The FeedBlitz API

Version 3.0
March 28, 2017

Easily enabling powerful integrated email subscription management services using XML, HTTPS and REST

The FeedBlitz API

Copyright

Disclaimer

About FeedBlitz

FeedBlitz is a service that monitors blogs, RSS feeds and Web URLs to provide greater reach for feed publishers. FeedBlitz takes all the headache out of converting feed and blog updates into email digests, delivered daily to subscribers' inboxes. FeedBlitz manages subscriptions, circulation tracking, testing, and is compatible with all major blogging platforms and services such as Blogger, TypePad and FeedBurner.

FeedBlitz enables end users to monitor any feed or blog, anonymously if they wish, regardless of whether the publisher of that feed is using FeedBlitz. FeedBlitz therefore provides a simple way for users to receive updates from their trusted sources using a familiar and ubiquitous medium - email.

Visit FeedBlitz online at www.feedblitz.com

Change History

8/9/2015 – v2.1.
Added SendMail, Mailing Metrics, RSS feed and Custom Field REST resources.
Added new intro sections, Simple API plugin development workflow.

7/26/2016 – v2.2
Added PUT and <mastertemplate> elements to the /template resource

8/17/2016 – v2.2.1
Added /customreg resource

8/29/2016 – v2.2.2
Added <conditions> elements to /newsflash; documented use of list ids to <suppressionlist> entity.

9/29/16 – v2.2.3
Fixed doc error for reports resource

11/19/16 – v2.3	Added /mailings resource
3/28/2017 – v3.0	Merged in the separate autoresponder API doc

Integrating FeedBlitz: APIs and More

As FeedBlitz has evolved, have its capabilities. Where there was once a single REST API, there are two basic APIs you can use:

- The Simple API – Based on simple URL approaches to perform common tasks
- The XML REST API – Complex and powerful features for platform integrators

In addition, there are FeedBlitz's core capabilities, available after logging in to the site. These include Triggers and Parsers, which deliver powerful integration capabilities between FeedBlitz and almost any third party product or service, via email.

Finally, select platform partners have access to an OEM API that enables FeedBlitz to be used as a white label solution. Contact FeedBlitz support for information about information about the requirements necessary for access to these capabilities.

Each of these integration approaches is targeted towards different audiences with different needs; they also demand differing levels of web programming expertise to use.

Which integration technique you need to use depends very much on what you want to do and the tools you're comfortable using. For the most common application, subscribing a visitor, there are many different ways to achieve this task

For the articles and updates on the FeedBlitz API, see http://www.feedblitz.com/category/api

Example: Building a Subscription Plugin for FeedBlitz

See http://www.feedblitz.com/the-feedblitz-api-workflow-for-building-a-plugin/

The goal of this example is to show you what API calls to make so you can programmatically add a subscriber's email (and other information) to a list at FeedBlitz. Let's assume you're developing a plugin for WordPress – using this example, any WP plugin developer should be able to integrate with FeedBlitz. The example below isn't limited to plugins, of course; it can work with any web site, product or service you can extend programmatically. You use simple Web requests, so it's pretty easy for a web developer to do.

Prerequisites

The user of your plugin will need:

- A FeedBlitz premium account
- At least one active mailing list
- An active FeedBlitz API key from https://www.feedblitz.com/f?pl_api

Workflow

Once you've validated the pre-reqs from the user, you:

1. Get a list of the publisher's active lists from the REST API.
2. Optionally get a list of custom fields and tags the publisher already has in FeedBlitz.
3. In your configuration UI, the publisher (i.e the user using you plugin):
    a. Picks a list that your plugin will subscribe visitors to.
    b. Optionally selects the custom fields to use, or specifies new ones using your UI.

When a visitor uses your plugin to join a list, you then:

1. Present your UI, performing any necessary validation;
2. Call the FeedBlitz Simple API to create the subscription
3. Assuming all is well, FeedBlitz starts the dual opt-in process to add the subscriber.

In Detail

All URLs below are simple HTTP GETs

1. Get a List of Active Mailing Lists

Use a quick call to the REST API as follows:

```
https://www.feedblitz.com/f.api/syndications?key=<api_key>&summary=1
```

The returned XML will contain one or more <syndication> elements, one for each list. Make sure you check the status elements and only present lists that are "ok." Grab the <id> and <name> elements within each; you will need to persist the <id> that the user selects. (Don't forget that you need to add "text/xml" to your Accept-Type header to make the REST API work.)

2. Get Active Custom Fields

If the publisher has custom fields defined in their account, it would be nice to offer those on your UI. Here's the REST API call:

```
https://www.feedblitz.com/f.api/fields?key=<api_key>
```

The returned XML may contain or more <field> elements. Grab the <name> and the <id> and the <uihidden> elements. FeedBlitz allows publishers to create hidden custom fields; publishers can use hidden fields to track campaigns, referring pages, etc. Hidden fields should not be displayed to end users (i.e. not shown to site visitors or potential subscribers); it's obviously ok to show them to the publisher during set up.

If there are no custom fields defined, you can also offer your own (e.g. “Name”); FeedBlitz will add them when you send the data to FeedBlitz at subscription time. Again, this a REST API call, so be sure to add “text/xml” to your Accept-Type header before accessing the resource.

Once the user has configure the FeedBlitz elements and the rest of your capabilities, save the relevant information and use it to build the user interface you want to present to site visitors.

3. Subscribe the Visitor

Configuration all done, potential subscribers will now be using your user interface on the publisher’s web site. Based on the saved configuration and user-supplied data, and making sure as best you can that the visiting user agent is not a bot (recommended to prevent spambots: use <script> to generate your UI on the fly, and make sure there’s a valid referrer header); call the simple subscription API as mentioned last week with the correct parameters:

```
https://www.feedblitz.com/f?SimpleApiSubscribe&key=<api_key>&email=<email>&listid=<lis
tid>
```

Optionally add tags and name / value pairs (see the Simple API reference documentation) and you’re done. With just two REST calls to assist in configuration, and one Simple API call per new subscriber. FeedBlitz handles the entire dual opt-in process from there on, as well as all subscriber management such as bounce handling, unsubscribes, reminding pending subscribers to activate, etc.

Code Sample

For plugin writers and others using PHP, here are some basic utility functions which will provide almost everything you need to interact with FeedBlitz to do the following:

- Get a list of active lists
- Start the dual opt-in subscription process for a list

The functions use cURL for maximum compatibility.

```
<?php
// helper function to access the FeedBlitz API via GET
function fbz_get_web_page( $url )
{
  $options = array(
    CURLOPT_RETURNTRANSFER => true, // return web page
    CURLOPT_HEADER => false,        // don't return headers
    CURLOPT_FOLLOWLOCATION => true, // follow redirects
    CURLOPT_ENCODING => "",         // handle all encodings
    CURLOPT_USERAGENT => "PHP FeedBlitz Web Form Handler", // a UA is required
    CURLOPT_AUTOREFERER => true,    // set referer on redirect
    CURLOPT_CONNECTTIMEOUT => 120,  // timeout on connect
    CURLOPT_TIMEOUT => 120,         // timeout on response
    CURLOPT_MAXREDIRS => 10,        // stop after 10 redirects
    CURLOPT_SSL_VERIFYPEER => false,
  );
  $ch = curl_init( $url );
  curl_setopt_array( $ch, $options );
  $content = curl_exec( $ch );
  $err = curl_errno( $ch );
  $errmsg = curl_error( $ch );
  curl_close( $ch );
  return $content;
}

// turns the returned XML into a PHP array
function fbz_prepare_results($page)
{
  $xml = simplexml_load_string($page, "SimpleXMLElement", LIBXML_NOCDATA);
  $json = json_encode($xml);
  $array = json_decode($json,TRUE);
  return $array;
}

// returns an array of list IDs, names and current subscriber counts
function fbz_get_lists($api_key, $lists)
{
  $url = "https://app.feedblitz.com/f.api/syndications?key=" . $api_key
."&summary=1&status=ok";
  $page = fbz_get_web_page($url);
  $array = fbz_prepare_results($page);
  if($array['rsp']['@attributes']['stat']=="ok")
  {
    $lists_array = $array['syndications']['syndication'];
    if ( ! empty( $lists_array ) ) {
      foreach ( $lists_array as $list_data ) {
        $lists[ $list_data['id'] ]['id']        = $list_data['id'];
        $lists[ $list_data['id'] ]['name']      = $list_data['name'];
        $lists[ $list_data['id'] ]['subscribers_count'] =
$list_data['subscribersummary']['subscribers'];
      }
    }
  }

  return $lists;
}

// get basic custom fields info for the account
function fbz_get_fields($api_key, $fields)
{
  $url = "https://app.feedblitz.com/f.api/fields?key=" . $api_key ."&status=ok";
  $page = fbz_get_web_page($url);
  $array = fbz_prepare_results($page);
  if($array['rsp']['@attributes']['stat']=="ok")
  {
    $fields_array = $array['fields']['field'];
    if ( ! empty( $fields_array ) ) {
      foreach ( $fields_array as $field_data ) {
        $fields[ $field_data['id'] ]['id']          = $field_data['id'];
        $fields[ $field_data['id'] ]['name']        = $field_data['name'];
        $fields[ $field_data['id'] ]['description'] = $field_data['description'];
        $fields[ $field_data['id'] ]['hidden']      = $field_data['hidden'];
      }
    }
  }

  return $fields;
}
// starts the dual opt-in process for the specified email address
function fbz_subscribe($api_key, $email, $listid, $tags)
```

```
{
  $url="https://app.feedblitz.com/f/?SimpleApiSubscribe&key=" . $api_key . "&email=" .
$email . "&listid=" . $listid;
  if($tags!="") {$url=$url . "&tags=" . $tags;}
  $page = fbz_get_web_page($url);
  $xml = simplexml_load_string($page, "SimpleXMLElement", LIBXML_NOCDATA);
  $json = json_encode($xml);
  $array = json_decode($json,TRUE);
  return $array['rsp']['@attributes']['stat']=="ok";  // true if all is well, false
otherwise
}

// example: Get a publisher's lists
$apikey = "{{apikey}}";                   // Replace {{apikey}} with the FeedBlitz
user's API key.
$lists = fbz_get_lists($apikey,$lists);  // gets all the active lists
var_dump($lists);                         // for debugging
?>
```

The FeedBlitz Simple API Reference

The “Simple API” is a simple URLs to call to accomplish the most common tasks without (much) programming capabilities, nor needing to delve into XML. There are currently two calls in this API:

1. Adding an email address to a list.
2. Unsubscribing a subscriber
3. Updating custom field data about a subscriber

Adding an email address to a list

IMPORTANT: FeedBlitz will send an activation email per the list’s settings to the address specified in the call. No CAPTCHA is supplied, so the burden of responsibility for eliminating spambots falls onto you, the developer. Use of <script> to generate your form is a good way to defeat many automated spam bots.

Any initial UI (user interface) presented prior to that is the responsibility of you, the developer, to build. This API adds email addresses only; it cannot add any of FeedBlitz’s supported social media subscriptions.

To use the simple API, you simply use this URL (an HTTPS GET, for the technically minded), as follows:

```
https://www.feedblitz.com/f?SimpleApiSubscribe&key=<api_key>&email=<email>&listid=<lis
tid>
```

The following parameters are required:

<api_key>	The publisher’s API key, URL encoded
<email>	The email you want to add to your list, URL encoded.
<listid>	The ID of the list to add the subscriber to.

The list id is displayed on each list’s dashboard, under the list’s title and description.

Additionally, there are optional parameters you can supply to the API, to tag a subscriber and / or add custom fields to their record.

- To tag a subscriber, e.g. to add product purchase information, you supply a tags parameter, of the form tags=<taglist> where <taglist> is a comma separated, URL encoded list of tags to apply to the subscriber (Note: Tags are added as custom fields, with the value “1”).
- To provide custom field data, e.g. names, zip codes, etc., you may optionally provide other parameters in the URL as name / value pairs. The name will be treated as a custom field name, and the value the value to assign to that field for this subscriber. If the custom field doesn’t exist, it will be created.

Examples

Here are some examples, assuming the API key for the FeedBlitz account owning list 84 is "Abc123", to help show how to use this API:

1. `https://www.feedblitz.com/f?SimpleApiSubscribe&key=Abc123email=phil%40example.com&listid=84`

Starts the dual opt-in for phil@example.com to list 84.

2. `https://www.feedblitz.com/f?SimpleApiSubscribe&key=Abc123email=phil%40example.com&listid=84&tags=Widgets`

Starts the dual opt-in for phil@example.com to list 84, tagging the user with the "Widgets" tag.

3. `https://www.feedblitz.com/f?SimpleApiSubscribe&key=Abc123email=phil%40example.com&listid=84&tags=Widgets,Boxes,Stuff`

Starts the dual opt-in for phil@example.com to list 84, tagging the user with three tags: Widgets, Boxes and Stuff.

4. `https://www.feedblitz.com/f?SimpleApiSubscribe&key=Abc123email=phil%40example.com&listid=84&Name=Phil%20Hollows`

Starts the dual opt-in for phil@example.com to list 84, and assigns the value "Phil Hollows" to the custom field called "Name"

5. `https://www.feedblitz.com/f?SimpleApiSubscribe&key=Abc123email=phil%40example.com&listid=84&tags=CampaignX&FirstName=Phil&LastName=Hollows`

Starts the dual opt-in for phil@example.com to list 84, and assigns the value "Phil" to the custom field called "FirstName", "Hollows" to the "LastName" field, and tags the subscriber with the "CampaignX" tag.

What Happens

If the API call is successful, the API will return list-specific XML and start the dual opt-in process; it is up to the developer using the API to generate the UI update appropriate to their platform to tell the visitor to check their inbox. If unsuccessful, the reason for the failure will be returned in error XML of the form:

```
<rsp stat="fail">
<err code="-1" msg="Specified list not owned by this client account" />
</rsp>
```

The reason for the failure will be in the msg attribute of the <err> element.

Unsubscribing a Subscriber

Removes a subscriber from the specified list, or from all lists they're active on in the account.

```
https://www.feedblitz.com/f?SimpleApiUnregister&key=<api_key>&email=<email>&listid=<li
stid>
```

The <listid> parameter is optional. If it is omitted, the subscriber will be removed from all lists within the account; otherwise from the specified list only. Unsubscribing is silent, in that the subscriber is not informed of the status change.

Setting a Custom Field Value

This API lets you add or update a custom field in the account for any given subscriber. Note that custom field names are global to the account, not to any one list or subscriber, and a global field's value is global to a subscriber within any account.

If the field doesn't exist in the account, it is created. If the field value already exists, it is updated with the value you pass in. If the email address specified isn't in a list managed by the account, nothing happens.

For example, if you create a field name "foo" then the field "foo" will be available to be set for all of the subscribers in the account. If you set the value of "foo" to be "bar" for "phil@example.com" then "bar" will be the value of "foo" associated with that address throughout that account.

You can use this API to tag a subscriber as they interact with your product or service, independently of their subscription activities.

Endpoint:

```
https://www.feedblitz.com/f?SimpleApiSetFields&key=<api_key>&email=<email>
```

You can add fields by adding one or more additional parameters, either in the URL (if you GET it), or as form parameters if you POST it. Similarly, you can add tags via a comma separated list; tags are treated as custom fields with a value of 1.

Examples:

Here are some examples, assuming the API key for the FeedBlitz account is "Abc123", to help show how to use this API:

1. `https://www.feedblitz.com/f?SimpleApiSetFields&key=Abc123email=phil%40example.com&tags=Widgets`

Adds / updates the tag "Widgets" to phil@example.com

```
2. https://www.feedblitz.com/f?SimpleApiSetFields&key=Abc123email=phil%40e
   xample.com&tags=Widgets,Boxes,Stuff
```

Adds / updates three tags to phil@example.com: Widgets, Boxes and Stuff.

```
3. https://www.feedblitz.com/f?SimpleApiSetFields&key=Abc123email=phil%40e
   xample.com&Name=Phil%20Hollows
```

Assigns the value “Phil Hollows” to the custom field called “Name”

```
4. https://www.feedblitz.com/f?SimpleApiSetFields&key=Abc123email=phil%40e
   xample.com&tags=CampaignX&FirstName=Phil&LastName=Hollows
```

Adds a tag to phil@example.com, and also creates / updates FirstName and LastName

The FeedBlitz REST API

The FeedBlitz API enables users to access FeedBlitz features programmatically, allowing innovative new applications based on and including FeedBlitz data and functionality. The API enables comprehensive integration at the data and application layers using a consistent and easy to understand interface.

Applications of the API vary in their potential complexity, and might include:

- Most recent subscriber displays on a web site.
- Integrating, embedding and hiding FeedBlitz signup with existing forms and web sites.
- Automatically creating FeedBlitz subscription *and* syndication accounts.
- Enabling email syndications and subscriptions to your content programmatically.
- Integrating FeedBlitz in Web 2.0 “mashups”.
- Creating RSS and OPML services based on your FeedBlitz subscriptions or syndications.
- Providing a branded or innovative management interfaces, e.g. for mobile users, IM clients, corporate web sites and intranets.

Some of these applications are expanded on in the sections below.

The API itself is a REST (representational state transfer) API, using XML and HTTP. The caller sends HTTP GET and POST requests as documented below to the API server, and they are returned an XML document containing the appropriate response.

API Design and Usage Considerations

The API is designed to minimize the number of queries the caller needs to make to generate a useful dataset. As such, it is a relatively rich interface giving the user access to many core FeedBlitz resources. In particular, some complexity has been added to “join” resources at runtime to return comprehensive information in a single interaction, thus avoiding the need to repetitively call the API for one resource and then for a series of related objects. It is designed, then, to be efficient in terms of minimizing the number of client – server round trip interactions.

As such, some API calls may be computationally and bandwidth extensive. *API users are strongly encouraged to ensure they call the API with the narrowest scope possible* required to fulfill their needs, and not to call it excessively. For example, the feeds a user is publishing will typically not change very often; there is therefore no need to call the API every minute to query the user's syndications. An hourly or daily check is usually more than enough, using a cached copy for local serving for the user's purposes.

Privacy and Security

The API may not be used to overcome FeedBlitz's privacy and security policies. API users are bound by the same terms of service and privacy policies as the core FeedBlitz.com site (see http://www.feedblitz.com/tos.asp). For example:

- You may not use the API to determine the email of an anonymous subscriber.
- You may not use the API to discern information about resources unrelated to your account.
- You may not use the API to subscribe a user against their will.
- You may not use the API alter or change a different user's email, password or resources they control.

Some features are not available via the API by design for security or anti-abuse reasons. This includes subscriber import, which must be performed via the feedblitz.com user interface.

If FeedBlitz determines what appear to be attempts to abuse this API, such activity will result in the immediate termination of your FeedBlitz account, without refund (if applicable).

All API calls require authentication to the API server on each and every call. An authentication failure will return an error code to the caller. The results returned, like in the feedblitz.com GUI, are restricted in scope to the entities that the authenticated user has access to. Attempts to access entities outside of the caller's scope will return a failure code and / or empty resource XML.

The API is open to all users and publishers. In its initial release it will not be limited in terms of the number of calls allowed.

Accessing the API

API keys may be requested via FeedBlitz.com at **My Account | API Keys**.

The API is called by accessing a resource path (see the documentation below) as follows:

```
https://www.feedblitz.com/f.api/resourcepath?key=<your_api_key>
```

Use your API key in the above path (without the < > symbols).

Access Notes:

- 128-bit SSL is required; plaintext HTTP requests will be redirected.

The HTTP verbs allowed are:

GET	Return information about the resource
POST	Search, edit or update the resource
PUT	Create a new resource
DELETE	Delete the resource

No other verbs are supported. The extent to which each resource supports one or more of these verbs is documented in each resource's reference section below. Each call to the API is *stateless*; it is up to the caller to maintain any state variables necessary for their application. Authentication information is required on each and every call to the API.

Notes

- Any path entries beyond a valid path will simply be ignored.
- Anonymous users will not show their email addresses or IDs to the caller *unless* the API's caller is that user themselves, in which case the fields will be correctly populated.

Important:

- As FeedBlitz features are added other tags and attributes may appear in the returned XML. You should expect unknown or undocumented tags and handle them appropriately.
- A PUT is *not* the same as a POST if applied to an existing resource. A PUT to an existing resource will fail.
- Any changes made by a POST or DELETE are final and immediately committed. It may not be possible for you to undo and of these changes, even if you have a cached coy of the data you wish to restore. ***The API assumes that the caller has made all the appropriate application checks prior to the API being called when modifying data.***

FeedBlitz, LLC is not liable for any data loss or damages you may incur by using this API. Use of the API implies no warranty or guarantees about reliability, performance or applicability, and is subject to the FeedBlitz standard Terms of Service at www.feedblitz.com/tos.asp

Example usage:

```
https://www.feedblitz.com/f.api/subscriptions
```

This call returns the XML containing all the subscriptions (if any) for the currently authenticated user.

Basic FeedBlitz API XML

XML sent or received is sent using Unicode UTF-8 encoding.
All XML is wrapped within <feedblitz> API tags as follows

```
<feedblitzapi version="1.0" >
.
. resource specific XML
.
</feedblitzapi>
```

The version number is required and MUST be 1.0.

API resources usually follow the following scheme:

```
/resource class/resource identifier/resource class/identifier
```

So this is a valid resource:

```
/syndications/57/subscribers/124
```

It represents subscriber number 124's subscription to syndication 57. Using different API methods, it is possible to access, alter and delete this resource (provided you have the privileges to do so), as well as add new resources.

API Methods

GET – Read Data

A GET for the above resource will return the XML for subscriber 124's subscription to syndication 57. This call will only be successful if the authenticated user is publishing syndication number 57, and if subscriber 124 is in fact subscribed to it. If the call is unsuccessful the returned XML will contain an empty <subscription /> tag (if the user does not own syndication 57) or a populated <subscription> tag with an empty <subscriber /> tag if subscriber 124 does not subscribe to feed 57.

It is possible to use shorter but complete paths as follows:

Path	Comment
/syndications	Returns all the user's published feeds
/syndications/57	Returns the user's published feed # 57
/syndications/57/subscribers	Returns all the subscribers to feed 57
/syndications/57/subscribers/124	Subscriber 124's subscription to feed 57

Note that /syndications/subscribers is an *invalid* resource. To fetch all their subscribers, a user should simply GET the /subscribers resource.

Also note that you should only use the paths documented here. Some resource paths you may expect to work are in fact invalid and not supported, such as /subscribers/124/subscriptions/57 It is in fact just the same as /syndications/57/subscribers/124 which can be used instead. /subscribers/124 *is* valid, returning all 124's subscriptions that this called syndicates.

POST – Edit and Search

If supported, you can POST edits at any point in the path structure. The results will depend on the path used and the data in the POST. Use the returned XML to judge the success or otherwise of your edit; do not rely on the HTTP return code (which will always be 200 OK in this release).

In this example, a POST at each point in the hierarchy behaves as follows:

Path	Comment
/syndications	Updates all the syndication entities included in the POST data
/syndications/57	Only updates feed #57. Fails if any other feed id is specified.
/syndications/57/subscribers	Edits the subscribers included in the POST data for feed 57
/syndications/57/subscribers/124	Updates subscriber 124's subscription to 57, if the IDs match.

The results of a POST are typically (but not always – see the documentation) the same as a GET for the specified resource.

POST is also used for searching and sorting result sets. Where supported, replace the ID with the literal 'search":

Path	Comment
/syndications/search	Search and sort this user's syndications
/syndications/57/subscribers/search	Search the subscribes to feed 57

Not all resources support searching. See the documentation below for resource-specific information.

Important: You must POST with the MIME type text/xml or application/xml for your data to be interpreted correctly. You cannot post with using an ordinary HTML form in a browser.

DELETE – Resource Removal

DELETEs may be issued to remove the associated resources from the system. Resources are typically not physically removed from FeedBlitz; rather they are marked as deleted. Deleted resources are retrieved, if present, by a GET statement, so be sure not to present them to end users if that would be confusing or inappropriate. The scope of a DELETE is the same as for a GET; it works at the level specified by the resource path. Use DELETE with care, and ensure that the user has confirmed their desire to make the relevant changes before you call the API. Use the returned XML, not the HTTP return code, to evaluate the success of the requested operation.

Path	Comment

/syndications	Deletes all published feeds, and all the subscriptions to them.
/syndications/57	Deletes feed # 57 and removes any subscriptions.
/syndications/57/subscribers	Deletes all the subscribers to feed 57, but leaves the syndication itself alone.
/syndications/57/subscribers/124	Deletes subscriber 124's subscription to feed 57 only.

PUT – Add a New Resource

New resources are added by using a PUT at the relevant level in the path:

Path	Comment
/syndications	Add a new syndication
/syndications/57/subscribers	Add a new subscriber to feed 57

A PUT, like a POST, typically returns the full resource XML (as if a GET were performed on the newly created entity). Exceptions are made for privacy reasons in the /user resource, where only the email address of the user (if the user is non anonymous) is returned to the caller along with a status code indicating whether the operation was successful or not.

Important:

- A PUT on an existing resource is NOT treated by the FeedBlitz API as being the same as a POST to that resource. A PUT on an existing resource will not edit that resource's settings by design. You MUST use the POST method to alter an existing resource's variables.
- There are currently no DTDs or other formal online XML documentation for validating XML documents used by the API.
- Not all methods are supported by all resources. Don't assume.

Important: You must PUT with the MIME type text/xml or application/xml for your data to be interpreted correctly. You cannot post with using an ordinary HTML form in a browser.

Sample API Interactions

Using FeedBlitz Functionality: Adding Email Services to Your Online Application

Say you want to embed FeedBlitz capabilities into an existing form on your web site. You therefore need to present the FeedBlitz image verification test as part of your form, as the FeedBlitz API does not allow subscriptions or accounts to be created without passing the image verification test. This is to prevent automated submissions from spambots, and is the approach used on the feedblitz.com web site. Here's how you would do it (all these examples assume that you authenticate properly to the API server):

1) GET the /captcha resource to generate the challenge key and the path to the verification image.
2) Insert these values into your form and create a field in your form to capture the user's response.
3) POST the response and other captcha / subscription / registration data back to the appropriate API resource.
4) Check the return document for success, warning or failure codes.

If the image verification was passed and the API validated that the feed is available for subscriptions, then the user will have been added to the subscription and an appropriate activation email sent to then automatically by FeedBlitz. If not, the returned document will contain another captcha challenge for you to present to the user.

This basic technique, depending on the resource you PUT to and the data delivered, enables you to:

- Automatically start the subscription process for your site's registrants, or
- Automatically set up FeedBlitz email services for users of your application or service by creating a publication (syndication) account.

Adding Custom Validation to Email Signup

Say your site is required to validate the user's age, but the out of the box validation question FeedBlitz provides is not enough. (e.g. to satisfy the US COPPA regulations you might want to verify a user's age by having a form where they enter their birth date, rather than simply asking them to confirm they are 13 or over).

In this case you would present your own signup form, along with the FeedBlitz verification image as pulled using the API. Once the user has passed your validation code from your custom form, you pass the email address and the response to the image verification challenge to the API to sign them up for your content.

Testing the API

The best way to get started is to access your user information. Simply type in the URL for your ID into your browser to GET the XML document:

`https://www.feedblitz.com/f.api/user`

Enter your FeedBlitz user name and password in your browser's security dialog, and the XML for your information will appear. How that information is presented to you depends on your browser; the FeedBlitz API does not provide an XML stylesheet as part of the response.

FeedBlitz API RESOURCE REFERENCE GUIDE

Subscribers (Subscriber Management)

Core resource path: `/subscribers`

`/subscribers` returns all the subscribers to all the user's feeds. Each subscriber's XML record includes basic information about the subscriber and the user's feeds to which they are subscribing. Here's an example reply, showing that there is one (<count>) subscriber in this set of resources. This subscriber has two subscriptions syndicated by this API user, one to feed 90291 and one to feed 96755. The subscriber may have subscriptions to other syndications; they are not shown to this caller.

```
<?xml version="1.0" encoding="utf-8" ?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
        <rsp stat="ok">
              <success code="0" msg="Authorized" />
        </rsp>
        <subscribers>
             <count>1</count>
             <subscriber>
                   <id>21</id>
                   <email>phil@hollows.com</email>
                   <status>ok</status>
                   <count>2</count>
                   <subscription>
                         <id>90291</id>
                         <name>CNN.com</name>
                         <description>CNN.com delivers up-to-the-minute news and
                        information on the latest top stories, weather,
                        entertainment, politics and more.</description>
                         <link>http://rss.cnn.com/rss/cnn_topstories.rss</link>
                         <created>2006-08-21 11:37:38 -0500</created>
                         <updated>2006-08-22 09:56:51 -0500</updated>
                         <status>unsubscribed</status>
                         <liststatus>ok</liststatus>
                         <referer />
                         <ip />
                   </subscription>
                   <subscription>
                         <id>96755</id>
                         <name>CNNMoney.com</name>
                         <description>From CNN and Money magazine, CNNMoney.com
                        combines business news and in-depth market analysis with
                        practical advice and answers to personal finance
                        questions.</description>
                         <link>http://rss.cnn.com/rss/money_topstories.rss</link>
                         <created>2006-08-21 11:37:38 -0500</created>
                         <updated>2006-09-06 16:52:28 -0500</updated>
                         <status>ok</status>
                         <liststatus>ok</liststatus>
                         <referer>http://foo.com/f/f.fbz?NewList</referer>
                         <ip>65.96.75.0</ip>
                   </subscription>
            </subscriber>
        <subscribers>
</feedblitzapi>
```

Note that each resource class contains a <count> of the number of resources it contains; this is to simplify API uses that present summary information. In the above sample there are two <count> tags. The first indicates that the <subscribers> block contains 1 <subscriber>, and the second that the <subscriber> block contains two <subscription>s. Note that there is a <subscribers> (plural) block containing one or more <subscriber> (singular) entries.

Tag	Comment
<id>	The subscriber’s unique identifier in FeedBlitz. If the subscriber has elected to be anonymous, this will be an empty <id /> tag.
<email>	The subscriber’s current email address. If the subscriber has elected to be anonymous, this value will be “anonymous” and not a valid email address.
<status>	The subscriber’s current system status.

<subscription> resources are fully described in the subscription section below.

If a subscriber is listed as anonymous their email address will appear as “anonymous”

GET – Fetching Subscriber Information

Valid resource paths that return subscriber XML are:

Path	Comment
/subscribers	Returns all the user’s subscribers and subscriptions
/subscribers/124	All subscriber # 124 subscriptions
/syndications/57/subscribers	Returns all the subscribers to feed 57
/syndications/57/subscribers/124	Subscriber 124’s subscription to feed 57

If you request a resource that does not exist (say subscriber 124 does not actually subscribe to your syndication 57) you will be returned an empty <subscribers /> tag.

You can add the following URL parameters when you use GET:

Limitstart, limitcount, summary and since.

POST - Searching and sorting subscribers

You may POST to a subscriber URL replacing the subscriber ID with the word “search” to scan the user’s subscriber database, as follows:

Path	Comment
/subscribers/search	Search all subscribers
/syndications/57/subscribers/search	Search subscribers to feed 57

XML is posted to the resource to indicate the search and sort parameters, as follows:

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
       <searchpattern>foo</searchpattern>
       <searchstatus>undeliverable</searchstatus>
       <sort order=”desc”>email</sort>
       <limitstart>250</limitstart>
       <limitcount>500</limitcount>
</feedblitzapi>
```

Tag	Comment

<searchpattern>	Performs a literal (i.e. not keyword) search for the specified text in the subscriber's email address.
<searchstatus>	Returns subscribers with that given status.
<sort>	Sorts by the specified field. Specify "desc" as the order attribute to sort backwards; otherwise sorts will be returned in ascending order regardless. The default sort is by subscriber id. Valid sort values are: • id — sort by subscriber id • email — sort by email address • status — sort by subscriber's system status • created — sort by subscription create date • updated — sort by subscription updated date
<limitstart>	Only return results from this count forward. If omitted zero (the beginning) is assumed. The first row has a row number of zero.
<limitcount>	Restrict the number of returned objects this number. So the example above will return rows 250 to 749 inclusive, if they exist. Note that the results returned may not return <limitcount> individual subscribers; rather <limitcount> subscriptions are returned.

Note that when sorting by subscriber fields, all the appropriate subscription data is placed in a single subscriber record – there will be exactly one subscriber record in the returned result set.

If, however, you sort by a subscription date field (created / updated) then the subscribers are presented in subscription date order. There therefore may be multiple <subscriber> blocks for the same subscriber in the returned XML (depending on the search parameters). It is also possible, depending on the date values, for any one <subscriber> record to have multiple <subscription> records if that subscriber happens to have consecutive subscriptions in the date sorted result set.

If both <searchpattern> and <searchstatus> is specified both conditions must be satisfied (i.e. the search is <searchpattern> AND <searchstatus>). You may omit or specify empty tags for search parameters you do not care about, so the following is a valid search for all subscriber emails containing "noemail.org":

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
      <searchpattern>noemail.org</searchpattern>
</feedblitzapi>
```

The following search returns confirmed subscribers in reverse chronological order, and so could be used to provide "welcome to our newest subscriber" data on a web site.

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
      <searchstatus>ok</searchstatus>
      <sort order="desc">updated</sort>
</feedblitzapi>
```

POST – Editing Subscribers

You may POST to change a subscription's state, with the following restriction: you may not change a subscriber with a deleted, unsubscribed or ignored status. If you post a field other than status it will be ignored.

The following XML sets subscriber 17's state to "deleted"

```
<feedblitzapi version="1.0">
      <subscribers>
            <subscriber>
                  <id>17</id>
                  <status>deleted</deleted>
            </subscriber>
      </subscribers>
</feedblitzapi>
```

You can use this XML in two ways, removing a subscriber from one or multiple lists in a single operation:

- To remove subscriber 17 from ALL your syndications, post the above XML to the `/subscribers` resource path
- To remove subscriber 17 from syndication 57, post the above XML to `/syndications/57`

If you wish, you may omit the subscriber number from the XML and post it to the subscriber-specific path as follows (since the subscriber ID is determined by the path):

```
<feedblitzapi version="1.0">
      <subscribers>
            <subscriber>
                  <status>deleted</deleted>
            </subscriber>
      </subscribers>
</feedblitzapi>
```

- To remove subscriber 17 from ALL your subscriptions, post the above XML to the `/subscribers/17` resource path
- To remove subscriber 17 from syndication 57, post the above XML to `/syndications/57/subscribers/17`

If an ID is specified in the XML and in the PATH, the ID specified in the PATH will be used. No warning will be given in the response.

After a POST has been made to edit subscribers, you will be returned the appropriate subscriber XML for resource specified in the path.

DELETE – Deleting a subscriber's subscription

A DELETE is analogous to a POST setting the status to "deleted" (indicating that you the owner of the list removed the subscriber, as opposed to the subscriber unsubscribing themselves). You may DELETE in the following resources:

Path	**Comment**
/subscribers	Deletes all the subscribers to all your syndications

/subscribers/17	Deletes subscriber 17 from all your syndications
/syndications/57/subscribers	Deletes all the subscribers to syndication 57, but leaves the syndication in place.
/syndications/57/subscribers/17	Deletes subscriber 17 from syndication 57

After a delete, you will be returned the XML for the relevant resource.

PUT - Adding a new subscriber

You may not PUT to a subscriber resource. To create a new subscription you must PUT to the /user resource instead. See the documentation for the /user resource for important information about starting the subscription process for a syndication that you manage using the API.

Subscriptions (Managing the Updates you Receive)

Core resource path: `/subscriptions`

Returns data about a user's subscriptions (i.e. the feeds to which they themselves are subscribing), or an empty <subscriptions /> tag. Information is provided about the subscription itself, but no publisher information is provided in order to protect the privacy of the publisher. Sample XML is below.

```
<?xml version="1.0" encoding="utf-8" ?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
      <rsp stat="ok">
            <success code="0" msg="Authorized" />
      </rsp>
      <subscriptions>
            <count>1</count>
            <subscription>
                 <id>90291</id>
                 <name>CNN.com</name>
                 <description>CNN.com delivers up-to-the-minute news and
                information on the latest top stories, weather, entertainment,
                politics and more.</description>
                 <link>http://rss.cnn.com/rss/cnn_topstories.rss</link>
                 <created>2006-08-21 11:37:38 -0500</created>
                 <updated>2006-08-22 09:56:51 -0500</updated>
                 <status>unsubscribed</status>
                 <liststatus>ok</liststatus>
                 <referer>http://www.cnn.com/</referer>
                 <ip>1.2.3.4</ip>
            </subscription>
      </subscriptions>
</feedblitzapi>
```

Tag	Comment
<count>	Number of <subscription> objects in a <subscriptions> block
<id>	The subscription's unique FeedBlitz ID
<name>	The name of the feed as defined to FeedBlitz. The feed's polled XML may be different.
<description>	The description of the content as defined to FeedBlitz. The polled XML may differ.

<link>	The URL polled by FeedBlitz
<created>	Timestamp when the feed was added to FeedBlitz
<updated>	Timestamp of the last status change (e.g. from pending to ok)
<status>	The feed's current status (pending / ok / paused / deleted / finished / ignored) as set by the subscriber
<liststatus>	The subscription's current system status as of the last poll (e.g. ok / fail)
<referer>	The source of the subscription. This may be: • Blank – no referrer was found, or the subscription was added prior to FeedBlitz tracking referrer URLs • URL – the referring URL as reported by the user's client • Import: <text> - the subscriber was imported. The text describes the IP and the FeedBlitz user ID used to perform the import. • Autosyndication: <URL> The subscription was added automatically because the user is subscribed to a syndicated OPML (reading list / blogroll) file • Autosubscription: <URL> The subscription was added automatically because the user had subscribed themselves to an OPML (reading list / blogroll) file • Autoresponder: feedid The subscription is an autoresponder, associated with the specified feed. **Note**: We retain HTTP's mis-spelling of the word referrer here, using a single "r"
<ip>	The IP octet of the host creating (not updating) this subscription. May be blank if the subscription was created prior to FeedBlitz tracking subscription IPs.
<uid>	User ID required to access the URL. Only returned if you are the owner of the subscription.
<pwd>	Password required to authenticate to the remote URL. Again, only displayed if you are the owner of the subscription.
<includetags>	Comma separated list of tags to include in the email. These tags are applied only to posts that pass the publisher's tags (if any)
<excludetags>	Comma separated list of tags to exclude from the email. These tags are applied only to posts that pass the publisher's tags (if any)
<excludeifboth>	By default, if an include and exclude tag are present, the post is included. This is "1" if the post should be excluded. Applies to the subscriber's tags only, and only to posts filtered in by the publisher's tags.

It is easy to convert this output using a script or XSL into an RSS feed, HTML page or OPML blogroll for a user based on this output. For most users there should be no reason to check this content more than daily, unless the caller knows that a subscription change has just been made.

Important

- Premium subscription information is not currently provided.

GET – Fetching Subscription Information

Valid paths returning subscription XML are:

Path	Comment
/subscriptions	Returns all the user's subscriptions
/subscriptions/57	Returns the user's subscription to feed # 57

POST - Searching Subscriptions

Like some other resources, Subscriptions support searching via POST of an XML document to the resource URL:

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
      <searchpattern>foo</searchpattern>
      <searchstatus>fail</searchstatus>
      <searchlink>http://foo.com</searchlink>
      <sort order="desc">updated</sort>
</feedblitzapi>
```

Tag	Comment
<searchpattern>	Performs a literal (i.e. not keyword) search for the specified text in the subscription's name, description or URL.
<searchstatus>	Returns subscriptions with the given status. Valid feed status values are: • ok • fail • paused • deleted
<searchlink>	Searches for an exact match for the URL. Since URL's are unique to any one user within FeedBlitz, it makes no sense to specify <searchlink> and <searchpattern>.
<sort>	Sorts by the specified field. Specify "desc" as the order attribute to sort backwards; otherwise sorts will be returned in ascending order regardless. The default sort is by list id. Valid sort values are: • id — sort by subscriber id • status — sort by subscriber's system status • created — sort by subscription create date • updated — sort by subscription updated date • link — sort by subscription URL • description — sort by the feed's description

Specifying more than one search criteria (e.g. <searchpattern> and <searchstatus>) requires that all conditions to be satisfied for a subscription to be returned, i.e. the search is an AND operation.

POST – Editing Subscriptions

You may POST to change a subscription's state. The behavior and values you may change with a POST depend on the nature of the subscription you are editing.

- If you are subscribing to a different user's syndication, only the status will be changed.
- If you are subscribing to your own syndication, you may change the name, link, uid, pwd and status values. Your syndication will be updated with these values.
- If your subscription is not a FeedBlitz syndication, you may change the name, link, uid, pwd and status values.

The XML you post may have multiple <subscription> tags within the single <subscriptions> collection. In addition, you may operate on the following resources:

Path	Comment
/subscriptions	Changes all your subscriptions as specified in the XML
/subscriptions/123	Changes subscription # 123.

If an ID is specified in the XML and in the PATH, the ID specified in the PATH will be used. No warning will be given in the response.

After a POST has been made to edit subscribers, you will be returned the appropriate subscriber XML for resource specified in the path.

To unsubscribe from a feed, set its status to "unsubscribed."

DELETE – Deleting a subscription

A DELETE is analogous to a POST setting the status to "deleted" If you wish to stop receiving updates from a subscription (including one that you are syndicating), you may delete it or POST an "unsubscribed" status to the resource.

You may DELETE in the following resources:

Path	Comment
/subscriptions	Deletes all your subscriptions
/subscriptions/17	Deletes subscription 17

After a delete, you will be returned the XML for the relevant resource.

PUT - Adding a new subscription

Using PUT creates a new, independent subscription in FeedBlitz <u>for the caller alone</u>. *This method should only be used where there is no existing FeedBlitz syndication for the user to subscribe to*. This is the method used in FeedBlitz.com when a user sets up a new subscription from the FeedBlitz dashboard itself. This is NOT the method used when the user enters their email address in your signup dialog on your website (that's the PUT to the \user resource).

So, if you expect more than one user to subscribe to any given feed in FeedBlitz, or you want to syndicate this feed to others, do not use this method.

Instead, create a new Syndication resource (if necessary) and then PUT to the appropriate /user resource to subscribe the user to it. This latter approach greatly simplifies management tasks, since updating the subscribers is then simply a matter of changing the relevant /syndication resource, as opposed to having to modify multiple individual distinct subscriptions. Using /syndications also makes all the subscribers to a feed appear in a single location in the FeedBlitz GUI.

If you do need to create a standalone subscription, PUT to this resource. If you are calling on behalf of a third party client, you should specify the (client) IP and referrer fields if applicable; otherwise the system will attempt to derive them from the runtime environment.

You may PUT any field to the API. You may omit fields that do not apply, but <name> and <link> are required. Date fields will be ignored and the actual times inserted.

Syndications

Core resource path: `/syndications`

Returns data about the user's current published syndications, or an empty <syndications /> tag. If a syndication is found, comprehensive information is provided (including summary subscriber metrics) to avoid further API queries. A summary version is also available (see the "summarizing syndications" section below).

```
<?xml version="1.0" encoding="utf-8" ?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
      <rsp stat="ok">
            <success code="0" msg="Authorized" />
      </rsp>
      <syndications>
            <count>1</count>
            <syndication>
                  <id>33369</id>
                  <publisherid>21</publisherid>
                  <publisheremail>phil@hollows.com</publisheremail>
                  <name>FeedBlitz Widgets</name>
                  <description>FeedBlitz Widgets   a test feed</description>
                  <link>http://foo.com/foo/atom.xml</link>
                  <created>2006-02-24 23:03:11 -0500</created>
                  <status>ok</status>
                  <liststatus>ok</liststatus>
                  <turbo />
                  <uid />
                  <pwd />
                  <notifyowner>1</notifyowner>
                  <tease>Read the whole entry</tease>
                  <entrylength />
                  <keepprivate>1</keepprivate>
                  <type>Atom 1.0</type>
                  <linktrack>0</linktrack>
                  <opentrack>0</opentrack>
```

```
                        <isauto>0</isauto>
                        <autoresponderid>0</autoresponderid>
                        <validationquestion />
                        <validationfail />
                        <includetags />
                        <excludetags />
                        <excludeifboth />
                        <addcomments>0</addcomments>
                        <forwardemail>1</forwardemail>
                        <timezoneid>35</timezoneid>
                        <dynacontent>
                               <id>103</id>
                               <type>0</type>
                               <providerid>2</providerid>
                               <value>pub-xxx</value>
                        </dynacontent>
                        <dynacontent>
                               <id>140</id>
                               <type>1</type>
                               <providerid>4</providerid>
                               <value>UA-xxx-1</value>
                        </dynacontent>
                        <subscribersummary>
                               <subscribers status="deleted">2</subscribers>
                               <subscribers status="ignored">1</subscribers>
                               <subscribers status="ok">3</subscribers>
                        </subscribersummary>
                </syndication>
        </syndications>
</feedblitzapi>
```

Entries are:

Tag	Comment
<count>	Number of <syndication> objects in a <syndications> block
<id>	The syndication's unique FeedBlitz ID
<publisher>	The subscriber ID of the publisher (blank if anonymous)
<publisheremail>	The publisher's address (or "anonymous")
<name>	The name of the feed as defined to FeedBlitz. The feed's polled XML may be different.
<description>	The description of the content as defined to FeedBlitz. The polled XML may differ.
<link>	The URL polled by FeedBlitz
<created>	Timestamp when the feed was added to FeedBlitz
<status>	The feed's current status (ok / paused / deleted) as set by the owner
<liststatus>	The feed's current system status as of the last poll (e.g. ok / fail)
<turbo>	The syndication's turbo frequency, if any. -2 indicates a standard (free service), as does zero or empty. 1440 indicates an active time zone override. -3 indicates on-demand, -5 Express. Other positive values indicate the number of minutes between each poll cycle.
<uid>	User id if required by the poller for feeds needing authentication
<pwd>	Password if required to access an feed needing authentication
<notifyowner>	Whether the user wishes to receive (un)subscribe email notifications
<tease>	Call to action text if the user is truncating feed content
<entrylength>	Maximum # of characters of an entry. If 0, only headlines are sent.
<keepprivate>	Do not show in FeedAdvisor results or other public places

<type>	Type of feed at create time. Not updated by the system and not to be relied upon.
<linktrack>	1 if the user wishes to track click-throughs
<opentrack>	1 if the user wishes to track email opens
<isauto>	1 if this syndication may be used as an autoresponder
<autoresponderid>	The ID of the feed this feed uses as an autoresponder
<validationquestion>	User must confirm this text for a subscription to succeed
<validationfail>	Text to present if the user does not confirm the verification
<subscribersummary>	Block of tags summarizing subscriber info for this feed
<includetags>	Comma separated list of tags to include in the email
<excludetags>	Comma separated list of tags which should not be in the mail
<excludeifboth>	By default, if an include and exclude tag are present, the post is included. This is "1" if the post should be excluded.
<addcomments>	1 if the user has comments links added to their outbound emails.
<forwardemail>	1 if the user has "email to a friend" enabled in the outbound emails.
<timezoneid>	The time zone ID of the feed; used to determine the time and scope of a "nightly" run.
<dynacontent>	Automatically inserted content for ads and site metrics.
<subscribers status="">	Count of subscribers with this status.

The <subscribers> summary tag only shows the # of subscribers present. So if there are no pending registrations there will be no <subscribers status="pending"> entry (as is the case in the example record above). Note that the <subscribers> tag here is not the same as the <subscribers> resource documented elsewhere in this document. If the called requires complete subscriber information they can access the appropriate /subscribers resource for the relevant syndication(s).

It is easy to convert this output using a script or XSL into an RSS feed, HTML page or OPML blogroll for a user based on this output. For most users there should be no reason to check this content more than daily, unless the caller knows that a subscription change has just been made.

Important

- Premium feed information is not currently provided.

GET – Fetching Syndication Data

Valid paths returning syndication XML are:

Path	Comment
/syndications	Returns all the user's published feeds
/syndications/57	Returns the user's published feed # 57

See the subscribers section for additional resource URLs that are syndication-specific.

Summarizing Syndications

For some needs, such as presenting a GUI with just the name, description and URL of the syndication, the XML returned by this call is unnecessarily verbose. The caller may reduce the amount of information returned by specifying the "summary=1" in the resource URL that they GET or POST to. An example is:

```
https://www.feedblitz.com/f.api/syndications?summary=1
```

When the API sees a summary flag associated with a request for a syndication resource, the returned XML is reduced to the following:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <rsp stat="ok">
                <success code="0" msg="Authorized" />
         </rsp>
         <syndications>
                <count>1</count>
                <syndication>
                       <id>33369</id>
                       <name>FeedBlitz Widgets</name>
                       <description>FeedBlitz Widgets   a test feed</description>
                       <link>http://foo.typepad.com/foo/atom.xml</link>
                       <status>ok</status>
                       <liststatus>ok</liststatus>
                       <turbo>1440</turbo>
                       <subscribersummary>
                              <subscribers status="deleted">2</subscribers>
                              <subscribers status="ignored">1</subscribers>
                              <subscribers status="ok">3</subscribers>
                       </subscribersummary>
                </syndication>
         </syndications>
  </feedblitzapi>
```

This form is simpler and easy to convert to RSS, HTML and OPML. There is no analogous summary operation for subscription resources.

POST – Searching Syndications

Like some other resources, Syndications support searching via POST of an XML document to the resource URL:

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
       <searchpattern>foo</searchpattern>
       <searchstatus>fail</searchstatus>
       <searchlink>http://foo.com</searchlink>
       <sort order="desc">updated</sort>
</feedblitzapi>
```

Tag	Comment
<searchpattern>	Performs a literal (i.e. not keyword) search for the specified text in the syndication's name, description or URL. It is not possible to restrict the search to just the URL, say.
<searchstatus>	Returns syndications with that given status. Valid feed status values are: • ok • fail

	• paused • deleted
<searchlink>	Searches for an exact match for the URL. Since URL’s are unique to any one user within FeedBlitz, it makes no sense to specify <searchlink> and <searchpattern>.
<sort>	Sort the results, if omitted the default sort of list id will be used. You may sort on any tag in the syndication record that does not have child tags, e.g. you may sort on the “created” field but not the “subscribersummary” tag. You may sort on these fields even if you have requested summary results only, when the field itself is not returned in the response XML.

Specifying multiple search parameters (e.g. both <searchpattern> and <searchstatus>) requires all conditions to be satisfied for a syndication to be returned. You may post a search to a syndication URL with the summary flag enabled; this will return the summary form of the syndication XML resource.

POST – Editing Syndications

You may POST syndication information to the relevant resource URL. Dates are stripped out, as are any <subscribersummary> values. Note that changing the information in a syndication changes it for all subscribers, and may also change the appearance of the feed in FeedBlitz sites, including FeedAdvisor.com.

Setting a syndication’s status to “deleted” will stop circulation of email updates and remove the circulation from subscriber’s dashboards. Marking it as “paused” stops circulation, but leaves the feed in subscriber’s dashboards (in the paused state).

A POST will return the appropriate XML for the selected resource.

DELETE – Deleting Syndications

Deletes the resource, returning the appropriate XML on completion.

PUT – Creating a New Syndication

PUT the appropriate XML data to the /syndications resource. The feed will become immediately available for you to manage via the API or the FeedBlitz.com GUI. The caller will also be added automatically as a new subscriber.

IMPORTANT: No autodiscovery or validation is performed on the URL passed to the API. If you wish to perform autodiscovery you should perform that yourself in advance, determine the URL to be added as the link, and then POST it to the API.

If successful, you will be returned the full XML for the new resource. The core element in the returned XML is the ID. The ID is the same value as used in the HTML "FEEDID" input parameter for FeedBlitz signup dialogs and links. You may therefore use this information to dynamically construct a signup dialog yourself without having to use the feedblitz.com GUI.

Captcha (Image Verification)

Resource path: `/captcha`

Returns a path to a verification image and a key to go with it. Users type the key embedded in the image, which is then sent back to the API server for verification. The captcha image may be used standalone, and is also embedded with the creation of a new user or a new subscription resource.

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <rsp stat="ok">
                <success code="0" msg="Authorized" />
         </rsp>
         <captcha>
                <random>123</random>
                <img>http://feedblitz.com/an_image_file</img>
                <response>what_the_user_typed</response>
         </captcha>
  </feedblitzapi>
```

When used with a GET the <response> tag is not returned. To validate a user's response, post the <random> and the <response> tags to the resource (the <img> tag is not required). If it validates, the <rsp> tag from the POST will be OK. If not, a new <captcha> will be served back to you to be re-presented to the user.

You may use the /captcha resource standalone to validate a user is not a machine, as well as using it embedded in the /user resources as described below.

User (Account registration and Subscription Signup)

Resource path: `/user`

The /user resource has three distinct uses:

1) To retrieve information about the authenticated user
2) To register a new account at FeedBlitz
3) To start the new subscription signup process

GET – About the Current User

Returns information about the current logged on user, including their password. The API requires SSL, so the integrity of the password information is protected in transit.

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <rsp stat="ok">
                <success code="0" msg="Authorized" />
         </rsp>
         <user>
                <id>123</id>
                <email>phil@feedblitz.com</email>
                <password>sekret</password>
                <mailformat>HTML</mailformat>
                <keepprivate>0</keepprivate>
                <status>ok</status>
                <created>2005-08-01 22:23:00 -0500</created>
                <ip>1.2.3.4</ip>
                <failcount>0</failcount>
                <bounces />
         </user>
  </feedblitzapi>
```

Tag	Comment
<id>	The user's unique identifier within FeedBlitz
<email>	The user's email address
<password>	The user's FeedBlitz password
<keepprivate>	1 if the user is set to be reported to publishers as anonymous, 0 otherwise
<status>	ok / pending / undeliverable / deleteme / deleted / unsubscribed
<created>	Date the ID was created
<ip>	IP from which the user was created. May be an octet or "Imported by:" and text describing the source of the address.
<mailformat>	The literal: HTML or TEXT
<failcount>	Number of bounces received. Used for soft bounce processing (hard bounces send the user to the "deleteme" state, from which they are set to "deleted" nightly.
<bounces>	Number of consecutive delivery failures as recorded by the FeedBlitz servers. A successful delivery resets this value to zero (i.e. an empty tag).

There is exactly one user if the caller authenticates: the caller themselves.

POST – Updating the Current User

The current user may change any of their settings (except their ID) by posting the XML to the /user resource.

Important: If you set the current user's state to "deleted" they will no longer receive updates, but their syndications will remain intact (so that their subscribers will continue to receive updates). This is by design: should the publisher of an email become undeliverable and then removed by the system (asset to "deleted") then it is unreasonable to terminate their subscribers as well.

A user who successfully logs in to either the GUI or the API will have their status restored.

To remove a user AND their syndications AND their subscribers, you need to DELETE the user using the DELETE method (see below).

PUT – Registering a New User Account

To register a new user and start the authentication process via the email activation link, PUT the XML document to the /user resource (API callers must authenticate with a different ID). A valid captcha entry MUST be part of the XML document sent to the resource, as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <user>
                <email>phil@feedblitz.com</email>
                <password>sekret</password>
                <keepprivate>0</keepprivate>
                <mailformat>TEXT</mailformat>
                <ip>1.2.3.4</ip>
                <captcha>
                       <random>123</random>
                       <response>what_the_user_typed</response>
                </captcha>
         </user>
  </feedblitzapi>
```

If the image verification fails (i.e. the <response> is incorrect) you will be returned a fresh captcha that must be used in the next attempt.

If the registration fails (because the user already has an account) then a warning or error status will be returned indicating the nature of the problem.

Regardless, the returned document will only contain the email you attempted to register; it is NOT the same as a GET for that user. This preserves privacy and security settings that the extant user may have in place. Also note that a PUT to an existing user is NOT the same as a POST – you may only use PUT to create a new user. Callers must POST to update an account's settings.

If the PUT is successful the user will be sent an activation email. You may neither use the API nor the FeedBlitz GUI using the new account's credentials until the user has activated their account using the link in the confirmation email. If the link must be resent the user must use the lost password facility at feedblitz.com

PUT – A New Subscription to an Existing Syndication

This operation mimics the effect of the user adding their email to your FeedBlitz signup form. The process is similar to creating a new account, except in this case there is a subscription entity in the user XML as well as the captcha.

You do not need to know the user's FeedBlitz ID; the email address will be used. The subscription's ID is required however, as in the following XML:

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

```
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
        <user>
              <email>phil@feedblitz.com</email>
              <captcha>
                    <random>123</random>
                    <response>what_the_user_typed</response>
              </captcha>
              <subscription>
                    <id>1234</id>
                    <referer>http://foo.com/signup.html</referer>
                    <ip>1.2.3.4</ip>
              </subscription>
        </user>
  </feedblitzapi>
```

You may have multiple <subscription> blocks in the XML to sign a user up for multiple options (e.g. a daily and a weekly summary). Review the returned XML carefully for errors.

If the email address exists in FeedBlitz already, an email will be sent to the user inviting them to confirm their subscription. If not, an account will be created and then the activation email sent. Like the PUT above, the only returned user information will be the email address supplied by the caller, and a PUT is not the same as a POST – a PUT on an existing user will not update that user's information (such as their password).

DELETE – Deleting a User

The delete method not only marks the user as deleted, it also suspends all the user's syndications, subscriptions, and the subscriptions of any users subscribed to that user's syndications.

Autoresponders (aka Funnels)

Resource URL: `/autoresponders`

FeedBlitz offers an autoresponder capability. This resource returns all the autoresponders published by the user. If none are defined an empty <autoresponder /> tag is returned.

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
        <rsp stat="ok">
              <success code="0" msg="Authorized" />
        </rsp>
        <autoresponders>
              <count>1</count>
              <autoresponder>
                    <id>123</id>
                    <publisherid>57</publisherid>
                    <name>Thank You</name>
                    <description>Send a free daily email course for 7
                  days</description>
                    <link>http://mysite.com/courseware/email_free.rss</link>
              </autoresponder>
        </autoresponders>
  </feedblitzapi>
```

Tag	Comment

<id>	The ID of the feed. Use with the /subscriptions resource to find more information on the feed.
<publisherid>	The ID of the publisher
<name>	The name of the autoresponder as defined by the user to FeedBlitz
<description>	The feed's description, as entered by the user to FeedBlitz.
<link>	The source URL that is polled to feed the autoresponder.

This resource only supports the GET method. To change an autoresponder you need to POST to the appropriate /syndication resource.

For information on setting up and using autoresponders in FeedBlitz, see http://feedblitz.blogspot.com/2006/05/feedblitz-announces-rss-and-blog.html

Time Zones

Resource path: `/timezones`

In FeedBlitz 2 it I possible to specify the time zone of a syndication. In the standard service, FeedBlitz polls each syndication daily, starting shortly after 1am local time (before version 2, polling started shortly after 1am US eastern time). FeedBlitz adjusts its poll time for daylight saving changes in both the selected time zone and the server's own time zone.

The /timezone resource gives information about all the potential time zones available to FeedBlitz. A syndication's time zone ID is a reference to the relevant entity from this call.

Tag	**Comment**
<id>	The time zone ID
<name>	The common name for the time zone
<display>	The display name for the zone. Includes the offset from GMT as text, and one or more city or country names as appropriate.
<offset>	The *current* offset from GMT in text, e.g. -05:00
<offsetminutes>	The *current* offset from GMT in minutes

This resource only supports the GET method.

Dynamic Ad and Metric Insertion

Deprecated.

Mailing Resources

These resources enable the API user to send emails to a list (or a segment of a list). The mailing API has two commands, accessible via either PUT or POST, which map directly to the analogous commands in the FeedBlitz user interface:

Newsflash	Send any arbitrary message to the list
OnDemand	Send selected elements from the list's source feed to the list.

Unlike the user interface, these resources have a `<testmode>` element, enabling API developers to test their integrations without the risk of sending an email to an active list.

Newsflash

Resource path: `/newsflash/<listid>`

Methods: PUT, POST

Either PUT or POST works; they are synonyms. There is no difference in the XML sent / returned, nor in their effects.

To send a newsflash, POST or PUT the required XML to the list's newsflash resource. So, for example, to send a newsflash to list ID 84:

Resource URL: `/newsflash/84`

Posted XML:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <newsflash>
                <subject>Hello World</subject>
                <message>This &amp; that &amp; the other</message>
                <tags>testing,123,Is This Thing On</tags>
         </newsflash>
  </feedblitzapi>
```

Tags will be converted to a category header in the SMTP envelope for rendering by email clients that support categories.

Tag	Comment
<subject>	Required. Entity encoded subject line. The actual subject sent in the email will be derived by converting this text using the list's single subject envelope setting.
<message>	Required. Entity encoded HTML message. Will be processed by default using the list's active template unless the <usetemplate> entity is zero (see below).
<tags>	Optional: Comma separated list of entity encoded categories to send along with the mailing.

<seg_criteria>	Optional: Send to a segment of the list. Contains entity encoded segment expression per FeedBlitz's standard custom field expression rules. Default is empty, i.e. send to all subscribers.
<testmode>	Optional: When inner text is 1 (i.e. <testmode>1</testmode>) puts mailing in "sandbox" – nothing is sent. Otherwise mailings will proceed (i.e. implied default is 0)
<usetemplate>	Optional: Use to disable processing of the message by the list's template. (i.e. <usetemplate>0</usetemplate>). Default is 1, i.e. the template will be used. Set to zero when you want the message body to be sent as-is (it will always be treated as HTML by FeedBlitz, conversion to a corresponding text part is automatic).
<textpart>	Optional: Specify the text alternative to the main HTML message to be sent as a text MIME part in the mailing. If none is specified, FeedBlitz will automatically generate a text part from the HTML message.
<suppressionlistid>	Optional: Comma separated list of suppression lists owned by the account. Emails in the suppression lists will not be mailed the content. The list may be followed by a + or a – sign in which case it is treated as a ListID. + indicates suppress active subscribers to the specified list, - means suppress unsubscribed.
<fromname>	Optional: Override the list's settings with this friendly name
<fromaddress>	Optional: Override the list's settings with this sending address. This is also the reply-to address.
<conditions>	A set of autoresponder <condition> elements wrapped by <conditions>…</conditions>. See the Autoresponder API for insight into <condition> XML. Set the <anycondition> element at the <conditions> level (i.e. <conditions> <anycondition>1</anycondition> <condition>…</condition> </conditions>) to make the <condition>s evaluate as logical OR instead of logical AND.

Subject, Message, tag and seg_criteria elements must be entity encoded. The API does not check for validity in the HTML send in the body; it will simply format and send the content you define.

IMPORTANT: Once a mailing starts it cannot be stopped.

With test mode enabled no activity will occur but the API will report a success in the response along with a warning that test mode is on.

On Demand

Resource path: `/ondemand/<listid>`

Methods: PUT, POST

Either PUT or POST works; they are synonyms. There is no difference in the XML sent / returned, nor in their effects.

To send an on demand mailing, POST or PUT the required XML to the list’s ondemand resource. So, for example, to send an ondemand mailing to all subscribers on list ID 84 consisting of two posts:

Resource URL: `/ondemand/84`

Posted XML:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
         <ondemand>
                <link>Guid for first post</link>
                <link>Guid for second post</link>
         </ondemand>
  </feedblitzapi>
```

Tag	Comment
<link>	Required. Entity encoded link or GUID (guid preferred). Used to identify the articles to send. May be included multiple times (as in the above example) to specify multiple posts. The physical order of the <link> elements in the submitted XML defines the physical order in the eventual mailing; the first link will be the top article in the mailing, the next link will be the second etc. There are important uniqueness constraints, see the remarks below.
<seg_criteria>	Optional: Send to a segment of the list. Contains entity encoded segment expression per FeedBlitz’s standard custom field expression rules. Default is empty, i.e. send to all subscribers.
<testmode>	Optional: When inner text is 1 (i.e. <testmode>1</testmode>) puts mailing in “sandbox” – nothing is sent. Otherwise mailings will proceed (i.e. implied default is 0)

The seg_criteria and link elements must be entity encoded.

Notes:

- The underlying RSS feed must have unique GUIDs (<id> elements in Atom) or links, to enable FeedBlitz to correctly disambiguate similar posts from the feed. If this is not possible the mailing attempt will fail with an appropriate error returned to the caller.
- The link elements must be in the current feed visible to FeedBlitz. If not, FeedBlitz will not be able to find the post content and the mailing will fail. If this happens unexpectedly and you use server-side caching ensure that FeedBlitz can defeat the cache (or that caches have been properly flushed) prior to sending an on demand mailing to reduce the risk of a <link> not being found. It may also happen in the event of server side failure or performance problems. FeedBlitz times out fetches at 20s.
- You cannot specify the same link (feed post) multiple times.
- The <link> contents will be checked against the feed item’s link *and* GUID elements, to enable on demand mailings for feeds using formats that don’t supply GUID or ID elements. You may therefore specify the post’s URL in the <link> as it appears in the source feed. It is recommended, however, that you provide the GUID (or <id> element if your feed is Atom) as these elements are required to be unique according to the specs.

IMPORTANT: Once a mailing starts it cannot be stopped.

With test mode enabled no activity will occur but the API will report a success in the response along with a warning that test mode is on.

As of 8/7/2015, a successful API call to OnDemand will return the associated mailing ID in the response XML. This mailing ID can then be used to query the mailing's metrics.

SendMail – Send a one-off email to a single subscriber

This resource sends a one-off email to an existing, opted-in subscriber to a list owned by the account. The list ID is required for tracking, metrics, bounce handling and providing the correct unsubscribe option. It is significantly more efficient (read: expedient) to use this API to email a single reader than to send a Newsflash or OnDemand mailing to an entire list with a segment of email="subscriber@example.com" segment.

Resource path: /sendmail/<listid>

Methods: POST

Sample Posted XML:

```
<feedblitzapi version="1.0" >
    <sendmail>
        <subject>This &amp; that - an API initiated mailing</subject>
        <message>
            &lt;h1&gt;Headline&lt;/h1&gt;&lt;p&gt;Something or other&lt;/p&gt;
        </message>
        <fromname>Philbert</fromname>
        <fromaddress>phil@hollows.com</fromaddress>
        <testmode>0</testmode>
        <recipients>phollows@gmail.com</recipients>
    </sendmail>
</feedblitzapi>
```

Tag	Comment
<subject>	Required. Entity encoded subject. May contain HTML tags that will be removed from the email subject line, but will be included in the body of the email if the list's active template uses the <$BlogDescription$> tag
<message>	Required: The HTML markup, entity encoded to be valid XML.
<recipients>	Required: Comma separated emails. Any emails not active (i.e. unsubscribed, deleted from the system, not yet activated, or simply not found) on the list will be suppressed
<fromname>	Optional: Override the list's settings with this friendly name
<fromaddress>	Optional: Override the list's settings with this sending address. This is also the reply-to address.
<testmode>	Optional: If 1, the email won't be sent. Useful for testing API interactions without aggravating real subscribers. Default value if not specified is 0 i.e. email will be sent.

<tags>	Comma separated list of entity encoded categories or tags. Will be added to the SMTP headers and email apps which display categories (e.g. MS Outlook) will show them
<usetemplate>	Optional: Disable the list's template for this mailing, if sent to 1. In which case the HTML specified in the <message> element will be sent as is. Otherwise, the message will be treated as a virtual blog post / RSS entry and formatted with the list's current template. Default value is 1.
<textpart>	Optional: Specify the text alternative to the main HTML message to be sent as a text MIME part in the mailing. If none is specified, FeedBlitz will automatically generate a text part from the HTML message.

Email Design Templates

The template resource allows programmatic definition of the template used by mailings sent from FeedBlitz.

Resource path: `/template/<listid>`

Methods: GET, POST, DELETE, PUT

Example: `/template/84`

The best way to understand this resource is to define a template manually in the FeedBlitz user interface (it need not be activated), and then GET it with this method.

When saving a template using PUT or POST, the 1st three elements are required. You can save a template deactivated if you wish; if not specified initially, the template is inactive. If a feed's template is inactive, it uses the account's master template (if specified in the UI) or the FeedBlitz default template otherwise.

Tag	Comment
<fromname>	The friendly name of the sender. Required for PUT / POST
<fromaddress>	The email address in the "From" column. Also used for replies if the recipient responds to the mailing, whether manually or via out-of-office email autoresponder. Required for PUT / POST
<template>	HTML template to be used. Required for PUT / POST
<active>	0 – inactive 1 – active
<mastertemplate>	0 – template is NOT the master for the account 1 – template is the master for the account

Note: All text fields should be encoded to ensure XML validity.
Note: Inactivating a template prevents all elements of the template (mastertemplate, sender name, address and the template itself) from being used.
Note: For POST and PUT, the <mastertemplate> value is optional. If not specified, the account's master template ID is unchanged. If you set a list to be the mastertemplate, it will make sure that a prior mastertemplate record is removed (i.e. you don't have to explicitly disable mastertemplate on the prior list if you're switching the mastertemplate from a prior list to this one).

Report Resource

The report resource parallels the reports available in the FeedBlitz user interface. The default (used by a simple GET) is the delivery metrics report for the selected list covering the most recent 7 days.

Resource path: `/report/<listid>`

Methods: GET, POST

Example: `/report/84`

Posted XML:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
        <report>
              <type>11</type>
              <range>7</range>
        </report>
  </feedblitzapi>
```

Tag	Comment
<type>	The type of report requested. Valid types are: 0: Click through summary 2: Open analysis 5: Click throughs by reader 6: Opens by reader 7: Click throughs by tag 9: Unsubscribe summary 10: New subscription referrers 11: Delivery metrics 12: Engagement 13: Subscriber count history
<range>	Number of historic days' data to fetch relative to now. Default is 7, the most recent 7 days.

Note: Long time frames for large lists may result in time outs. If you encounter persistent performance problems please contact FeedBlitz tech support.

The data returned by the API is analogous to the rows of the corresponding tables in the FeedBlitz API. Each row's data set is surrounded by a set of <values> tags; the tags within each set vary by report.

Here is an example two row result set for the delivery metrics type:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
      <success code="0" msg="Authorized" />
</rsp>
<report>
      <values>
            <Cycle_ID>4030813</Cycle_ID>
            <Cycle>Four Hour [2011.06.14 12:00]</Cycle>
            <Sent>1</Sent>
            <Opens></Opens>
            <Clicks></Clicks>
            <Unsubs></Unsubs>
            <SoftBounces></SoftBounces>
            <HardBounces></HardBounces>
            <Complaints></Complaints>
            <Fwds></Fwds>
      </values>
      <values>
            <Cycle_ID>4030808</Cycle_ID>
            <Cycle>12 Hour [2011.06.14 12:00]</Cycle>
            <Sent>3</Sent>
            <Opens>3 (100.0%)</Opens>
            <Clicks>1 (33.3%)</Clicks>
            <Unsubs></Unsubs>
            <SoftBounces></SoftBounces>
            <HardBounces></HardBounces>
            <Complaints></Complaints>
            <Fwds></Fwds>
      </values>
</report>
</feedblitzapi>
```

Where present, the <cycle_id> inner content directly maps to a single send and its corresponding archive URL, of the form:

http://archive.feedblitz.com/<listid>/~<cycle_id>

e.g. http://archive.feedblitz.com/84/~4031278

Mailing Metrics Resource

This report returns the summary metrics for an individual mailing / cycle ID. It is analogous to the mailing summary report in the UI.

Resource path: `/metrics/<listid>`

Methods: POST

Example: `/metrics/84`

Posted XML:

```
<?xml version="1.0" encoding="utf-8" ?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
        <metrics>
              <mailingid>4976718</mailingid>
        </metrics>
  </feedblitzapi>
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
      <success code="0" msg="Authorized" />
</rsp>
<metrics>
      <sent>26243</sent>
      <opens>2479</opens>
      <uniqueopens>1933</uniqueopens>
      <clicks>724</clicks>
      <uniqueclicks>101</uniqueclicks>
      <unsubscribes>33</unsubscribes>
      <softbounces>17</softbounces>
      <hardbounces>31</hardbounces>
      <complaints>3</complaints>
      <forwards>0</forwards>
</metrics>
```

```
</feedblitzapi>
```

Currently metrics are not available for A/B test detail if an A/B test has been run.

RSS Feed REST APIs

These resources access elements of FeedBlitz's RSS service.

Feeds Resource

Returns a list of active RSS feeds and summary data about the feed. If a feed ID is not specified, all the feeds for the account will be returned/

Resource path: `/feeds/<listid>`

Methods: GET

Example: `/feeds`

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
	<success code="0" msg="Authorized" />
</rsp>
<feeds>
	<feed>
		<id>1027><id>
		<title><![CDATA[FeedBlitz News]]></title>
		<uri>FeedBlitz</uri>
		<circulatiom>29234</circulation>
		<source>http://www.feedblitz.com./feed</source>
		<status>ok</status>
		<reason/>
	</feed>
</feeds>
```

- The "reason" element will provide insight into any problems if the status is not "ok"
- If the publisher has multiple RSS feeds, there will be multiple <feed>…</feed> elements.

`/feeds/<feedid>`

Provides summary data for the specified feedid.

`/feeds/<feedid>/stats/<fromdate>/<todate>`

Provides summary data (circulation, reach, views, clicks, downloads) for the specified feed.

The <feedid> resource is required.

<fromdate> and <todate> are optional. If not specified, the results returned will be for yesterday, US eastern time (feed circulations are calculated daily, so circulation is not accurate for *today*.)

Dates are always specified in YYYY-MM-DD. e.g. /feeds/1027/stats/2014-07-29

If only one date is specified, data is only returned for that day. If two dates are supplied, data is returned for the date range specified, inclusive.

Data returned is XML similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
      <success code="0" msg="Authorized" />
</rsp>
<stats>
      <day>
            <date>2014-07-29</date>
            <circulation>29324</circulation>
            <reach>2675</reach>
            <views>1234</views>
            <clicks>56</clicks>
            <downloads>0</downloads>
      </day>
</stats>
```

If an element is missing, e.g. downloads for feeds which aren’t podcasts, its value may be assumed to be zero.

Since metrics are calculated daily, there is no need to repeatedly call the API on an intraday basis. Call it once for the relevant day and then persist the results locally.

/feeds/<feedid>/items/<fromdate>/<todate>

This provides the same data as the “stats” API except shows metrics for an individual link (post link or link inside a post). The same metrics are returned, and the same remarks about dates and frequency apply. The XML structure is slightly different, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
      <success code="0" msg="Authorized" />
</rsp>
<stats>
      <day>
            <date>2014-07-29</date>
            <item>
                  <title><![CDATA[Rewind the Week - Robot Edition]]></title>
                  <url>http://www.feedblitz.com/feedblitz-rewind-the-week-robot-
edition/</url>
                  <reach>2675</reach>
                  <views>1</views>
                  <clicks>56</clicks>
                  <downloads>0</downloads>
            </item>
```

```
        </day>
</stats>
```

FeedBlitz only knows the titles of links contained in the RSS metadata. If a click is recorded on a link within a post, and FeedBlitz does not know its title, the <title> element above will replicate the URL of the clicked link.

/feeds/<feedid>/readers/<fromdate>/<todate>

Provides information about how a feed is consumed and by what reader. Same remarks about listid and dates apply here as to the stats and items resources.

FeedBlitz differentiates between four kinds of RSS feed consumers:

1) Search engines, which do not count towards a feed's circulation.
2) RSS aggregator services, like feed.ly, which centrally access a feed on behalf of many subscribers.
3) Individual RSS readers, typically installed by a user on their desktop, tablet etc.
4) Browsers, where the consuming application isn't a specialized RSS reader.

FeedBlitz extracts a "shorthand" identity for a visiting user agent. If that's a known user agent, a longer user-friendly name will be presented. If not, just the short hand user agent element. FeedBlitz doesn't persist entire user agent strings in its metrics database.

The returned XML looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
        <success code="0" msg="Authorized" />
</rsp>
<readers>
        <day>
                <date>2014-07-29</date>
                <reader>
                        <useragent>feedfetcher</useragent>
                        <name>Google Reader</name>
                        <subscribers>274</subscribers>
                        <type>RSS Service</type>
                </reader>
        </day>
</readers>
```

Custom Field APIs

Custom fields in FeedBlitz are data points beyond the subscriber's email that are available to the account. There are no limits to the number of custom fields in a single account; they are currently global to the entire account, however. Field data are stored encrypted using AES.

The following resources are supported. Currently, only the GET method is supported.

Resource: **`/fields`**

Returns XML metadata for the fields defined by the publisher:

```
<?xml version="1.0" encoding="UTF-8"?>
  <feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <fields>
    <count>1</count>
    <field>
      <id>234956</id>
      <name>ReferrerName</name>
      <type>string</type>
      <default />
      <order>2</order>
      <required>0</required>
      <hidden>0</hidden>
      <description>Who referred you?</description>
      <help>Did a current member of program give you your special invite? If so, 
please enter their name here so they receive proper credit.</help>
      <created>2015-07-28 19:18:04 -0500</created>
      <updated>2015-07-28 19:22:13 -0500</updated>
    </field>
  </fields>
</feedblitzapi>
```

Resource: **`/fields/id/<fieldid>`**

Returns field metadata for a single field ID, as in:

```
/fields/id/234956
```

Resource: `/fieldvalues/subscriber/<subscriberid>`

Returns the field values stored in the account for the given subscriber, as in:

```
/fieldvalues/subscriber/26562441
```

The returned XML lists all the fields associated with the account, and then any values found for the given subscriber ID.

Sample returned XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <fields>
    <count>1</count>
    <field>
      <id>14473</id>
      <name>DateAdded</name>
      <type>date</type>
      <default>1969/07/04</default>
      <order>0</order>
      <required>0</required>
      <hidden>1</hidden>
      <description />
      <help />
      <created>2013-11-05 18:36:29 -0500</created>
      <updated>2013-11-05 18:36:29 -0500</updated>
    </field>
  </fields>
  <subscribers>
    <count>1</count>
    <subscriber>
      <email>feedmanager@example.com</email>
      <publishervalues>
        <count>1</count>
        <fieldvalue>
          <id>32605820</id>
          <fieldid>14473</fieldid>
          <name>DateAdded</name>
          <value>1969/07/04</value>
          <created>2015-07-24 10:54:57 -0500</created>
          <updated>2015-07-24 10:54:57 -0500</updated>
        </fieldvalue>
      </publishervalues>
    </subscriber>
  </subscribers>
</feedblitzapi>
```

Custom Dual Opt-In Registration Activation Message

The dual opt-in message is defaulted by FeedBlitz. Its content is fixed; its formatting depends on the list’s active template and whether or not a SmartForm was used to generate the subscription (in which case the SmartForm’s design is used to format the activation email).

The HTML sent can be overridden as a custom registration email. The content you create will be wrapped by the appropriate template (list or SmartForm).

A custom registration email must contain the special tag <$BlogSubLink$> (suitably encoded). FeedBlitz will insert the activation URL using that tag as the placeholder. It does not create an anchor (<a> tag), just the URL.

The following resources are supported.

Resource: **`/customreg/<listid>`**

Methods are: GET, POST, DELETE.

POST creates the alternative; if you DELETE the entry, FeedBlitz reverts back to the default for the specified list.

Sample returned XML:

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

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <customreg>
    <customreg>
      <title>Activate Your FeedBlitz News Subscription</title>
      <message>&lt;html&gt;
&lt;head&gt;
       &lt;title&gt;&lt;/title&gt;
&lt;/head&gt;
&lt;body&gt;
&lt;p&gt;Hi!&lt;/p&gt;
&lt;p&gt;You&amp;#39;re almost ready. Just one last step.&lt;/p&gt;
&lt;p&gt;&lt;a href=&quot;&lt;$BlogSubLink$&gt;&quot; title=&quot;Activate your
account&quot;&gt;&lt;b&gt;&lt;u&gt;Click here to activate your
subscription&lt;/u&gt;&lt;/b&gt;&lt;/a&gt;.&lt;/p&gt;
&lt;/body&gt;
&lt;/html&gt;</message>
    </customreg>
  </customreg>
</feedblitzapi>
```

Mailings Resource

Determines the cycle ids used for the /metrics resource to get open, click through and other data about subscriber engagement.

Fromdate and todate are optional. When specified, do so as YYYY-MM-DD. They are always US eastern time.

- If both are omitted, the date used is yesterday.
- If one is provided, the date used is for that day
- If both are provided, the date range is used.

The cycle ID is returned along with its nominal start time. Frequency denotes the type of mailing, and the time of day indicates the time of day (e.g. 4pm-7pm) for that slot. Positive numbers represent the interval, i.e. 60 means hourly; negative numbers map to <turbo> elements in a list's schedule. The meaning of polltime varies with the mailing frequency.

Resource: **`/customreg/<listid>/<fromdate>/<todate>`**

Methods: GET only

Sample response:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0">
<rsp stat="ok">
<success msg="Authorized" code="0"/>
</rsp>
<mailings>
<mailing>
       <cycleid>5324267</cycleid>
       <date>2016-10-06 10:30:00</date>
```

```
	<frequency>-5</frequency>
	<polltime>630</polltime>
	<timeofday>-1</timeofday>
</mailing>
<mailing>
	<cycleid>5364797</cycleid>
	<date>2016-11-09 11:00:00</date>
	<frequency>-5</frequency>
	<polltime>660</polltime>
	<timeofday>-1</timeofday>
</mailing>
</mailings>
</feedblitzapi>
```

Introduction to FeedBlitz Funnels (Autoresponders, ARs)

An autoresponder (AR) is a predetermined sequence of emails (entries) that go out to a subscriber, typically with a fixed time between steps in the sequence, initiated by an event such as:

- Subscribing to the AR directly
- A transaction, such as making a purchase, downloading a white paper or e-book, etc.
- Joining a list that triggers a child AR sequence

There are two types of autoresponder:

- Traditional, where every subscriber starts at the beginning of the sequence
- Countdown, where the subscriber starts mid-way through the sequence, based on a fixed end date.

FeedBlitz does not currently support countdown autoresponders.

Prior to the release of these capabilities, FeedBlitz's autoresponders had the following capabilities:

- Delivering a fixed sequence of emails based strictly on time delays.
- Optionally delaying the start to a specified day of the week and time.
- Optionally using on subscribe, on unsubscribe and on sequence completion triggers to change a subscriber's other subscriptions.

Further, no useful metrics were collected about a subscriber's interactions or an individual entry.

FeedBlitz's Autoresponder Capabilities

With the additional new features, made available via this API, FeedBlitz ARs can now, in addition to the above:

- Perform other tasks at each step in the sequence:

  - Add or delete a tag or custom field
  - Jump to a different entry in the sequence instead of the next one
  - Change a subscription to another list at any point in the sequence
  - Fetch a URL
  - Add or drop the subscriber from a group
- Make performing the task conditional, depending on:
  - The value of a tag or custom field
  - Whether a segment expression evaluates to true for that subscriber
  - Whether the subscriber is subscribed to or unsubscribed from a different list
  - Whether the subscriber has opened or clicked inside a given entry in the AR
- Collect metrics on a per-entry and per-subscriber per-entry basis.

Deferment Schedules

A deferment schedule is the ability to tell an autoresponder that it may only send (or more generically, process) an entry:

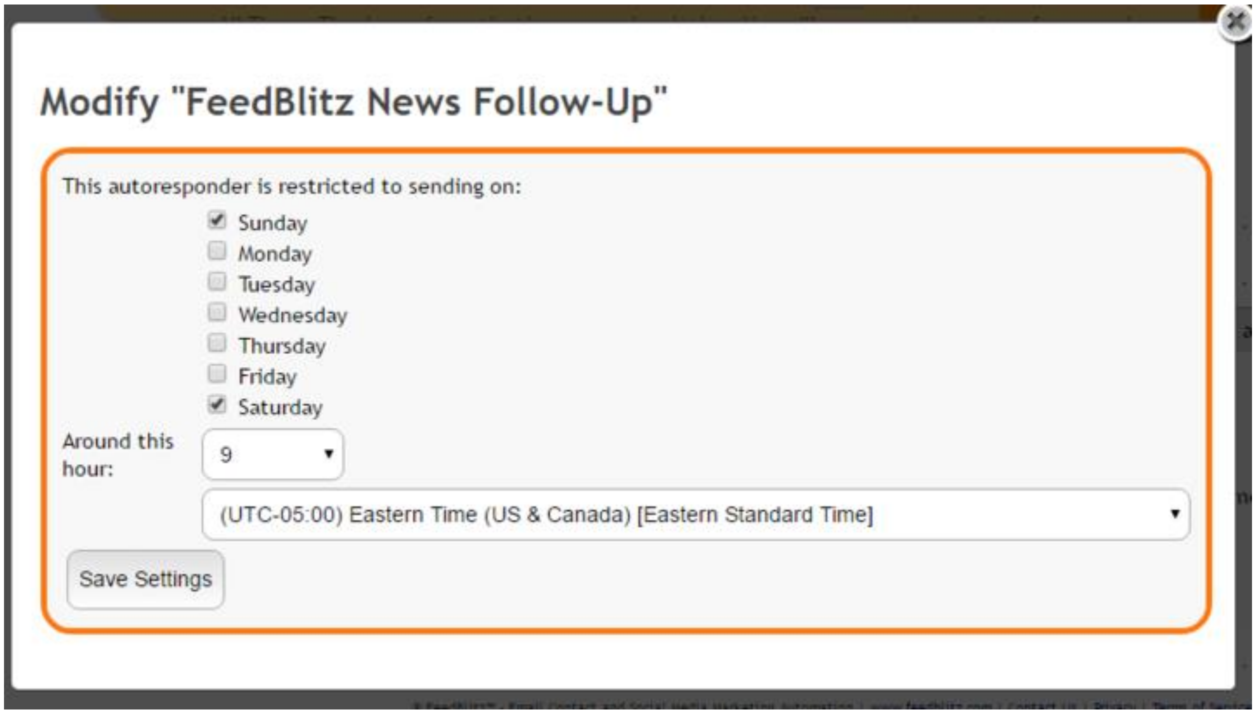
- On the specified days of the week.
- At the specified time.
- In the specified time zone.

If an entry would be processed but doesn't match the deferment schedule, then processing (including all actions) is deferred for an hour and then it is checked again. This hourly check repeats until processing is allowed by the deferment schedule, at which point the current entry / entries are processed, and the sequence picks up again.

The default deferment schedule is to have no deferment schedule. In other words, an autoresponder entry may be processed at any time, which maintains backwards compatibility with current autoresponders.

Deferment schedules are currently global to the selected autoresponder (i.e. they apply to all entries in the sequence); may not be overwritten at the entry level; and while multiple days may be selected, only one time may currently be used.

The UI for this in FeedBlitz looks like this, and is found via the settings screens for the autoresponder:



Modify "FeedBlitz News Follow-Up"

This autoresponder is restricted to sending on:

- [✓] Sunday
- [ ] Monday
- [ ] Tuesday
- [ ] Wednesday
- [ ] Thursday
- [ ] Friday
- [✓] Saturday

Around this hour: 9

(UTC-05:00) Eastern Time (US & Canada) [Eastern Standard Time]

Terminology

An *autoresponder* (called a funnel in the FeedBlitz user interface) is an individual list that defines one or more *entries* in a temporal sequence. Each *entry* is processed in order as the subscriber progresses through the autoresponder's sequence. An entry may have one or more *actions*, and an action may have one or more *conditions*. If an entry does have more than one condition, all the conditions must be satisfied for that action to be taken (i.e. the condition results are combined with a logical AND).

Benefits

With these capabilities, it is possible to take different actions and send different messages based on tag (e.g. campaign), field value (e.g. purchase history), create follow-up campaigns or automate list hygiene programs. It is permitted to have AR entries in the sequence that do not send email at all.

Remarks and Restrictions

- There is no default action for entries created via the API. If the entry is to be emailed, the email action must be specified.
- There is no default condition for entries created by the API, which implies that the action is to be taken always.

The FeedBlitz AR API Reference

This is a REST API, and documentation follows the approach laid out in the API v2 guide.

Managing an AR via the FeedBlitz APIs

The core elements of a FeedBlitz AR are the same as a regular mailing list. You manage an AR's metadata (e.g. its title) via the `/syndications` resource. An AR has a `<turbo>` value of -100, and will also be returned with the `<isauto>` element have a value of 1.

To create an AR from scratch, you can PUT to the REST API with the appropriate `<turbo>` value, or create a list using one of the other API methods available, and then POST to the REST API to change the `<turbo>` value.

The new AR API resources described here therefore relate to the entries *within* an AR, and not the *containing* autoresponder syndication.

Entry Status

FeedBlitz stores three types of entries: Published (status is "ok"), draft ("draft") and for entries deleted by the API ("deleted"). Only published entries are evaluated for a subscriber by the autoresponder, and deleted entries are not visible in the FeedBlitz UI. A request for an AR's entries via the API returns all the relevant entries, no matter what their status.

AR Content Resources

```
/autoresponder/<listid>
/autoresponder/<listid>/entries
/autoresponder/<listid>/entries/<entryid>
```

The `<listid>` is the AR's list id in FeedBlitz (and is reported by the `/syndication` or `/autoresponders` (plural) resource). It is required for all AR operations.

GET Method

You can GET any of the resources listed above. The XML returned varies, depending on the resource requested.

```
/autoresponder/<listid>
```

Returns very high level information about the AR's sequence. Example XML:

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<autoresponder>
<id>634210</id>
<deferdays>2,3,4,5,6</deferdays>
```

```
<defertime>600</defertime>
<defertz>20</defertz>
<autoresponderentries>
<count>8</count>
</autoresponderentries>
</autoresponder>
</feedblitzapi>
```

A GET of the entries resource: `/autoresponder/<listid>/entries` provides the entire XML dump of the AR's entries, e.g.

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<autoresponder>
<id>503829</id>
<deferdays>2,3,4,5,6</deferdays>
<defertime>600</defertime>
<defertz>20</defertz>
<autoresponderentries>
<count>1</count>
<autoresponderentry>
<id>672</id>
<status>ok</status>
<subject>Did you get yours yet? [Free]</subject>
<guid>1f96a4a4-0f00-11e5-ad3a-0019998b9c3f</guid>
<html>&lt;p&gt;
                    &lt;span style=&quot;font-family:arial,helvetica,sans-
serif;&quot;&gt;Hi,&lt;/span&gt;&lt;/p&gt;
…
</html>
<link>http://www.feedblitz.com/f/f.fbz?articles=503829&amp;guid=1f96a4a4-0f00-11e5-
ad3a-0019998b9c3f&amp;ajax=4</link>
<ordinal>1</ordinal>
<delay>2880</delay>
<actions>
<action>
<type>mail</type>
</action>
</actions>
</autoresponderentry>

… more entries …

</autoresponder>
</feedblitzapi>
```

Note that elements are presented in increasing order of delay and ordinal; i.e. the first entry is the one that will be evaluated first for a new subscriber.

IMPORTANT

Physical order of actions and conditions is significant. Actions are presented in the order in which they will be taken, first to last, and conditions are presented for each action the same way. It is therefore possible for an action to occur (e.g. apply tag X) that will affect subsequent actions for the same entry (e.g. don't do this if the subscriber has tag X). FeedBlitz won't warn you about this class of error. Reordering of actions and conditions via the API is achieved via a POST to the correct resource.

Finally, getting the specific entry XML generates the XML for just that entry within the sequence. e.g. `/autoresponder/<listid>/entries/672`

Deferment Schedule Elements

Element	Comment
deferdays	Comma separated numbers 1 (Sunday) to 7 (Saturday) indicating the allowed days. An empty element (or its absence in a GET) indicates ALL days are allowed (i.e. <deferdays/> and <deferdays></deferdays> and <deferdays>1,2,3,4,5,6,7</deferdays> are all equivalent.). The order of the numbers is inconsequential, and repeated day numbers ignored.
defertime	Number of minutes in the time zone when emails should go out. Empty / absent implies any time. FeedBlitz runs autoresponders every hour, so it only makes sense (currently) to use increments of 60. 600 means 10am in the selected time zone. Actual mailing times are approximate and will happen shortly after the specified time. Default is any time, allowed values are 0 (midnight) to 1439 (11:59 pm).
defertz	The time zone of the time of day. This is a numeric ID and must match the <id> of the timezone returned by the /timezones resource. The default value is US Eastern, ID 35. If you persist this as a number be aware that it is signed, and that negative IDs exist.

You can set these values with a PUT or POST to the relevant autoresponder resource, including when you update /autoresponders/<id>/entries

If you combine updating one or more entries with a schedule deferment, and the deferment schedule is OK but the entry action isn't, be aware that the deferment schedule will update regardless.

It is OK to repeatedly send deferment schedule data to the API even if it hasn't changed; FeedBlitz will only update its persistent store when a value changes. It is also OK to change on at a time in a post. i.e. you can update the `<defertime>` element without specifying `<defertz>` and `<deferdays>` in your XML payload.

By way of example, these elements specify that the AR can only process actions Monday to Friday at 10am, US Central Time.

```
<deferdays>2,3,4,5,6</deferdays>
<defertime>600</defertime>
<defertz>20</defertz>
```

AR Entry Elements

Element	Comment
id	FeedBlitz assigned ID for the entry. Not editable.
status	ok, draft, or deleted.

subject	Subject line of email to be sent, if actions allow
html	HTML markup for the email to be sent, entity encoded to make the XML valid
guid	Unique ID. You may provide / edit this, in which case it is you responsibility to ensure its global uniqueness. If absent, FeedBlitz will provide a GUID.
link	Link to a web page associated with the entry, used when formatting the mailing. If blank, FeedBlitz will generate one, which will be a web based version of the html, formatted with the relevant template.
ordinal	An integer representing the entry's position in the sequence. Increasing, but not necessarily monotonically. Not necessarily static; it may change after an AR is saved, either via the API or the UI.
delay	The nominal delay (in minutes) from the start of the sequence that this entry should be evaluated after. NB FeedBlitz AR's are currently processed hourly, so entries separated by less than an hour will likely be processed at the same time.
usetemplate	If "0" the AR does not use the AR's expected template; instead the HTML is sent as is. If `<template>` is missing it is assumed to be "1" and will use the AR's setting for the template.

AR Action Elements

There will typically be one or more <action> elements associated with an entry (the default action for AR's created via the UI is to always send the email; there is no default action for entries created via the API – you must provide one).

Element	Comment
type	The type of action (See below). Required. The type element must also be the first element if other elements at the action level (i.e. <anycondition> are specified.
anycondition	Whether all conditions must be satisfied for the action to take place (the default), or whether any condition being satisfied enables the action. Valid values are empty, 0 (these are equivalent, all conditions must be satisfied, i.e. logical AND), or 1 (any condition being satisfied triggers the action, i.e. logical OR)

Every `<action>` has a `<type>` element. Other elements vary, depending on the `<type>`. The following types are implemented:

Type	Comment
mail	Sends the <html> email defined for the entry
cf	Sets a custom field ("cf") or tag value.
list	Adds to or unsubscribes from a different list in the publisher account
jump	Set the next entry in the sequence; overrides the standard temporal delivery.

url	Sends an HTTP GET to the specified URL.
group	Adds to, or removes, an email address to / from a static group

Currently the API does not validate action types. If you create an entry with a type other than those specified above, it will be persisted by the API, but will do nothing. It is therefore possible to use custom action types to persist your own information in an action. We *strongly* advise using a namespace-style declaration of the type (e.g. MegaCorp::App::ActionType) to avoid potential conflicts with future FeedBlitz defined actions. FeedBlitz will never use a namespace-style action type.

The next few sections describe the elements associated with each FeedBlitz action type.

mail action

There are no other action elements for the `mail` type. In other words, the presence of a `mail` action tells the entry to send the defined email, if the associated conditions permit.

cf action

element	Comment
fieldname	Required: Custom field name or tag to set. NO WHITESPACE.
value	Required: The value to set. For tags, this is typically “1” or “0” (w/o the quotes). You may specify an empty value here.
tag	Optional: Set to 1 if this is a tag – used when naming a new field

The `<fieldname>` you set does not have to have been previously defined by the account. If it is new, then the custom field will be created by FeedBlitz as a hidden field (i.e. not visible to the publisher’s subscribers). If the `<tag>` element is 1 then the description of the newly created field will be prepended by the text “Tag: ” – otherwise the description of the field will mirror the fieldname. Fieldnames may not have spaces at FeedBlitz, and are case insensitive.

list action

element	Comment
listid	Required: A List ID to affect. “*” (meaning “all”) is permitted for unsubscribes only.
subscribe	Required: 1 – to subscribe, 0 - unsubscribe

The `<listid>` must be a list or AR owned by the publisher of the AR requesting the action. If the ListID is specified as “*” (w/o the quotes) AND the action requested is unsubscribe, then the subscriber will be unsubscribed from ALL lists and ARs owned by the publisher on which that subscriber is currently active. The action of unsubscribing to * will remove the subscriber from the autoresponder initiating the request.

Both the subscribe and unsubscribe actions are silent, in that the subscriber is not informed.

If the action is to subscribe a user, and the specified list is an autoresponder, the action will start the target autoresponder sequence (typically this means: send the first email in that sequence, but it actually means: evaluate the first entry and process its actions if that is to be done immediately, i.e. with a delay of zero minutes).

If the action is to subscribe a subscriber, that action will not take place if the subscriber has previously unsubscribed or deleted a prior subscription to the target list. i.e. this action cannot be used to override a previously taken removal.

jump action

element	Comment
entryid	Required: The entry id in this AR that the subscriber should get next
delay	Required: The number of minutes from now (i.e. action evaluation time) to delay before the entry is sent

If the entry exists, the system will ensure that the next entry the subscriber receives will be the one specified after the relevant delay. After that, the subscriber will continue with the time-based sequence from that point forward, unless other jump actions are specified.

So, for example, it is possible to create a continuous weekly email by having an action on the 7th entry jump back to the 1st with a delay of 1440.

It is permitted for an entry to jump back to itself. FeedBlitz will not prevent you from creating infinite loops – in fact it's a feature, per the weekly AR described above. Absent other actions, the only way to stop such a loop is for the subscriber to unsubscribe.

Note that because FeedBlitz processes autoresponders hourly, jump delays should be at least 60 minutes. Setting a delay less than that risks the destination entry from being missed, in which case the subscriber will be sent the one after that in the temporal sequence after the relevant delay.

url action

element	Comment
url	Required: The URL to GET

FeedBlitz will append the following parameters to the URL query string (or add a query string if there isn't one specified in the URL):

element	Comment
email	The email address of the subscriber
subid	The subscriber's ID at FeedBlitz
listid	The autoresponder's list ID
entryid	The entry ID of the entry triggering the action

We recommend, but do not require, SSL for the URL in order to secure the email address of the visitor in flight. FeedBlitz will call the URL exactly once, and will not reattempt the fetch in case of failure.

group action

element	Comment
groupid	Required: The id of the group
join	Required: The action to take. Must be either "join" or "drop" (without the quotes)

Groups are static collections of email addresses. They can be used to restrict mailings, or to filter bulk subscriber operations. Adding to or dropping an email address is silent; the address is not emailed, nor is any validation or opt-in performed.

Action Conditions

An action may have one or more conditions associated with it. When the action is processed, any conditions present will be evaluated for the subscriber. If multiple conditions are specified, they must all be true for the action to take place. If no conditions are present, the action always occurs.

Conditions, when present, are specified within an individual action element as follows:

```
<action>
… action parameters …
<conditions>
<condition>
<type> … </type>
… other condition parameters …
</condition>
</conditions>
</action>
```

Condition XML may also be used in certain other FeedBlitz API operations outside of autoresponder actions. When specified like this, some tags may be specified at the `<conditons>` level, such as `<anycondition>, e.g.`

```
<conditions>
<anycondition>0</anycondition>
<condition>
<type> … </type>
… other condition parameters …
</condition>
</conditions>
```

The order of conditions is significant; they are evaluated in the order they appear within the XML.

Like actions, every `<condition>` has a `<type>` element. Other elements vary, depending on the `<type>`. The following condition types are implemented:

Type	Comment
tag	Dependency on a tag or custom field
expression	Dependency on a custom field segment expression
status	Dependency on a subscriber's status (subscribed, unsubscribed) on a list owned by the publisher
activity	Based on whether the subscriber has opened or clicked an entry in this sequence
group	Dependency on whether the subscriber is / is not in a group
suppressionlist	Dependency on whether the subscriber is / is not in a suppressionlist

Currently the API does not validate condition types. If you create a condition with a type other than the ones listed above, it will be persisted, and - when evaluated - unknown condition types will be treated as being TRUE by FeedBlitz. Like actions, it is therefore possible to use custom condition types to persist your own information in a condition. Again, we *strongly* advise using a namespace-style declaration of the type (e.g. MegaCorp::App::ConditionType) to avoid potential conflicts with future FeedBlitz defined conditions. FeedBlitz will never use a namespace-style condition type.

tag condition

element	Comment
fieldname	Required: The custom field name or tag name to use
operator	Required: An operator (see below)
value	Required: The value to test against

If the condition <fieldname> <operator> <value> is true, the action may proceed. If the feldname is not found in the publisher's account, the condition evaluates to false.

The following operators are defined:

operator	Comment
istagged	True if the custom field value stored for the subscriber is not empty
isnottagged	True if the custom field value is not stored, or is empty, for the subscriber
==	See note after table
!=	See note after table
<	See note after table
<=	See note after table
>	See note after table
>=	See note after table
contains	True if the subscriber's stored value contains the specified string
notcontains	True if the subscriber's stored value does not contain the specified string
startswith	True if the subscriber's stored value starts with the specified string

notstartswith	True if the subscriber's stored value does not start with the specified string
in	The test <value> should be a comma separated list (individual entries will be trimmed of leading and trailing spaces prior to evaluation). True if the stored field value matches one of the entries in the list.
notin	Opposite of "in"

For mathematical logical operators, if both the stored value and the test value are numbers, they are converted to integers and the test performed. If either are a non-numeric string, then the relevant string comparison is performed using C++ semantics (e.g. "Z100" is greater than "Z10", but "Z20" is less than "Z9"). String comparisons are always case insensitive.

expression condition

element	Comment
expression	Required: FeedBlitz custom field expression

Evaluates the expression with the full power of FeedBlitz's segment expression engine. The API will not validate the expression's correctness. If the expression is itself invalid, the condition is deemed to be false.

status condition

element	Comment
status	Required: One of the following: subscribed, notsubscribed, unsubscribed, notunsubscribed
listid	Required: One or more list ids owned by the publisher

Note that notsubscribed is not the same as unsubscribed, which is why there are four status options. If a subscriber has never subscribed to a list, notsubscribed is true, but unsubscribed is false.

The unsubscribed and notunsubscribed statuses include where a subscriber did not activate a pending subscription, or were moved from the list by the list's owner (which is stored in the FeedBlitz database as "deleted").

When multiple lists are specified, the condition is TRUE if the condition (subscribed, not subscribed, etc) is true for ANY of the specified lists.

activity condition

element	Comment
event	Required: One of the following: opened, notopened, clicked, notclicked, engaged, notengaged (engaged = opened or clicked)
entryid	Optional*: An entry ID on this autoresponder. Either listid or entryid must be specified.

listid	Optional*: A mailing list ID. Either listid or entryid must be specified.
cycleid	Optional: Restrict activity checks to a specific mailing. Only valid when listid is specified.
joinedbefore	Optional: Restrict activity checks to subscribers whose join date (lastupdated) precedes the date or time specified.
joinedafter	Optional: Restrict checks to subscribers who joined after the specified date or time.
eventafter	Optional: Only true if the open, click or engagement happened after this date or time.
url	Optional: The specific URL to check for click / engagement

For obvious reasons, you should typically test entryids that precede this one in the AR sequence. Note that "clicked" means that the subscriber clicked on any link tracked by FeedBlitz in the entry. It does not track a specific link. This test is useful for list hygiene or follow on campaigns.

All criteria are logical AND; all must be passed for the condition to be true. If both entryid and listid are specified, the listid is ignored and the entry id assumed to belong to the selected AR.

Dates are YYYY-mm-dd, which are taken to mean midnight US eastern on that day. You can specify times as YYYY-mm-dd HH:MM:SS. Times sent in various RFC encodings that include time zone data will be converted to US eastern before processing. (see URL notes for caveats).

eventafter exists to simplify and expedite list cleanups. You can ask for subscribers who joinedbefore <date> and notengaged and eventafter <date>. This eliminates the need to use segment expressions to do this task (and is also considerably faster, as FeedBlitz tracks the last interaction time for each subscriber / list pair). There is no eventbefore entity.

So to track any clicks on an AR, specify this condition, the AR's listid (not the entryid) and use the clicked event.

URL Matching

Important caveats with URL matches:

- The <url> entity is not case sensitive, but is otherwise an exact match. i.e. http://www.feedblitz.com is not the same as https://www.feedblitz.com is not the same as http://www.feedblit.com/

- Date granularity for URLs is at the day level only, so specifying an eventafter of 2016-02-29 means that the condition will be true if the event happened on March 1st or later

- url matching is not currently available with cycleid precision; it is if the subscriber has ever clicked on the URL. Cycleid is ignored when performing a URL check, but is still used for other checks managed by this condition (so you can specify clicked, cycleid and url, and if the subscriber didn't click on the mailing at all then the URL check is moot).

- url checks are only performed for clicked, notclicked, engaged and nonengaged events.

- url checking is currently relatively slow. For performance reasons / best practice, specify a URL that is only used in the relevant mailing, in which case you know it's restricted to a given cycleid anyway. url checking is performed after all other tests and only if all other tests are passed.

group condition

element	Comment
status	Required: One of the following: found, notfound
groupid	Required: The group IDs

The group must be owned by the account. You may specific multiple suppression list IDs to make this condition an OR, as follows:

found = in ANY of the specified groups i.e. IN (A,B,C…X)
notfound = in NONE of the specified groups i.e. NOT IN (A,B,C…X)

suppressionlist condition

element	Comment
status	Required: One of the following: found, notfound
slid	Required: The suppression list's IDs

The suppression list must be owned by the account. You may specific multiple suppression list IDs to make this condition an OR, as follows:

found = in ANY of the specified suppression lists i.e. IN (A,B,C…X)
notfound = in NONE of the specified suppression lists i.e. NOT IN (A,B,C…X)

DELETE method

Applies to:

```
/autoresponder/<listid>
/autoresponder/<listid>/entries/<entryid>
```

In other words, using the API you can delete the entire AR (which leaves the entries alone; instead it marks the syndication as deleted); or you can delete a single entry, which removes that entry from the published or draft sequence, currently setting the entry's status to "deleted."

If you delete a single deleted entry and that entry was the target of pending jumps, the jumps will be reset to use the next entry in the sequence if there is one; the elapsed time will be extended as appropriate. In other words, if the target is entry 2 in the sequence and entry 3 happens 1440 minutes after that, and then 2 is deleted, all the subscribers with pending jumps to 2 will be updated to have pending jumps to 3, and that jump will happen a day (1440 minutes) later than the jump to 2 would have.

If an entry is deleted and there are no others after it in the sequence, subscribers waiting for that entry will be marked as having finished the sequence the next time the autoresponder is processed.

Trying a delete on `/autoresponder/<listid>/entries` will return an error.

POST method

Post edits to one or more entries. You can post to:

```
/autoresponder/<listid>
/autoresponder/<listid>/entries
/autoresponder/<listid>/entries/<entryid>
```

Note: the first two are equivalent.

When you POST, only entries will be affected; use the syndication resource to affect AR metadata.

You should provide the XML that looks like this:

```
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<autoresponder>
<id>634210</id>
<autoresponderentries>
<autoresponderentry>
<id>672</id>
<status>ok</status>
<subject>Did you get yours yet?</subject>
<html>&lt;p&gt;Did you get the free e-book OK?&lt;/p&gt;</html>
<delay>720</delay>
<actions>
<action>
<type>mail</type>
</action>
</actions>
</autoresponderentry>
</autoresponderentries>
</autoresponder>
</feedblitzapi>
```

The list ID and the entry ID should match the resource path you choose. You may include multiple entries (to the same AR) within a single update, in which case you must POST to one of the first two resource paths.

All fields may be edited with the exception of the Entry ID. For the `autoresponderentry` elements, you need only supply those entities you wish to change; the others will be fetched from the existing persistent store.

IMPORTANT: If you are using actions and conditions, and you specify an `<actions>` element, you MUST specify all actions and conditions completely. POSTed actions will *completely* replace the current data, if specified.

To leave existing actions and conditions unaffected, do NOT include `<actions>` in your POSTed XML. To remove all existing actions, specify an empty `<actions/>` (or `<actions></actions>`) element.

If you change a status, the effect is immediate on all live current subscribers. If the sequence time is extended (via a change in `<delay>`) it will currently NOT bring back subscribers who are marked as having finished the sequence.

Entry ordinals are recreated by FeedBlitz. However, you may specify an `<ordinal>` in a POST (and a PUT) to tell FeedBlitz how to order entries that have the same `<delay>`. You may have multiple entries at the same point in the sequence, and if you want entry ID 987 to be processed before entry ID 123 at the same point in time, then POST to 987's resource with an `<ordinal>` less than 123's. That will achieve the effect you desire. When you next GET the resource, the ordinals will be changed to reflect the desired order of operations. We do not recommend persisting ordinals; instead GET the relevant entry, extract its ordinal, and then POST (or PUT) as appropriate.

PUT method

Works for any AR resource. You can PUT any of the elements retrieved in a GET, but `<id>` will be ignored. The following are minimally required: `<status>` (ok or draft) and `<delay>` (0 or positive integer). The `<link>` element must be a valid URL if specified. Otherwise, this is just like a POST. If you specify a mail action but no HTML or subject, FeedBlitz *will* send a blank email. So make sure your data is complete. The returned XML will have the new entry's ID if the entry is added, which you can then use and persist as you wish.

Note: There are no uniqueness constraints within FeedBlitz for a new entry within an AR. If you therefore PUT the same entry repeatedly, it will show up multiple times in the AR, with each new entry having a different entry ID.

AR Metrics Resources

Use these resources to access metrics for an autoresponder as a whole, by entry, or for a specific entry, respectively.

```
/autoresponder/<listid>/stats/<fromdate>/<todate>
/autoresponder/<listid>/entries/stats/<fromdate>/<todate>
/autoresponder/<listid>/entries/<entryid>/stats/<fromdate>/<todate>
```

The `<listid>` is the AR's list id in FeedBlitz (and is reported by the `/syndication` or `/autoresponders` (plural) resource). It is required for all AR operations.

`<fromdate>` and `<todate>` are optional. If no dates are specified, the metrics will be retrieved from yesterday (US Eastern). If you only provide a `<fromdate>`, metrics will be provided for that date only. If you want to specify a given `<todate>`, you must provide a `<fromdate>`. The `<fromdate>` may be the current day, and will retrieve any data stored so far today.

No metrics are available prior to August 22, 2015; the data was not being collected prior to that.

Dates must be provided as `YYYY-MM-DD`. Metrics are calculated in real time; metrics from the prior day are typically stable and may be persisted locally. You may request data from today's date.

Only the GET method is supported. The XML returned by the calls is similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <autoresponder>
    <id>1234567890</id>
    <stats>
      <day>
        <date>2015-08-27</date>
        <sent>60</sent>
        <opens>31</opens>
        <clicks>3</clicks>
        <unsubscribes>0</unsubscribes>
        <complaints>0</complaints>
        <hardbounces>0</hardbounces>
        <softbounces>0</softbounces>
        <forwards>0</forwards>
      </day>
    </stats>
  </autoresponder>
</feedblitzapi>
```

If there are multiple days, there are multiple `<day>` elements, as in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <autoresponder>
    <id>1234567890</id>
    <stats>
      <day>
        <date>2015-08-27</date>
        <sent>60</sent>
        <opens>31</opens>
        <clicks>3</clicks>
        <unsubscribes>0</unsubscribes>
        <complaints>0</complaints>
        <hardbounces>0</hardbounces>
        <softbounces>0</softbounces>
        <forwards>0</forwards>
      </day>
      <day>
        <date>2015-08-28</date>
        <sent>38</sent>
        <opens>22</opens>
        <clicks>2</clicks>
        <unsubscribes>0</unsubscribes>
        <complaints>0</complaints>
        <hardbounces>0</hardbounces>
        <softbounces>0</softbounces>
        <forwards>0</forwards>
      </day>
    </stats>
  </autoresponder>
</feedblitzapi>
```

`<day>` elements are returned in chronological order, and a day element will be returned for every day in the date range, so be careful! `<day>` elements may be empty (apart from the `<date>`) if there was no activity on that date.

When stats are generated at the entry level, the data XML is presented as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <autoresponder>
    <id>1234567890</id>
    <stats>
      <day>
        <date>2015-08-27</date>
        <autoresponderentry>
          <id>670</id>
          <sent>6</sent>
          <opens>8</opens>
          <clicks>1</clicks>
          <unsubscribes>0</unsubscribes>
          <complaints>0</complaints>
          <hardbounces>0</hardbounces>
          <softbounces>0</softbounces>
          <forwards>0</forwards>
        </autoresponderentry>
        <autoresponderentry>
          <id>672</id>
          <sent>3</sent>
          <opens>1</opens>
          <clicks>0</clicks>
          <unsubscribes>0</unsubscribes>
          <complaints>0</complaints>
          <hardbounces>0</hardbounces>
          <softbounces>0</softbounces>
          <forwards>0</forwards>
        </autoresponderentry>
… etc …
    </day>
    <day> … </day>
… etc …
    </stats>
  </autoresponder>
</feedblitzapi>
```

When querying `/entries/stats`, only entries where there was activity will be displayed for any given day. Entry data will be displayed, if there is data to report, even if that entry has subsequently been deleted or moved back to drafts from the current sequence.

Data for a single entry will be returned looking like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
  <autoresponder>
    <id>1234567890</id>
    <stats>
      <day>
        <date>2015-08-27</date>
        <autoresponderentry>
          <id>670</id>
          <sent>6</sent>
          <opens>8</opens>
          <clicks>1</clicks>
          <unsubscribes>0</unsubscribes>
          <complaints>0</complaints>
          <hardbounces>0</hardbounces>
          <softbounces>0</softbounces>
          <forwards>0</forwards>
        </autoresponderentry>
      </day>
  <day> … </day>
… etc …
    </stats>
  </autoresponder>
</feedblitzapi>
```

Note that `<sent>` data is only recorded when the entry's action includes sending an email, and the conditions are met to permit that email to be sent.

Trigger Resources

A trigger is an action that takes place when one of the following events takes place:

- A subscriber activates a subscription to a list;
- A subscriber unsubscribes from a list;
- A subscriber finishes and autoresponder sequence.

These triggers are similar to the List Actions allowed on an AR entry, but predate these capabilities, as well as applying to any list.

Triggers are associated with a FeedBlitz list.

When an event on a list occurs that matches the list's triggers, the trigger can take one of the following actions:

- Subscribe a user to a different list
- Unsubscribe a user from a list
- Unsubscribe a user from all lists owned by the publisher.

A triggering event may have multiple triggers associated with it, so many different actions may be undertaken automatically by the system.

The following resource are available:

```
/triggers/<listid>
/triggers/<listid>/<triggerid>
```

GET Method

Retrieves the XML for all triggers for the list, or for the individual trigger. Returned XML looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<feedblitzapi version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink">
<rsp stat="ok">
<success code="0" msg="Authorized" />
</rsp>
<triggers>
<id>797935</id>
<trigger>
<id>887</id>
<event>Unsubscribe</event>
<action>UnsubscribeFrom</action>
<listid>806207</listid>
</trigger>
<trigger>
<id>888</id>
<event>Unsubscribe</event>
<action>UnsubscribeFrom</action>
<listid>914313</listid>
</trigger>
… etc …
```

```
</triggers>
</feedblitzapi>
```

The `<id>` at the `<triggers>` level is the list id these triggers are associated with.

Trigger Entry Elements

Element	Comment
id	FeedBlitz assigned ID for the trigger. Not editable.
event	The event this trigger is linked to. Valid options are: subscribe, unsubscribe, and Finish. Events are case insensitive; finish only applies to autoresponders.
action	The action to take when the specified event occurs. Valid actions are: SubscribeTo, UnsubscribeFrom, UnsubscribeFromAll, Webhook. Case insensitive.
listid	The list affected by the SubscribeTo or UnsubscribeFrom actions. If the action is "UnsubscribeFromAll" this value will be 0 or "*" and can be ignored.
webhook	For the "Webhook" action, the URL to which subscriber data will be POSTed.

DELETE method

You can delete at either of the trigger resource levels. Deleting an individual trigger ID leaves other triggers intact; deleting at the list level removes them all.

POST Method

Edits one or more triggers.

If you POST XML to a given trigger id resource, the ID specified in the XML you post (which is optional) must match the trigger ID in the resource path. Posting updates to the list id level will edit all the trigger IDs specified in the XML. You may specify only the element(s) you wish to change within an individual trigger; the other elements will be populated by the current values.

PUT Method

Creates new triggers.

You may PUT to either level, they do the same thing (the trigger ID in the resource path will be ignored if you specific it). Any trigger IDs specified in the XML will be ignored; FeedBlitz will assign IDs if the PUT is successful, and return them to you.

API Support

For support, please write to the standard FeedBlitz support address with your case. The standard knowledge base will be updated as necessary.

API Support

For support, please write to the standard FeedBlitz support address with your case. The standard knowledge base will be updated as necessary.