (in this case, the client SHOULD store the timestamp and use it for future requests):

```
{"add": [], "remove": [], "timestamp": 12347}
```

Uploading episode actions

- Request: `POST /api/1/episodes/''{username}''.json`
- Requires HTTP authentication

Upload changed episode actions. As actions are saved on a per-user basis (not per-device), the API endpoint is the same for every device. For logging purposes, the client can send the device ID to the server, so it appears in the episode action log on the website.

Example JSON upload data:

```
[{"podcast": "http://example.com/feed.rss",
  "episode": "http://example.com/files/s01e20.mp3",
  "device": "gpodder_abcdef123",
  "action": "download",
  "timestamp": "2009-12-12T09:00:00"},
 {"podcast": "http://example.org/podcast.php",
  "episode": "http://ftp.example.org/foo.ogg",
  "action": "play",
  "position": "01:00:00"}]
```

Possible keys:

- `podcast` (required): The URL to the podcast feed the episode belongs to
- `episode` (required): The download URL/GUID of the episode
- `device` (optional): The device ID on which the action has taken place
- `action` (required): One of: download, play, delete, new
- `timestamp` (optional): An optional timestamp when the action took place, in [http://en.wikipedia.org/wiki/ISO_8601 ISO 8601 format] - **The timestamp MUST be in the UTC time zone**
- `position` (optional): Only valid for "play": the current play position in `HH:MM:SS` format

The return value is a JSON dictionary containing the timestamp (that can be used for retrieving changed episode actions later on):

```
{"timestamp": 12345,
 "update_urls": [] }
```

The client SHOULD save this timestamp if it wants to retrieve episode actions in the future in order to save bandwith and CPU time on the server.

`Update 2010-02-23:` In addition, the server MUST send any URLs that have been rewritten (sanitized, see [[bug:747]] and [[bug:862]]) as a list of tuples with the key "update_urls". The client SHOULD parse this list and update the local subscription and episode list accordingly (the server only sanitizes the URL, so the semantic "content" should stay the same and therefore the client can simply update the URL value locally and use it for future updates. An example result with update_urls:

```
{"timestamp": 1337,
 "update_urls": [
  ["http://feeds2.feedburner.com/LinuxOutlaws?format=xml",
   "http://feeds.feedburner.com/LinuxOutlaws"],
  ["http://example.org/episode.mp3 ",
   "http://example.org/episode.mp3"]]}
```

URLs that are not allowed (currently all URLs that contain non-ASCII characters or don't start with either http
or https) are rewritten to the empty string

and are ignored by the Webservice.

Retrieving episode actions

- Request: `GET /api/1/episodes/''{username}''.json`
- Request: `GET /api/1/episodes/''{username}''.json?podcast=''{url}`”
- Request: `GET /api/1/episodes/''{username}''.json?device=''{device-id}`”
- Request: `GET /api/1/episodes/''{username}''.json?since=''{timestamp}`”
- Request: `GET /api/1/episodes/''{username}''.json?podcast=''{url}''&since=''{timestamp}`”
- Request: `GET /api/1/episodes/''{username}''.json?device=''{device-id}''&since=''{timestam`
- Requires HTTP authentication

Download changed episode actions. The result is a list of all new episode actions since the given timestamp. The timestamp is the value returned by the episode upload request. The first three variants (without the “since” parameter) downloads ALL episode actions for the given user. Please note that this could be a potentially long list of episode actions, so clients SHOULD prefer the “since” variants whenever possible (e.g. when uploads have been taken place before).

The format of the action list is the same as with the action upload request, but the format is a bit different so that the server can send the new timestamp (that the client SHOULD save and use for subsequent requests):

```
{"actions": ''(list of episode actions here - see above for details)'',
 "timestamp": 12345}
```

There are two additional variants that take either a podcast URL or a device ID and returns only episode actions related to the given podcast or device. In the case of the device ID, all podcasts to which the device is currently subscribe to, are combined, and episode actions for these are added.

‘’Client implementation notes:” A client can make use of the device variant of this request when it is assigned a single device id. When adding a podcast to the client (without synching the subscription list straight away), the variant with the podcast URL can be used. The first variant (no parameters at all) can be used as a kind of “burst” download of all episode actions, but should be used as little as possible (e.g. after a re-install, although even then, the device-id parameter could be more useful).

(Re)naming devices and setting the type

- Request: `POST /api/1/devices/''{username}''/''{device-id}''.json`
- Requires HTTP authentication

Set a new name for the device ID. The device ID is normally generated by the client application, but for viewing the device on the web and for managing subscriptions, it’s easier to provide a “human-readable” name. The client application should do this using this API call. It can also provide the type of device, so that a special icon can be shown in the web UI. Only the keys that are supplied will be updated.

```
{"caption": "gPodder on my Lappy", "type": "laptop"}
```

Possible keys:

- `caption`: The new label for the device
- `type`: The type of the device. (Possible values: desktop, laptop, mobile, server, other)

Getting a list of devices

- Request: `GET /api/1/devices/''{username}''.json`
- Requires HTTP authentication

Returns the list of devices that belong to a user. This can be used by the client to let the user select a device from which to retrieve subscriptions, etc..

```
[{"id": "abcdef",
  "caption": "gPodder on my Lappy",
  "type": "laptop",
  "subscriptions": 27},
 {"id": "09jc23caf",
  "caption": "",
  "type": "other",
  "subscriptions": 30},
 {"id": "phone-au90f923023.203f9j23f",
  "caption": "My Phone",
  "type": "mobile",
  "subscriptions": 5}]
```

API 3 (Draft)

This is *a draft* for version 3 of the public API of gpodder.net.

Client Access

- Require client keys to identify clients, get a communication channel to developers
- Clients must send a valid User-Agent string
- API usage free for open source clients
- Quota for non-open clients – higher quota if more features are implemented; paid quota increase

Proposed Changes to API 2

- The classification between Simple and Advanced API is dropped
- A separate domain name for API requests will be used (something like api.gpodder.net, see *API Parametrization*
- The `/api/` prefix has been dropped (in favor of a API domain name) and a version prefix (`/3/`) has been added for all endpoints
- Device-Data and Settings are updated with PUT instead of POST (because they overwrite existing data)
- Things to consider: Evolving HTTP APIs

Additional Ideas

- Use authentication protocol OAuth2

Open Questions

- What's the best way of documenting a REST API? http://stackoverflow.com/questions/898321/standard-methods-for-documenting-a-restful-api http://answers.oreilly.com/topic/1390-how-to-document-restful-web-services/
- Specify deprecation guideline + timeline for when the old API will stop functioning

API Changes

This page lists changes to the Advanced API. The current version is 2.11. This versioning scheme has been introduced in Bug 1273.

Version 2.11

- added *Get All Subscriptions*.

Version 2.10

- added Authentication API (bug 1397)
- added Device Synchronization API (bug 1266)
- added Podcast Lists API (bug 1402)
- added include_actions parameter to Device Update API (bug 1419)

Version 2.9

- added XML format to some Simple API requests (bug 1362, bug 1383)

Version 2.8

- added JSONP as a format to Simple API requests (bug 1302)

Version 2.7

- added *API Parametrization*

Version 2.6

- added *released* to *Retrieve Episode Data* and *Get Favorite Episodes*

Version 2.5

- added "Subscribers Last Week" to *Retrieve Podcast Data* (bug 1188)

Version 2.4

- added *Save Settings* (bug 1082)
- added *Get Settings*
- added *Get Favorite Episodes*

Version 2.3

- added *Get Device Updates*

Version 2.2

- added *Retrieve Top Tags*
- added *Retrieve Podcasts for Tag*
- added *Retrieve Podcast Data*
- added *Retrieve Episode Data*

Version 2.1

- added `aggregated=true` to *Get Episode Actions* (bug 1030)

Version 2.0

- added *Upload Subscription Changes*
- added *Get Subscription Changes*
- added *Upload Episode Actions*
- added *Get Episode Actions*
- added *Update Device Data*
- added *List Devices*

1.4 Developer Documentation

Documentation in this section is intended for anyone who wants to contribute to gpodder.net

The sourcecode of the webservice gpodder.net is released as open source under the AGPLv3 and hosted at GitHub. Bugs can be reported at the gPodder Bugtracker.

1.4.1 Integrating Clients

If you want to integrate gpodder.net in some podcasting client, you might want to use one of the existing client libraries.

There are already several clients for different platforms which can be used as examples.

1.4.2 Contents

Installation

Dependencies

When no version number is indicated, it is advisable to install the current development version from the repository.

- Python >= 3.5
- PostgreSQL
- Redis

Basic setup

mygpo itself can be cloned from the repository:

```
git clone git://github.com/gpodder/mygpo.git
cd mygpo
```

On a Debian/Ubuntu based system, you can install dependencies with

```
make install-deps
```

Or:

```
sudo apt-get install libpq-dev libjpeg-dev zlib1g-dev libwebp-dev \
            build-essential python3-dev virtualenv libffi-dev redis postgresql
```

Now install additional dependencies locally:

```
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
pip install -r requirements-dev.txt    # for local development
pip install -r requirements-doc.txt    # for building docs
pip install -r requirements-setup.txt  # for a productive setup
pip install -r requirements-test.txt   # for running tests
```

That's it for the setup.

Configuration

Configuration of mygpo is done through environment variables. For development purposes you can set up a directory `envs/dev` and create a file for each variable that you want to set.

For a development configuration you will probably want to use the following

```
mkdir -p envs/dev
echo django.core.mail.backends.console.EmailBackend > envs/dev/EMAIL_BACKEND
echo secret > envs/dev/SECRET_KEY
echo postgres://mygpo:mygpo@localhost/mygpo > envs/dev/DATABASE_URL
echo True > envs/dev/DEBUG
echo "127.0.0.1" > envs/dev/INTERNAL_IPS
```

(continues on next page)

(continued from previous page)

```
mkdir -p /tmp/mygpo-test-media
echo /tmp/mygpo-test-media > envs/dev/MEDIA_ROOT
```

You can perform this configuration with

```
make dev-config
```

See *Configuration* for further information.

Database Initialization

Now to initialize the DB:

First run the commands from *PostgreSQL Setup*. Then

```
envdir envs/dev python manage.py migrate
```

..and here we go:

```
envdir envs/dev python manage.py runserver
```

Accessing the dev server from other devices

Sometimes you might want to access the server from another machine than localhost. In that case, you have to pass an additional argument to the runserver command of manage.py, like this:

```
envdir envs/dev python manage.py runserver 0.0.0.0:8000
```

Beware, though, that this will expose the web service to your all networks that your machine is connected to. Apply common sense and ideally use only on trusted networks.

Updating derived data

Certain data in the database is only calculated when you run special commands. This is usually done regularly on a production server using cron. You can also run these commands regularly on your development machine:

```
envdir envs/dev python manage.py update-toplist
envdir envs/dev python manage.py update-episode-toplist

envdir envs/dev python manage.py feed-downloader
envdir envs/dev python manage.py feed-downloader <feed-url> [...]
envdir envs/dev python manage.py feed-downloader --max <max-updates>
envdir envs/dev python manage.py feed-downloader --random --max <max-updates>
envdir envs/dev python manage.py feed-downloader --toplist --max <max-updates>
envdir envs/dev python manage.py feed-downloader --update-new --max <max-updates>
```

or to only do a dry run (this won't do any web requests for feeds):

```
envdir envs/dev apython manage.py feed-downloader --list-only [other parameters]
```

Maintaining publisher relationships with user accounts

To set a user as publisher for a given feed URL, use:

```
envdir envs/dev python manage.py make-publisher <username> <feed-url> [...]
```

Web-Server

Django comes with a development webservice which you can run from the mygpo directory with

```
envdir envs/dev python manage.py runserver
```

If you want to run a production server, check out Deploying Django.

PostgreSQL Setup

Use the following to set up a local PostgreSQL.

```
CREATE USER mygpo WITH PASSWORD 'mygpo';
ALTER USER mygpo CREATEDB;  -- required for creating test database
CREATE DATABASE mygpo;
CREATE DATABASE test_mygpo;
GRANT ALL PRIVILEGES ON DATABASE mygpo to mygpo;
GRANT ALL PRIVILEGES ON DATABASE test_mygpo to mygpo;
ALTER DATABASE mygpo OWNER TO mygpo;
ALTER DATABASE test_mygpo OWNER TO mygpo;
ALTER ROLE mygpo SET statement_timeout = 5000;
```

Client Libraries

This is a list of libraries for accessing the APIs of gpodder.net and feeds.gpodder.net.

Please consult the *Integration Guide* before integrating the gpodder.net API in your application.

Library	Programming Language	Features	Latest Version	Used by	Status
mygpoclient	Python	gpodder.net (nearly full support)	1.6	gpodder	Active
libmygpo-qt	C++/Qt	gpodder.net (nearly full support)	1.0.7	Amarok,Clementine	Active
mygpoclient-java	Java	gpodder.net (partial) and feeds.gpodder.net (full)	0.0	Podder,Detlef	Active
mygpoddERlib	Java	gpodder.net (partial support)	0.0	gpodroid	Abandoned (?)

Configuration

The following configuration parameters can be set through environment variables.

General

- `ADMINS` - corresponds to Django's ADMINS setting. Specified as `Name <email@host.com>`. Multiple entries can be separated by whitespace.
- `DEBUG` - Debug flag, see Django's DEBUG setting
- `DEFAULT_BASE_URL` - base URL for creating URLs, eg `https://gpodder.net`
- `GOOGLE_ANALYTICS_PROPERTY_ID` - Google Analytics Property ID
- `MAINTENANCE` - Maintenance flag
- `SECRET_KEY` - see Django's SECRET_KEY setting
- `STAFF_TOKEN` - token which can be appended to URLs to access staff-only pages
- `SUPPORT_URL` - URL where users can get support

Advertising

- `ADSENSE_CLIENT` - Google AdSense Client ID
- `ADSENSE_SLOT_BOTTOM`- Ad for the ad slot on the bottom of the page
- `PODCAST_AD_ID` - Database Id of the podcast which is currently advertising

Celery Task Queue

- `BROKER_POOL_LIMIT` - corresponds to Celery's broker_pool_limit setting. Specifies the maximum number of connections that can be open in the connection pool.
- `BROKER_URL` - corresponds to Celery's broker_url setting. Specifies the URL / connection string to the broker.

Caching

The following settings correspond to Django's CACHE setting.

- `CACHE_BACKEND` - Django cache backend
- `CACHE_LOCATION` - Location of the cache

Database

- `DATABASE_URL` - Database connection string, see syntax and options.

Emails

- `DEFAULT_FROM_EMAIL` - From address for outgoing emails, see Django's DEFAULT_FROM_EMAIL setting.

Search

- `ELASTICSEARCH_SERVER` - `host:port` of the Elasticsearch server
- `ELASTICSEARCH_TIMEOUT` - timeout in seconds for queries to the Elasticsearch server
- `QUERY_LENGTH_CUTOFF` - Maximum non-whitespace length of search query

Directory

- `DIRECTORY_EXCLUDED_TAGS` - space-separated list of tags that should be excluded from the podcast directory
- `SEARCH_CUTOFF` - minimum search rank (between 0 and 1, default 0.3) below which results are excluded. See Django's documentation on Weighting queries

Feeds

- `FLICKR_API_KEY` - Flickr API key
- `SOUNDCLOUD_CONSUMER_KEY` - Soundcloud Consumer key

Logging

- `SERVER_EMAIL` - email address from which error mails are sent, see Django's SERVER_EMAIL setting
- `LOGGING_CELERY_HANDLERS` - space separated list of logging handlers for Celery log events
- `LOGGING_DJANGO_HANDLERS` - space separated list of logging handlers for Django log events
- `LOGGING_MYGPO_HANDLERS` - space separated list of logging handlers for mygpo log events
- `LOGGING_FILENAME` - filename for filesystem logs
- `LOGGING_DIR_GUNICRON` - directory for gunicron to log into
- `OPBEAT_APP_ID` - Opbeat App ID
- `OPBEAT_ORGANIZATION_ID` - Opbeat Organization ID
- `OPBEAT_SECRET_TOKEN` - Opbeat Secret Token

Social Login

- `GOOGLE_CLIENT_ID` - Google Client ID
- `GOOGLE_CLIENT_SECRET` - Google Client Secret

API

- `MAX_EPISODE_ACTIONS` - maximum number of episode actions that the API will return in one *GET* request.

Jupyter Notebook

You can use Jupyter Notebooks during development for exploring data and prototyping methods.

To do so, follow these steps

- Make sure you have all requirements from `requirements-dev.txt` installed.
- Run `make notebook`, which will start the notebook and open it in the browser .
- Navigate to the directory `notebooks` (listed in *.gitignore*) and create a new notebook.
- Use the following code in the first cell to setup your environment

  ```
  MYPROJECT = '/path/to/mygpo'
  import os, sys
  sys.path.insert(0, MYPROJECT)
  os.environ.setdefault("DJANGO_SETTINGS_MODULE", "local_settings.py")
  import django
  django.setup()
  ```

Translations

Translations for gpodder.net are managed on Transifex, in the gpodder.net project.

Importing translations

To import translations from transifex run the following

```
tx pull -a
make update
```

Upload new source strings

When changing the source code so that there are new internationalized strings that need to be translated run the following commands to upload these to Transifex.

```
make update-po
tx push -s
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search