

Table of Contents

TABLE OF CONTENTS	1
API SPECIFICATION (VERSION 1.2).....	2
DESCRIPTION.....	2
STRUCTURE OF API CALLS.....	2
STRUCTURE OF ACTION RESPONSES	3
CALLABLE ACTIONS.....	4
VALIDATE-KEY	4
ANNOUNCE-ARTICLE	5
AUDIT-COMMENT	6
REPORT-FALSE-NEGATIVES.....	9
REPORT-FALSE-POSITIVES	10
GET-STATS.....	11
DEVELOPER GUIDELINES.....	13
SORTING BY SPAMINESS.....	13
RETRAINING.....	13
ANNOUNCING.....	14
TRUSTED USERS.....	14
OPENID.....	14
TRANSPARENCY THROUGH STATISTICS	14

API Specification (Version 1.2)

Description

Defensio is a web service designed to help manage spam on publicly-accessible social web applications such as blogs. Our simple API exposes a handful of functions which return structured responses, making it as convenient as possible for developers to build highly-usable client apps on top of the Defensio service. It is our hope that the Defensio API becomes an important part of the web's ecosystem; particularly that part devoted to easing the burden of spam across all types of public social web platforms.

Structure of API Calls

Calls to actions performed by the Defensio web service adhere to the following structure:

```
http://api.defensio.com/{service-type}/{api-version}/{action}/{api-key}.{format}
```

where:

- `{service-type}` denotes the type of service being requested (currently supported are: "app" (i.e. use of Defensio within an application) and "blog" (i.e. use of Defensio to support a blogging platform); these names are used interchangeably in the examples below)
- `{api-version}` denotes the version of the API being called (currently "1.2")
- `{action}` denotes the desired action (see list supported below)
- `{api-key}` is a valid key unique to the calling user (e.g. an alpha-numeric string)
- `{format}` denotes the desired response format (currently supported are "xml" and "yaml")

Note that all calls to the Defensio web service are POST requests. Further note that when a POST parameter is a URL, it should be submitted as one that is valid and fully-formed (e.g. <http://my.blog.com>).

Required POST parameters are identified with an asterisk (*). Other parameters are optional but **strongly** recommended for improved performance.

Structure of Action Responses

By providing either the .xml or .yaml extension to each action call, the Defensio API allows developers the ability to choose their favorite (or most convenient) structured response format for easy, and consistent parsing. Why do we provide for multiple formats? Simply because we believe that choice is a good thing. In each case, the response contents are equivalent - only the way they are formatted changes. Their structure consists of a depth-2 tree structure, with 'defensio-result' as the root node and a list of key-value pairs as its children nodes. All action responses include a "status" node, whose value ("success" or "fail") denotes whether or not the call could be successfully processed. The "message" node of each response is provided for informational purposes only; it should not be relied upon programatically as it is not guaranteed to provide a constant string. The API version is always returned for convenience.

The following example responses demonstrate the minimal response set both in XML and YAML formats:

Sample XML Response	Sample YAML Response
<pre><?xml version="1.0" encoding="UTF-8"?> <defensio-result> <message></message> <status>success</status> <api-version>1.2</api-version> </defensio-result></pre>	<pre>defensio-result: message: " " status: success api-version: "1.2"</pre>

Note that for each action, HTTP status code "200 OK" will be returned if the credentials provided (key + blog URL) are valid; otherwise the action's .xml or .yaml response will still be returned, but with a HTTP status code "401 Unauthorized".

Callable Actions

The following six actions are exposed by the Defensio web service:

validate-key

This action verifies that the key is valid for the owner calling the service. A user must have a valid API key in order to use the Defensio web service. An example call would look like:

`http://api.defensio.com/blog/1.2/validate-key/key1234.yaml` with the POST parameters as indicated below.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x

Note that the validate-key action will always return HTTP status code "200 OK" even if the credentials provided are invalid (in which case the status node in the action response will be "fail")

announce-article

This action should be invoked upon the publication of an article to announce its existence. The actual content of the article is sent to Defensio for analysis. An example call would look like:

`http://api.defensio.com/blog/1.2/announce-article/key1234.yaml` with the POST parameters as indicated below.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL
article-author *	the name of the author of the article	any string
article-author-email *	the email address of the person posting the article	any valid email address
article-title *	the title of the article	string
article-content *	the content of the blog posting itself	text
permalink *	the permalink of the article just posted	valid URL

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x

audit-comment

This central action determines not only whether Defensio thinks a comment is spam or not, but also a measure of its "spaminess", i.e. its relative likelihood of being spam. An example call would look like: `http://api.defensio.com/app/1.2/audit-comment/key1234.yaml` with the POST parameters as indicated below.

It should be noted that one of Defensio's key features is its ability to rank spam according to how "spammy" it appears to be. In order to make the most of the Defensio system in their applications, developers should take advantage of the spaminess value returned by this function, to build interfaces that make it easy for the user to quickly sort through and manage their spamboxes.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL
user-ip *	the IP address of whomever is posting the comment	xxx.xxx.xxx.xxx
article-date *	the date the original blog article was posted	yyyy/mm/dd
comment-author *	the name of the author of the comment	any string
comment-type *	the type of the comment being posted to the blog	comment trackback pingback other
comment-content	the actual content of the comment	text
comment-author-email	the email address of the person posting the comment any valid email address	valid email address
comment-author-url	the URL of the person posting the comment	valid URL
permalink	the permalink of the blog post to which the comment is being posted	valid URL

referrer	the URL of the site that brought commenter to this page	valid URL
user-logged-in	whether or not the user posting the comment is logged-into the client platform	true false
trusted-user	whether or not the user is an administrator, moderator or editor of this blog; the client should pass true only if blogging platform can guarantee that the user has been authenticated and has a role of responsibility on this blog	true false
openid	the OpenID URL of the currently logged in user. Must be used in conjunction with user-logged-in=true. OpenID authentication must be taken care of by your application.	valid URL
test-force	FOR TESTING PURPOSES ONLY: use this parameter to force the outcome of audit-comment . optionally affix (with a comma) a desired spaminess return value (in the range 0 to 1).	"spam,x.xxxx" "ham,x.xxxx"

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x
signature	a message signature that uniquely identifies the comment in the Defensio system. this signature should be stored by the client for retraining purposes	alphanumeric string
spam	a boolean value indicating whether Defensio believe the comment to be spam	true false
spaminess	a value indicating the relative likelihood of the comment being spam. this value should be stored by the client for use in building convenient spam sorting user-interfaces	a float between 0 and 1, e.g. 0.9893

report-false-negatives

This action is used to retrain false negatives. That is to say, to indicate to the filter that comments originally tagged as "ham" (i.e. legitimate) were in fact spam. An example call would look like:

`http://api.defensio.com/blog/1.2/report-false-negatives/key1234.yaml` with the POST parameters as indicated below.

Retraining the filter in this manner contributes to a personalized learning effect on the filtering algorithm that will improve accuracy for each user over time.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL
signatures *	list of signatures (may contain a single entry) of the comments to be submitted for retraining. note that a signature for each comment was originally provided by Defensio's audit-comment action	comma-separated list of alphanumeric strings

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x

report-false-positives

This action is used to retrain false positives. That is to say, to indicate to the filter that comments originally tagged as spam were in fact "ham" (i.e. legitimate comments). An example call would look like:

`http://api.defensio.com/blog/1.2/report-false-positives/key1234.yaml` with the POST parameters as indicated below.

Retraining the filter in this manner contributes to a personalized learning effect on the filtering algorithm that will improve accuracy for each user over time.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL
signatures *	list of signatures (may contain a single entry) of the comments to be submitted for retraining. note that a signature for each comment was originally provided by Defensio's audit-comment action	comma-separated list of alphanumeric strings

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x

get-stats

This action returns basic statistics regarding the performance of Defensio since activation. An example call would look like: `http://api.defensio.com/app/1.2/get-stats/key1234.yaml` with the POST parameters as indicated below.

POST Parameters

Parameter	Description	Possible Values
owner-url *	the URL of the site owner using the service	valid URL

Response Structure

(note: all response nodes are children of parent node: defensio-result)

Response Node	Description	Possible Values
status	indicates whether or not the key is valid for this blog	success fail
message	a message provided by the action if applicable	any string
api-version	the version of the API used to process request	x.x
accuracy	describes the percentage of comments correctly identified as spam/ham by Defensio on this blog	a float between 0 and 1, e.g. 0.9983
spam	the number of spam comments caught by the filter	integer
ham	the number of ham (legitimate) comments accepted by the filter	integer
false-positives	the number of times a legitimate message was retrained from the spambox (i.e. "de-spammed" by the user)	integer
false-negatives	the number of times a spam message was retrained from comments box (i.e. "de-legitimized" by the user)	integer
learning	a boolean value indicating whether Defensio is	true

	still in its initial learning phase	false
learning-status	more details on the reason(s) why Defensio is still in its initial learning phase	any string

Developer Guidelines

The Defensio API (below) describes how applications interface with the Defensio web service, but does not suggest how the API should be used for maximum effectiveness. Below we provide guidelines for developers to help them extract the most benefit from the service, and thus create applications that are maximally powerful for the end-user.

Sorting by Spaminess

One of Defensio's most valuable features is its ability to provide a mathematically meaningful and consistent spaminess value for each comment it processes through the audit-comment action. We strongly recommend (almost to the point of insisting) that this function's spaminess return value be used in applications as the primary way to sort a list of spam messages, thus making it trivially easy for users to locate false-positives amongst their spam.

Spaminess might further (at the developer's option) be used in conjunction with a color gradient to provide a simple visual cue for the user in regards the spaminess range of each comment. Also, if the sheer volume of comment spam is onerous on the user, the developer might consider allowing users to specify a spaminess threshold above which "obvious" spam messages are hidden.

Retraining

Client applications should (again, strongly recommended or performance will greatly suffer) provide the ability for the user to identify and signal to our servers the presence of erroneous spam/ham filtering. The retraining actions report-false-positives and report-false-negatives are provided to this end. By informing our servers of such errors, users can contribute to a personalized learning effect on the filter's performance -- this means that the only way for a user to experience improved performance over time, is by correcting the filter when it goes astray. Developers should keep in mind that in order to use these functions they must store, on the client-side, the comment signature returned by the audit-comment action.

Announcing

To help personalize and populate the filter, every time a new article (e.g. blog post) is posted by the user it should (if feasible) be sent to Defensio's servers using the announce-article function. This will help improve filter performance by training the server with decidedly un-spammy content.

Trusted Users

If the client application is able to ascertain trusted identity credentials for the individual posting a comment (e.g. a blog's editor) then the application should pass `trusted_user=true` to the filter's audit-comment action. This aids in directing and personalizing filter performance.

OpenID

Defensio supports OpenID in its `audit-comment` action since API version 1.2. This new functionality leads to improved filtering accuracy. You are therefore encouraged to implement OpenID in your application and to pass along the information to Defensio whenever possible.

It is important to understand that Defensio does NOT perform any OpenID authentication. It is the responsibility of your application to do so.

Once an OpenID authentication has been properly performed, you can pass the user's OpenID URL to Defensio through the `openid` parameter of the `audit-comment` action. This parameter should only be supplied along `user-logged-in=true`.

Transparency through Statistics

We believe that users benefit from full transparency. To that end, developers should leverage Defensio's `get-stats` action in order to provide users with a summary of the filter's performance. At minimum, accuracy should be displayed in a convenient location (if it does not detract from the application's user experience).