# Zemanta service

Everything you need to know about Zemanta API beside the specification

by Andraž Tori, Tomaž Šolc

01/17/11

## Table of Contents

# Introduction

Digitalization of content started by putting written word into ASCII form. HTML and web eventually enabled linking and interleaving with other types of media such as images, sound and video. Flash and Javascript further enabled interactive widgets such as map views. Lately the content on the web is moving into direction of explicitly exposing relations between pieces of data. General intention of explicit relations is to allow computers to comprehend what pages are saying and use that knowledge to offer better service to humans when interacting with them.

While authoring text comes naturally for educated human beings many reasons exist why **creating fully featured web content is still cumbersome experience**. Those reasons can be split into two main categories. One issue is efficiently of finding the right content that should be included or connected to. This usually takes a lot of time. The other issue is efficiently telling the computer the nature of relationships between our content and external content and data. This usually requires skills and knowledge from depths of specifications and standards.

Zemanta is the service that tries to resolve those two issues by providing **semi-automatic process of content enrichment** to be more appealing to humans and at the same time placing it in correct relations to other content in a way computers can understand.

At one point in future we would imagine for computers to be able to help us manage our life with the knowledge they have about us. Imagine a computer that sends you on a date, or one that automatically discovers new career opportunities and arranges a job interview for you. Or suggests how to better manage your finances. But first things first. We need to put structure and candy into plain old text.

This document tries to provide needed information to a developer that wants to use Zemanta API. Explanations do not try to formally document the API since that documentation is found at http://developer.zemanta.com/docs , but provide a broader view about how Zemanta suggestions work and how they should be presented to the author inside authoring application.

From the infrastructure viewpoint Zemanta web service is a server that application sends content to and gets suggestions from. HTTP protocol is used with standard JSON or XML response formats.

Authoring application such as a content management system provides suggested content to the author and she can select appropriate pieces for incorporation into her own content. It is important to be aware that when author publishes her work the content suggestions are baked inside it and Zemanta service does not participate when content is served or read.

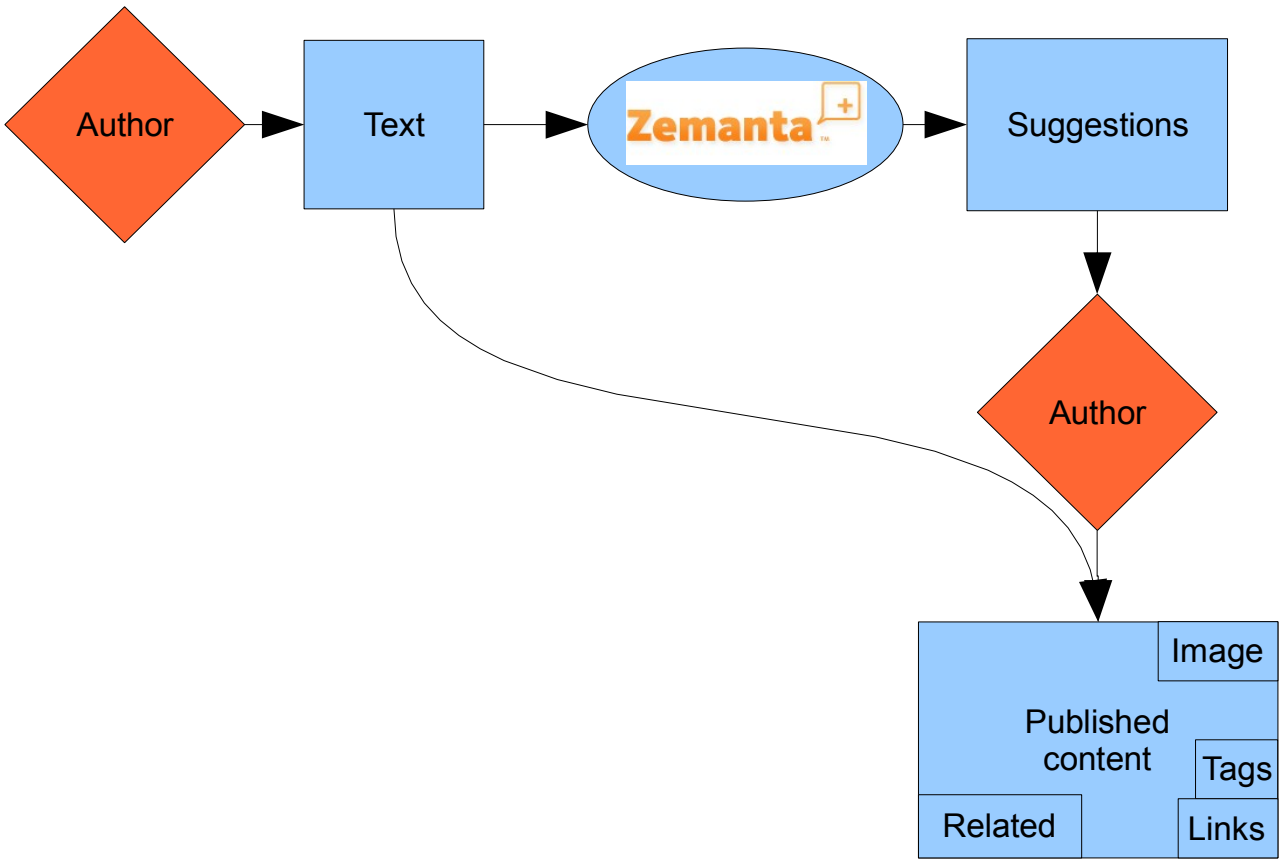


*Illustration 1: Authoring process with Zemanta*

This document first describes what Zemanta offers as suggestion, what information accompanies those suggestions and how that information should be used. Zemanta works from infrastructure and technological perspective. At the end there are examples of how to use Zemanta in different programming languages.

## Basics

Content on the web is generally authored in language called HTML (HyperText Markup Language) or languages that are later translated to HTML such as BBCode. These languages enable linking and inclusion of other materials into web pages.

For the purpose of this document content consists of title, main body of text and supporting content such as links, images and tags. Zemanta service is a web service that authoring software submits title and text into and gets suggestions as the response.

When we were deciding on what information to send back to the API client, we had the following in mind:

- human needs to decide if suggestion is correct and desired,
- human needs to decide if planned usage complies with the license of suggested content and
- computer needs to be able to correctly incorporate the suggestion into the existing content.

Zemanta service currently supports four fundamental types of suggestions:

- images
- related articles
- in-text links (markup)
- tags (keywords)

Each of those suggestions can mean qualitative improvement of existing submitted content. Images are meant to illustrate or present the topic or concept being talked about. Related articles help readers find more about the topic at hand and also make it easy for writer to become engaged and interwoven in the sub-web of writers with common interest. In-text links help readers when they don't know what the people, places or phrases mean or let them explore details of very specific concepts mentioned in the text. Tags help users navigate between topics inside and across sites. Also tags make indexing of content easier and more accurate.

Basis for Zemanta's suggestions are state of the art algorithms for processing natural language, machine learning, information retrieval and similar. Text at hand is related to background knowledge. Newly generated and gathered information is used to create context which is in turn used to query internal and external repositories for suggestions.

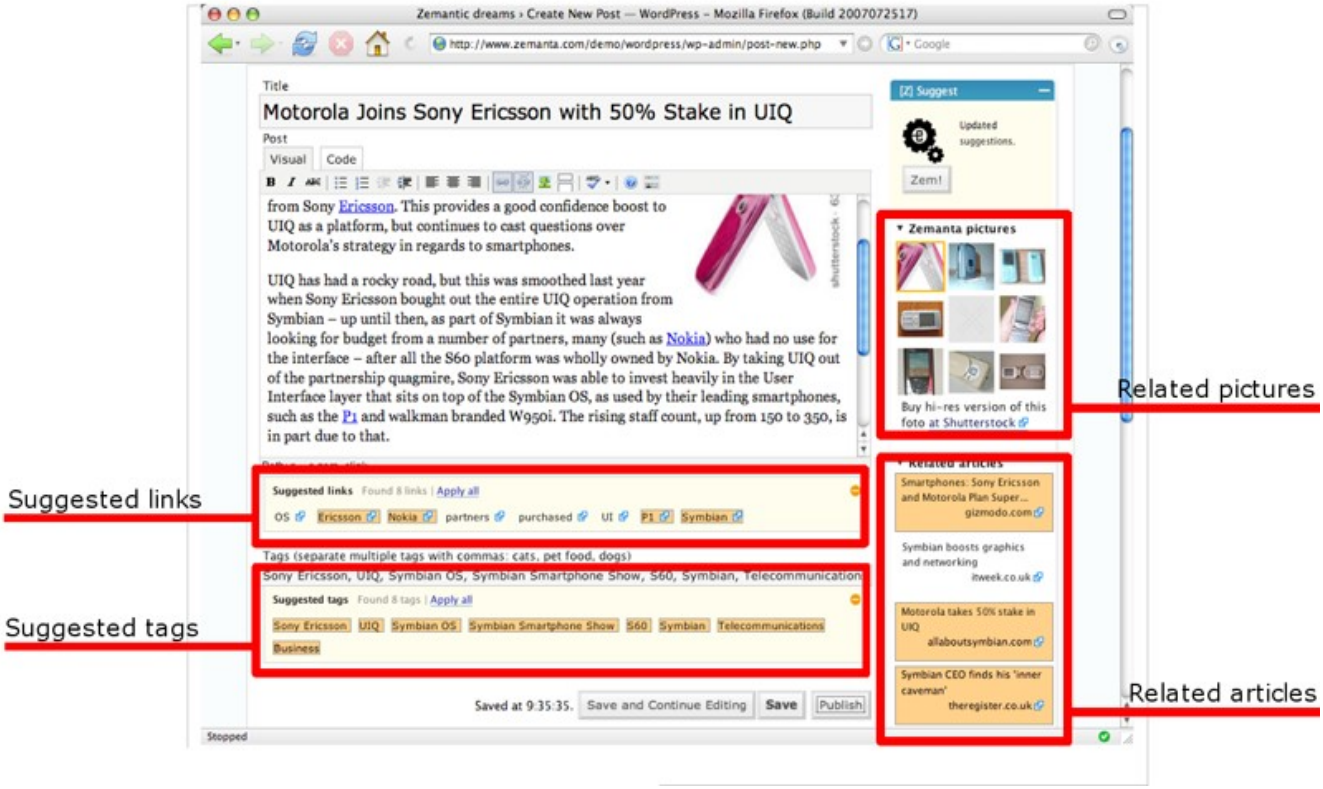


*Illustration 2: Four basic types of Zemanta suggestions shown in blog writing application*

Zemanta creators imagined first adopters were going to be bloggers and other people authoring large amounts of web content. That is why Firefox browser extension was created as the first delivery vehicle of this technology. **However we have no illusions about being able to imagine your way of using Zemanta, that is why Zemanta API was born**. API enables integration into open source and proprietary Content Management Systems and content hosting platforms. Suggestions can be integrated into word processing applications or email clients. Maybe into a game or maybe into a mobile platform. All this is possible as long as you adhere to conditions specified in Terms of Service at http://developer.zemanta.com/API_terms_of_use.

If there is anything unclear throughout this document about Zemanta API or about Zemanta in general, please do let us know via email at support@zemanta.com, if you have technical questions please leave them at developers' forums at http://developer.zemanta.com/forum if questions are about browser extensions, leave them at http://getsatisfaction.com./zemanta

## Roles in Zemanta ecosystem

Five primary roles participate in “classical” Zemanta ecosystem. The diagram shows them in orange.

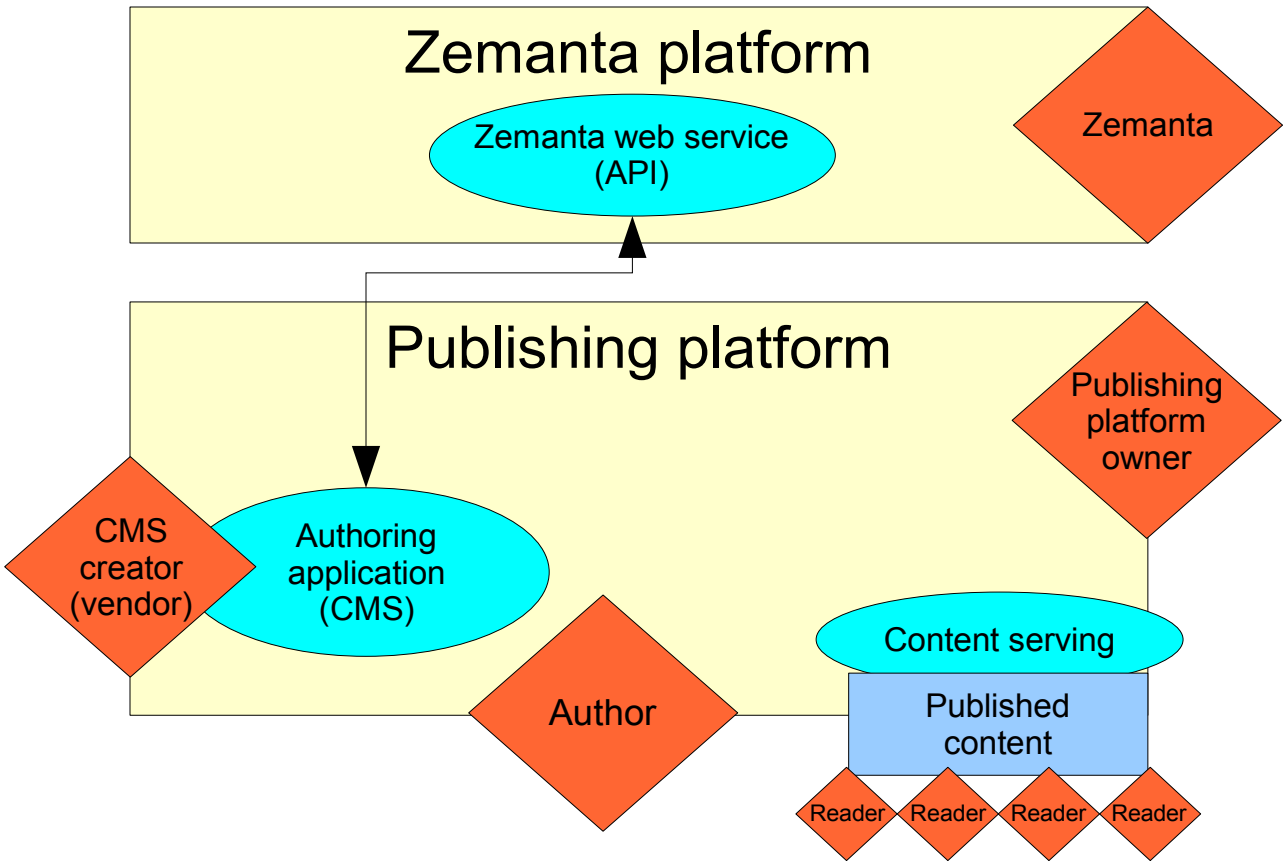


*Illustration 3: Roles in "classical" Zemanta ecosystem*

It is not uncommon that a single person or organization acts in many different roles. Roles represent:

- Author; a person creating content and using Zemanta suggestions
- CMS creator; person or organization providing a Content management software that has Zemanta service and user experience integrated as part of the authoring process
- Platform owner; a person or organization owning the specific hosting platform that runs the Content management software supporting Zemanta service and user experience
- Reader; a person or organization consuming the content
- Zemanta; a service provider during content creation process

If a single programmer both writes his own CMS, runs it and creates content with it, he is fulfilling three different roles. This is a common usage pattern for enthusiasts experimenting with Zemanta. When commercial entities are involved there are cases where CMS creator and platform provider is the same organization (some big blogging platforms) and cases where publishing platform owner is a different company from CMS creator who sold or open sourced the CMS software which publishing platform owner is using.

In some cases CMS creator is split into two different entities. This happens when existing open or closed source CMS is extended with Zemanta experience by an integrator. In such a situation we regard integrator as CMS creator for the scope of this document.

Reader can be wide internet public or when Intranet is concerned it can be just the limited public that has access to a specific document.

## Information for authoring application developers

When you step into Zemanta ecosystem you get full access to Zemanta developer portal where you can find documentation about API and this document. As long as you are testing the API you should use the API key provided to you.

**When you want to deploy your application for your end users, each of them should get their own Zemanta API key**, so he can control preferences and have personalized suggestions. Keep in mind that each single API key allows just 1000 requests per day.

Please contact Zemanta at support@zemanta.com for information on how to implement automatic assignment of API keys to each separate author using your software. Also contact us if you need larger quota of requests per day.

## Information for publishing platform owners

If you want to enable your users to experience Zemanta you need to install a special plug-in onto your CMS platform and enable it for all users. Plug-ins are currently available for W**ordpress (2.x, 3.x) and Movable type (4.x, 5.x), Drupal and Joomla**. If you are not using any of those, extending support to your platform of choice is part of possible commercial relationship.

There are other options how you can enable your users to experience Zemanta on one-by-one basis. You can recommend to your users to install Zemanta browser extension. Currently Zemanta browsers extension is available for Firefox, Internet Explorer, Chrome. Also available is a Windows Live Writer plug-in.

**Each of the authors using your platform should get their own Zemanta API key, so they can control preferences and have personalized suggestions.** Please contact Zemanta at support@zemanta.com for information on how to implement automatic assignment of API keys to each separate author using your software.

## Information for other integrators

There are many use cases for use of Zemanta suggestions outside of “classical” content authoring process. We cannot imagine all of them and we are eager to hear about them. Please contact us at support@zemanta.com with information about what you created and what you need from Zemanta to make your software even better.

We are very flexible and would like your input on the Zemanta ecosystem, on Zemanta API and the quality of suggestions.

# Suggestions

As already explained Zemanta service returns different types of suggestions, each of them is discussed here in detail.

## *Images*

A picture is worth thousand words. It makes it easy for the reader to associate specific page with his imagination, it can explain the topic much more efficiently or it can simply decorate the article it accompanies.

### Sources of images

There are multiple sources which Zemanta uses as a basis for image suggestions. Wikipedia and Flickr are probably most comprehensive while images from stock image providers are of higher aesthetical quality.

Zemanta's image collection from Wikipedia includes 5 million images. While Flickr provides over two billion photos, unfortunately  not all of them can be used by authors using Zemanta service. Most of images on Flickr have license forbidding reuse, so that leaves around 160 million images as possible suggestions. **Because Zemanta uses Flickr API, it can not take advantage of Zemanta's internal representation of concepts. This means less topically accurate images are suggested.** The same goes for other external image providers. Additional issue is proliferation of intentional and unintentional mistagging of Flickr images. This can sometimes cause completely unrelated images being suggested. [author of this document notes that semantic tagging of Flickr images would probably be the greatest contribution Yahoo could provide for progress of Semantic web]

It is important to know that **images indexed in Zemanta's database and even those offered as suggestions via external sources are not guaranteed to exist** at specified URLs. Therefore authoring application presenting the images must check if those resources exist and in case they do not fill the space with suggested images further down the list.

### Licenses

When suggesting images Zemanta tries to filter out those that would undoubtedly need image owner's approval. Most of the time that means suggesting images under CreativeCommons or similar licenses, public domain images and images that could be used as fair use. Zemanta also suggests images from stock photo providers when they allow their low resolution images to be used.

Zemanta provides necessary license information which author can interpret and decide whether intended use of the image is allowed under the terms of the license. It is important for both developers using the API and authors using the suggestions to know that **it is impossible for computer to reliably determine if usage of image in specific case is allowed**. Therefore maximal effort of informing the author is and should be done and the author is responsible for making the final decision.

API conveys the license along each image inside an attribute named “license”. This attribute is HTML formated and should be presented directly to the author (as Terms of service demand). It sometimes includes a link to location where author can get further information. Zemanta can not guarantee that supplied image licenses are correct since they all come from outside sources.

## Formats and dimensions

Suggested images are always raster images. This means they are either in JPG, PNG or GIF format. This happens even when original image is vector based (SVG), since Zemanta suggests links to converted image.

Each image suggestion consists of three URL attributes, one for small, medium and large version of the image.

Zemanta cannot fully control the sizes of images available from different repositories. We strive to respond with the small image width near 75 pixels, middle sized picture width near 200 pixels and large size image to be the highest resolution of the image available under the given license. Attributes “height” and “width” contain dimensions of the image available at the URL of the original wlarge version.

## Description

Each image suggestion includes “description” attribute. This is textual description of what image represents. Description may be inaccurate in some cases, especially with third party image sources. Further information about the image can be found at URL in the attribute “source_url”. This url points to the page inside the image repository that has further human-readable information about the image.

## Inclusion guidelines

Each image suggestion includes attribute “attribution” which describes source and author of the image when those are available. Data inside the attribute is HTML formated and where possible it includes link to the page of the author, image or source.

**Application that provides Zemanta image suggestions to authors has to automatically include that attribution along the image when it is inserted**. If it does not do so, it must provide other ways to allow authors to easily comply with the image license conditions and following precisely Terms of service.

Zemanta acknowledges that application writers cannot control what happens after attribution is inserted into the editing area. Authors' responsibilities cannot be controlled by software. Authors themselves are bound by Terms of service and licenses of specific images.

From technical viewpoint when author decides to use specific image suggestion, application should try to retrieve the inserted image and have it served from author's content serving platform. However in certain situations it is not possible to do so.

Zemanta also strongly suggest to place link to the “source_url” on the whole image as an anchor.

**Images added via Zemanta should have an upper element enclosing both image and its attribution. Class of that enclosing element should be “zemanta-img”**. This enables Zemanta to behave correctly when already enriched text is submitted to the service and using that information as feedback (in the future).

It is suggested that authoring applications make a copy of image and store it on authoring application server before publishing the image. This will guarantee that the image removal or change at the primary source (such as Wikipedia or Wikimedia commons) will not affect presentation of the published content.

### *Related articles*

Letting the reader know aobut related articles accomplishes many things for a publisher. Firstly reader experience is better since readers are able to dig deeper into their topic of interest. Happy readers are returning readers. Secondly important effect of actively linking to sub-web of common interest is building loosely-knit communities around specific topics. Thirdly a lot of blogs support so called "trackbacks". By linking to those blogs, author could automatically get a back-link to his page. While linking to other sites could be seen as counterintuitive to those monetizing through advertising and benefiting from keeping the user around for as long as possible, it is an established fact that on long term sites that are "good net citizens" receive more visitors.

Zemanta allows at the same time automatic search for related articles and full control of the author over article inclusion. From user feedback Zemanta has come to know that many authors only read suggested related articles by themselves and use gained information to write better content while not explicitly linking to sources. This is acceptable use of Zemanta service. Because of this usage pattern and because of the need of authors to check what articles they are linking to authoring applications should allow for easy visiting of suggested articles before they are included into main body of content.

Article is probably not the perfect term for describing the piece of content Zemanta suggests a link to, albeit the suggestion usually has the form of the article. It could also take different form such as a tweet, a photo album, a photo blog, famous quote or similar. However articles present the wast majority of these suggestions.

## Sources of articles

Zemanta aggregates articles from many different internet sources. Those include both mainstream news sites and wider blogosphere with emphasis on highly-regarded blogs. Zemanta tries to include articles from multiple domains, multiple world regions and multiple world-views. If you believe there are topic areas that are covered unsatisfactory, please let Zemanta know at support@zemanta.com.

In January 2011 around 50000 sources were being followed with new ones being added each day. Every source is manually vetted in order to prevent spam from creeping in. Zemanta also adds sources because of commercial relationships with the publishers.

Estimated time for a new article to be included in the index from which article suggestions are made is from few minutes to maximum of two hours. We try to have index as fresh as possible. Currently (January 2011) index consists of all articles seen in last three months, but we are trying hard to expand the timeframe. **Old articles are not being purged, but there is bias toward more recent articles in suggestion algorithm.**

## Licenses

Zemanta only suggests link to the article, not its content. Therefore author or publisher has no need to comply with any licensing conditions.

Basic metadata about the article is provided by Zemanta. "title" is a string representing the title of the article, "url" points to the article page (Zemanta tries to resolve url redirects, but that might not always

be possible). “published_datetime” represents a date and time when article was published. If this information was not available first time article was aggregated by Zemanta, harvest time is used. Datetime might be in future in some exceptional cases due to wrong data provided by publishers, usually because of wrong usage the timezone. Larger deviations of published time are heuristically adjusted by Zemanta. “text_highlight” and “title_highlight” represent search snippet which establishes a logical connection between source article and suggested article.

### Inclusion guidelines

Related articles should be added to the main body of text in a way that makes it clear that they represent a connection to a larger body of knowledge about the topic. Zemanta browser plug-in achieves that by putting the articles into a list with a title saying “Related articles”.

**Related articles added via Zemanta should have class defined as “zemanta-article”.** If you are using outer element such as fieldset to define a group of related articles, use “zemanta-related” as its class. This enables Zemanta to behave correctly when already enriched text is submitted to the service and using that information as feedback (in the future).

## *In-text links*

In-text links represent links inside the main body of text that lead reader to the information about very specific concepts and topics that were mentioned. Zemanta currently provides links to “entities”, but not to other blogs as in-text links.. Word “markup” is used to refer to in-text links inside the API and API documentation due to more general nature of the term markup.

### Sources of in-text links

Zemanta uses knowledge databases to establish connection between mentions of specific concepts or topics and the text involved. Such databases are Wikipedia, IMDB, MusicBrainz, Amazon book listings and similar. Understanding of text is used to find out whether specific phrase should be linked to specific page. In one context Zemanta may decide a connection is desirable in the other the same link might not be suggested. Context is also used to disambiguate between different possibilities when linking (for example Apollo the space program and Apollo the Greek god).

Links to some information providers may include affiliate ID. Affiliate ID information is part of preferences of each user.

### Licenses

Zemanta does not suggest any content of the pages pointed to by links. Therefore there are no need to comply with any licensing conditions for the author and publisher.

### Link description, types and metadata

Links are not anchored to specific location in the text, but instead to substrings of the text. This is done because original text might change before author decides to apply a link and it would be extremely hard for authoring software to correctly do the necessary bookkeeping. That's why “anchor” to which a link

should be attached is provided as an attribute. For each anchor it might be sensible to link to multiple destinations and just one should be chosen by the author. Possible destination links for specific anchor text are stated in “target” list of links. Each target has a “url”, “title” and “type” attribute. Some of the types are:

- wikipedia
- amazon
- imdb
- youtube
- homepage
- geolocation

Types are self explaining. New types might be added by Zemanta and authoring software must gracefully handle unknown types. New attributes might be available for precise information about specific types.

The engine does not suggest links to anchors that are already linked with <a href...> HTML statement when content is submitted.

### Inclusion guidelines

**In-text links added via Zemanta should have their class defined as “zem_slink”**. This enables Zemanta to behave correctly when already enriched text is submitted to the service including using that information as feedback (in the future).

When Zemanta finds a possible link and the anchor of the link is already partly or fully marked up, Zemanta will not return that link as a suggestion. For example “we are in Washington D.C.”, will prevent Zemanta suggesting correct link to Washington D.C. page. Exception to this rule are links with class “zem_slink”.

If anchor to be linked is entirely enclosed in the bold tag, link will be returned. Zemanta will also not return new link which would have anchor overlapping with already existing link. **If you want to get maximum accuracy of in-text links for further processing, strip input text of HTML tags before sending it to Zemanta service.**

### GUI guidelines

Authoring application creators should pay close attention to enable simple way of unlinking previously selected text, Zemanta suggests that GUI element that created the link should change state and on the second click unlink that same link.

## *Tags*

Tag is a relevant keyword or term associated with specific content. Labeling by keywords has long been used in scientific publications. Its web comeback happened when web users and developers realized tags are a very efficient method of attaching metadata to the infromation. While lacking formal

structure tags can provide valuable navigational enhancements and make it easy for web search engines to comprehend content more fully. Lack of formal structure also made it very easy for everyone to tag their content and even content published by others (like del.icio.us does).

Zemanta makes tagging even simpler. User does not need to make up tags, instead he is offered a number of possible tags and he just needs to choose appropriate ones (preferably via single click).

From information processing viewpoint tags are metadata that accompanies main body of information. They are used by CMS to provide navigational facilities and by search engines as informal clues on what content is about.

## Sources of tags

While some other services only try to find the most overrepresented rare words or proper names in the text, Zemanta goes deeper when processing content. **Zemanta offers both tags based on words and phrases that can be found inside author's text and also those that are only topics that could represent the content as a whole, but are not explicitly mentioned.** It goes even further and tries to find very concrete items and concepts that are related to what is being said, but are only connected through a third piece of information. Therefore author can expect topics, names and concepts as tags.

## Licenses

Tags are pieces of content that are generally not regarded as creative work and therefore not protected under copyright. However authors should be aware that brand names can be offered as tags.

## Tag delivery

Zemanta currently does not provide any additional information about reasoning why specific tag was suggested. There is a confidence factor, but it has to be noted that it might be unreliable.

Zemanta's tags can include whitespaces, but will never include commas. Inside API and API documentation “keyword” is a name used for tags, since that is more general and understandable outside web domain.

## Inclusion guidelines

Authoring application should store selected tags as metadata separately from main body of content. This is the standard practice in content management systems. Zemanta suggests that when tags are rendered they should be marked up semantically so advanced reading applications can make use of them (look at Technorati tags specification).

## GUI guidelines

Authoring applications should allow for inclusion and exclusion of tags with a single click. When there is a longer text field in which tags can be entered and edited authoring application should provide each suggested tag as clickable button.

## Signature

Each call to Zemanta suggest method also returns an attribute called “signature”. This has to be automatically added to each published article to fulfill Zemanta Terms of service. Authors are then allowed to manually delete the signature from each article.

Signature is available in different forms (which can be chosen via preferences discussed in next capture).

Main forms of signature are:

- “Zemified” icon
- Hidden pixie

### *Zemified icon*

Zemified icon simply reveals to the reader that author has used Zemanta during the process of authoring, it represents a way of giving credit to Zemanta's hard work.

### *Reblog*

Reblog functionality was deprecated in summer 2010.

### *Hidden pixie*

Both Zemified icon and Reblog allow Zemanta to detect which articles were written with its help and eventually include those blogs into its aggregation index.

## User (author's) preferences

Zemanta allows authors to set some preferences regarding how the service works. Some preferences affect suggestion algorithms while other affect just the user interface.

Preferences are currently tied to each separate API key. When authoring application calls function suggest.preferences one of the attributes that are part of the answer is “config_url”, it points to Zemanta's server where user can set his specific settings. Authoring application should give the user ability to open the page at that address in new window.

Currently user can set following options:

- alignment of the image (left or right)
- Amazon ID and if it is set at all
- styled HTML or unstyled XHTML insertion of suggestions
- type of signature (discussed in previous chapter)

Those preferences that should affect authoring applications are returned as attributes in responsof a preferences API call. Please use them accordingly. If there are any doubts, don't hesitate to contact Zemanta!

# Web service introduction

Zemanta web service is a classical type of web service. Using standard HTTP GET or POST protocol client sends a request via port 80 to api.zemanta.com and gets response encoded in JSON, XML or RDF/XML format.

Client identifies itself with API key. API key is a string that uniquely identifies specific instance of the application that is using the service. For example Firefox extension stores API key in the extensions' permanent configuration and uses that API key on every request to Zemanta service no matter which blogging platform the user visits (the same is true for other browser extensions).

Zemanta web service includes limitations on the number of requests per day and number of requests per second. First one depends on type of the account you have, default developer accounts allow for 1000 posts per day and 1 post per second. When you go beyond these limits you will get back the appropriate error message.

## *POST request*

For long texts it is not possible to encode them into a standard GET request due to limitations in URL length of some web servers and proxies. **In those cases users can use POST HTTP requests.**

Due to security measures browsers in general prevent cross-site POST requests (requests to sites different than one from which the page is loaded from). However Zemanta correctly sets allow-origin headers and therefore all modern browsers allow javascript to easily use POST requests to Zemanta API.

Zemanta also provides helpers inside the API which enable so called “window-name json” return format and thus POST requests can be used even in older browsers.

# Obtaining an API key

There are two ways of obtaining API keys. One is meant for developers and the other is available for applications to automatically assign keys to their users (authors).

## *Developer API key*

Developers and testers initially get only one key for their own usage. As a developer you first have to create an account at Zemanta developer portal (at http://developer.zemanta.com) and then continue to create your own API key. You have to enter

## *API keys for content management systems and platforms*

Zemanta provides a way to assign separate key to each end user (author) of Zemanta service when

Zemanta is integrated into third party CMS or content authoring platform. Please contact Zemanta at info@zemanta.com about the details.

# General API description

API is officially documented on Zemanta Developers' portal http://developer.zemanta.com/docs

Please consult documents there for official and formal descriptions of the API. This document is mainly concerned about giving you all the info you need on top of formal API description.

# Examples of API usage

Examples of how to use Zemanta API in different programming languages. Wherever you see string YOUR_API_KEY you should replace it with your own API key.

## *Python*

### XML encoding example

Following example is a simple call to the suggest function

```
import urllib

gateway = 'http://api.zemanta.com/services/rest/0.0/'
args = {'method': 'zemanta.suggest',
        'api_key': 'YOUR_API_KEY',
        'text': 'Cozy lummox gives smart squid who asks for job pen.',
        'format': 'xml'}
args_enc = urllib.urlencode(args)
print urllib.urlopen(gateway, args_enc).read()
```

### JSON encoding example

Here is the simplest way to use the API in python using simple json encoding. Simplejson library is needed.

```
import urllib
import simplejson
gateway = 'http://api.zemanta.com/services/rest/0.0/'
args = {'method': 'zemanta.suggest',
        'api_key': 'YOUR_API_KEY',
        'text': 'Cozy lummox gives smart squid who asks for job pen.',
        'format': 'json'}
args_enc = urllib.urlencode(args)
response_raw = urllib.urlopen(gateway, args_enc).read()
response = simplejson.loads(response_raw)

print response
```

```
print "Suggested links:"
for link in response['markup']['links']:
        print "    \"%s\" -->" % (link['anchor'])
        for target in link['target']:
                print "        %s" % (target['url'])

print "Suggested related:"
for article in response['articles']:
        print "    %s\n        %s" % (article['title'], article['url'])

print "Suggested images:"
for image in response['images']:
        print "    %s\n        %s" % (image['description'], image['url_l'])

print "Suggested keywords:"
print "\t",
for keyword in response['keywords']:
        print "%s," % (keyword['name']),
```

### *PHP*

This code example comes from Hubert Moreau. He posted it to Zemanta developer forums.

by hmoreau:

```
    <?php
    /* This are the vars you may need to modify */
    /* Some may be placed in conf files */
    /* Some may be generated by your application */
    $url = 'http://api.zemanta.com/services/rest/0.0/'; //Should be in a conf
    file
    $format = 'xml'; // May depend of your application context
    $text = "Place here the text you want to be parsed by Zementa"; // May
    depend of your application context
    $key = "your_zementa_api_key"; //Should be in a conf file
    $method = "zemanta.suggest"; // May depend of your application context

    /* It is esayer to deal with arrays */
    $args = array(
    'method'=> $method,
    'api_key'=> $key,
    'text'=> $text,
    'format'=> $format
    );

    /* Here we build the data we want to POST to Zementa */
    $data = "";
    foreach($args as $key=>$value)
    {
    $data .= ($data != "")?"&":"";
    $data .= urlencode($key)."=".urlencode($value);
    }

    /* Here we build the POST request */
    $params = array('http' => array(
    'method' => 'POST',
```

```
    'Content-type'=> 'application/x-www-form-urlencoded',
    'Content-length' =>strlen( $data ),
    'content' => $data
    ));

    /* Here we send the post request */
    $ctx = stream_context_create($params); // We build the POST context of the
request
    $fp = @fopen($url, 'rb', false, $ctx); // We open a stream and send the
request
    if ($fp)
    {
    /* Finaly, herewe get the response of Zementa */
    $response = @stream_get_contents($fp);
    if ($response === false)
    {
    $response = "Problem reading data from ".$url.", ".$php_errormsg;
    }
    fclose($fp); // We close the stream
    }
    else
    {
    $response = "Problem reading data from ".$url.", ".$php_errormsg;
    }
    ?>
    Tags

        * Php
        * sample
        * code

    #

    hmoreau — 6 May 2008 @ 2:11 amEdit comment

    The same using the curl library
    ----- sample_curl.php -----
    <?php
    /* This are the vars you may need to modify */
    /* Some may be placed in conf files */
    /* Some may be generated by your application */
    $url = 'http://api.zemanta.com/services/rest/0.0/'; //Should be in a conf
    file
    $format = 'xml'; // May depend of your application context
    $text = "Place here the text you want to be parsed by Zementa"; // May
    depend of your application context
    $key = "your_zementa_api_key"; //Should be in a conf file
    $method = "zemanta.suggest"; // May depend of your application context

    /* It is easier to deal with arrays */
    $args = array(
    'method'=> $method,
    'api_key'=> $key,
    'text'=> $text,
    'format'=> $format
```

```
    );

    /* Here we build the data we want to POST to Zementa */
    $data = "";
    foreach($args as $key=>$value)
    {
    $data .= ($data != "")?"&":"";
    $data .= urlencode($key)."=".urlencode($value);
    }


    /* Initialisation of curl */
    $ch = curl_init();
    /* Setup of the url*/
    curl_setopt($ch, CURLOPT_URL, $url);
    /* We want a post request */
    curl_setopt($ch, CURLOPT_POST, 1);
    /* Here we give to curl the data we want to send to Zementa*/
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
    /* We setup the response method of curl */
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    /* Execute curl and fetch the result */
    $response = curl_exec ($ch);
    /* Close curl connection */
    curl_close ($ch);
    ?>
```

### *Perl*

There is a Net::Zemanta module available from CPAN which is a thin Perl wrapper around the API methods (http://search.cpan.org/~avian/Net-Zemanta). It provides a simple object-oriented interface and is the preferred way of calling Zemanta from Perl code.

The following is a minimal example using the module:

```
use Net::Zemanta::Suggest;
use Data::Dumper;

my $zemanta = Net::Zemanta::Suggest->new(APIKEY => 'YOUR_API_KEY');

my $suggestions = $zemanta->suggest("Cozy lummox gives smart squid who asks for
job pen.");

print Dumper($suggestions);
```

If you do not want to introduce the Net::Zemanta dependecy you can craft your own code for calling the API. Bellow is a minimal example (without any error checking) which you can use as a template:

```
use LWP::UserAgent;
use HTTP::Request::Common;
use JSON;
use Data::Dumper;
```

```
my $gateway = 'http://api.zemanta.com/services/rest/0.0/';

my $args = {      method => 'zemanta.suggest',
                  api_key => 'YOUR_API_KEY',
                  text => 'Cozy lummox gives smart squid who asks for job pen.',
                  format => 'json' };

my $ua = LWP::UserAgent->new;
my $response = $ua->request(POST $gateway, $args);
my $result = from_json($response->content);

print Dumper($result);
```

### *C#*

This code is courtesy of Tom Altman, it has been submitted to Zemanta developer forum by him. Thank you very much.

By Tom Altman:

Please search for "<yourapihere>" and replace with your api! Good luck.

This is the "_textZ.aspx: page:

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="_textZ.aspx.cs"
    Inherits="_textZ" %>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
    <title>TEST Page</title>
    </head>
    <body>
    <form id="frm" runat="server">

    <table width="800" align="center" cellpadding="0" cellspacing="0"
    border="1">
    <tr>
    <td align="left" valign="top" width="66%"><asp:Label ID="lblContent"
    runat="server" /></td>
    <td align="left" valign="top" width="34%">
    <asp:Label ID="lblZem" runat="server" />
    <asp:label ID="CloudMarkup" runat="server" />
    </td>
    </tr>
    </table>

    </form>
    </body>
    </html>

    ok - now the _textZ.aspx.cs:
    using System;
```

```
using System.Collections;
using System.Configuration;
using System.Data;
using System.IO;
using System.Net;
using System.Text;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml;

using Newtonsoft.Json.Converters;
using Newtonsoft.Json.Utilities;

public partial class _textZ : System.Web.UI.Page
{
string str = "I have a dream that one day this nation will rise up and live
out the true meaning of its creed: We hold these truths to be self-evident,
that all men are created equal. I have a dream that one day on the red hills
of Georgia, the sons of former slaves and the sons of former slave owners
will be able to sit down together at the table of brotherhood. I have a
dream that one day even the state of Mississippi, a state sweltering with
the heat of injustice, sweltering with the heat of oppression, will be
transformed into an oasis of freedom and justice. I have a dream that my
four little children will one day live in a nation where they will not be
judged by the color of their skin but by the content of their character. I
have a dream today!";

protected void Page_Load(object sender, EventArgs e)
{
lblContent.Text = str;
//string json = getZemanta("zemanta.suggest",str,"json");
string xml = getZemanta("zemanta.suggest",str,"xml");

XmlDocument xmldoc = new XmlDocument();
xmldoc.LoadXml(xml);

XmlNodeList xmlnode = xmldoc.GetElementsByTagName("article");

lblZem.Text = lblZem.Text + "<ul>";
for (int i = 0;i < xmlnode.Count;i++) {
//XmlAttributeCollection xmlattrc = xmlnode[i].Attributes;
////XML Attribute Name and Value returned
////Example: <Book id = "001">

//lblZem.Text = lblZem.Text + xmlattrc[0].Name;
//lblZem.Text = lblZem.Text + ":\t" + xmlattrc[0].Value;

lblZem.Text = lblZem.Text + "<li><a href=\"" + xmlnode[i]["url"].InnerText +
"\">" + xmlnode[i]["title"].InnerText + "</a></li><br>";
}
lblZem.Text = lblZem.Text + "</ul>";
```

```
    populateCloud();
    }

    public static string getZemanta(string whichMethod, string whatContent,
    string whatFormat)
    {
    Uri address = new Uri("http://api.zemanta.com/services/rest/0.0/");

    // we need to create the web request
    HttpWebRequest wreq = WebRequest.Create(address) as HttpWebRequest;

    // we want to post the data - so set that here.
    wreq.Method = "POST";
    wreq.ContentType = "application/x-www-form-urlencoded";

    // build string with data for REST
    StringBuilder d = new StringBuilder();
    d.Append("method=" + HttpUtility.UrlEncode(whichMethod));
    d.Append("&api_key=" + HttpUtility.UrlEncode("<yourapihere>"));
    d.Append("&text=" + HttpUtility.UrlEncode(whatContent));
    d.Append("&format=" + HttpUtility.UrlEncode(whatFormat));

    // break it down to a byte array
    byte[] bd = UTF8Encoding.UTF8.GetBytes(d.ToString());

    // set length & write
    wreq.ContentLength = bd.Length;
    using (Stream ps = wreq.GetRequestStream()) { ps.Write(bd,0,bd.Length); }

    // response?
    using (HttpWebResponse wres = wreq.GetResponse() as HttpWebResponse) {
    // capture the response
    StreamReader r = new StreamReader(wres.GetResponseStream());

    // return the results
    return r.ReadToEnd();
    }
    }
```

### *Java*

Example comes from Thomas Francart who posted it in Zemanta developer forums.

***** File ZemantaWrapper.java *****

```
package com.mondeca.test.zemanta;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
```

```
import java.io.UnsupportedEncodingException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.xml.sax.SAXException;

/**
 *
 * @author thomas.francart@mondeca.com
 */
public class ZemantaWrapper {

      private static String ZEMANTA_API_URL =
"http://api.zemanta.com/services/rest/0.0/";
      // your API key
      private String key;

      public ZemantaWrapper(String key) {
            super();
            this.key = key;
      }

      /**
       * Calls the suggest API on the given input text, with XML
       * format. Parse result and return it as DOM.
       * @param text text to be processed.
       * @return
       * @throws ZemantaException
       */
      public Document zemanta_suggest(String text)
      throws ZemantaException {
            try {
                  // build parameters
                  String data = URLEncoder.encode("method", "UTF-8") + "=" +
URLEncoder.encode("zemanta.suggest", "UTF-8");
                  data += "&" + URLEncoder.encode("api_key", "UTF-8") + "=" +
URLEncoder.encode(this.key, "UTF-8");
                  data += "&" + URLEncoder.encode("text", "UTF-8") + "=" +
URLEncoder.encode(text, "UTF-8");
                  data += "&" + URLEncoder.encode("format", "UTF-8") + "=" +
URLEncoder.encode("xml", "UTF-8");

                  return doZemantaCall(data);

            } catch (UnsupportedEncodingException e) {
                  // you must be kidding ?!
                  e.printStackTrace();
                  return null;
            }
```

```
	}

	/**
	 * Calls the suggest feedback API on the given input text, with XML
	 * format. Parse result and return it as DOM.
	 * @param text text to be processed.
	 * @return
	 * @throws ZemantaException
	 */
	public Document zemanta_suggest_feedback(String text)
	throws ZemantaException {
		try {
			// build parameters
			String data = URLEncoder.encode("method", "UTF-8") + "=" +
URLEncoder.encode("zemanta.suggest.feedback", "UTF-8");
			data += "&" + URLEncoder.encode("api_key", "UTF-8") + "=" +
URLEncoder.encode(this.key, "UTF-8");
			data += "&" + URLEncoder.encode("text", "UTF-8") + "=" +
URLEncoder.encode(text, "UTF-8");
			data += "&" + URLEncoder.encode("format", "UTF-8") + "=" +
URLEncoder.encode("xml", "UTF-8");

			return doZemantaCall(data);

		} catch (UnsupportedEncodingException e) {
			// you must be kidding ?!
			e.printStackTrace();
			return null;
		}
	}

	/**
	 * Calls the preferences API, with XML
	 * format. Parse result and return it as DOM.
	 * @param text text to be processed.
	 * @return
	 * @throws ZemantaException
	 */
	public Document zemanta_preferences(String format)
	throws ZemantaException {
		try {
			// build parameters
			String data = URLEncoder.encode("method", "UTF-8") + "=" +
URLEncoder.encode("zemanta.preferences", "UTF-8");
			data += "&" + URLEncoder.encode("api_key", "UTF-8") + "=" +
URLEncoder.encode(this.key, "UTF-8");
			data += "&" + URLEncoder.encode("format", "UTF-8") + "=" +
URLEncoder.encode(format, "UTF-8");

			return doZemantaCall(data);

		} catch (UnsupportedEncodingException e) {
			// you must be kidding ?!
			e.printStackTrace();
			return null;
```

```
            }
      }

      private Document doZemantaCall(String data)
      throws ZemantaException {
            Document d = null;
            try {
                  URL url = new URL(ZEMANTA_API_URL);
                  String result = readURL(url, data, "UTF-8");
                  d = parse(new ByteArrayInputStream(result.getBytes("UTF-8")));
            } catch (UnsupportedEncodingException e) {
                  // you must be kidding ?!
                  e.printStackTrace();
            } catch (IOException e) {
                  throw new ZemantaException("Error while calling service
result",e);
            } catch (SAXException e) {
                  throw new ZemantaException("Error while parsing service
result",e);
            } catch (ParserConfigurationException e) {
                  throw new ZemantaException("Error while parsing service
result",e);
            }
            return d;
      }

      /**
       * Utility method. Parses given input stream and turn that into a DOM.
       * @param input
       * @return
       * @throws SAXException
       * @throws IOException
       * @throws ParserConfigurationException
       */
      private static Document parse(InputStream input)
      throws SAXException, IOException, ParserConfigurationException {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setNamespaceAware(true);
            DocumentBuilder domFactory = dbf.newDocumentBuilder();
            return domFactory.parse(input);
      }

      /**
       * Utility method. Calls the given URL with the given POST data.
       * @param url
       * @param data
       * @param charset
       * @return
       * @throws java.io.IOException
       */
    private static String readURL(URL url, String data, String charset) throws
java.io.IOException {
        StringBuffer buf = new StringBuffer(10000);
        URLConnection c = url.openConnection();
        c.setRequestProperty("User-Agent","");
```

```
        c.setDoOutput(true);
        OutputStreamWriter wr = new OutputStreamWriter(c.getOutputStream());
        wr.write(data);
        wr.flush();
        InputStreamReader sourceContent = new
InputStreamReader(c.getInputStream(),charset);

        BufferedReader in = new BufferedReader(sourceContent);
        buf.setLength(0);
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            buf.append(inputLine+"\n");
        }
        in.close();
        return buf.toString();
    }

}
```

***** File ZemantaTest.java *****

```
package com.mondeca.test.zemanta;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StringWriter;
import java.io.UnsupportedEncodingException;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class ZemantaTest {

      public void test()
      throws Exception {
            System.out.println("Testing zemanta service...");
            // change with your own API key
            ZemantaWrapper zemanta = new ZemantaWrapper("1234567890AZERTYUIOP");
            Document d = zemanta.zemanta_suggest(readFileContent(new
File("C:/inputzemanta.txt"), "UTF-8"));

            // pretty print result
            System.out.println(toString(d, "UTF-8"));
      }
```

```
	/**
	 * Pretty prints a DOM.
	 * @param node
	 * @param encoding
	 * @return
	 * @throws TransformerException
	 */
	public static String toString(Node node, String encoding)
	throws TransformerException {
		StringWriter writer = new StringWriter();
		Transformer tr = TransformerFactory.newInstance().newTransformer();
		tr.setOutputProperty(OutputKeys.INDENT, "yes");
		tr.setOutputProperty(OutputKeys.ENCODING, encoding);
		tr.transform(new DOMSource(node), new StreamResult(writer));
		return writer.toString();
	}

	/**
	 * Read content of a File into a String.
	 * @param f
	 * @param charset
	 * @return
	 * @throws IOException
	 */
	public String readFileContent(File f, String charset)
	throws IOException {
		// fetch content of the stream
		StringBuffer buf = null;
		try {
			buf = new StringBuffer(10000);
			InputStreamReader sourceContent = new InputStreamReader(new
FileInputStream(f),charset);

			BufferedReader in = new BufferedReader(sourceContent);
			buf.setLength(0);
			String inputLine;
			while ((inputLine = in.readLine()) != null) {
				buf.append(inputLine+"\n");
			}
			in.close();
		} catch (UnsupportedEncodingException e) {
			// !!! encoding not supported ?!
			e.printStackTrace();
		}
		return buf.toString();
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		try {
			ZemantaTest test = new ZemantaTest();
			test.test();
```

```
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }

}


***** File ZemantaException.java *****

package com.mondeca.test.zemanta;

/**
 *
 * @author thomas.francart@mondeca.com
 *
 */
public class ZemantaException extends Exception {

      public ZemantaException(String arg0, Throwable arg1) {
            super(arg0, arg1);
      }

      public ZemantaException(String arg0) {
            super(arg0);
      }

      public ZemantaException(Throwable arg0) {
            super(arg0);
      }

}
```

## Dictonary:

Zemanta service

Author

Authoring application

Authoring software (same as authoring application)

Reader

Developer

Platform

CMS

Flickr