Implementation Guide

Document Version 1.1

Freetag Version 0.230

Table of Contents

Introduction

This manual is written for application developers with familiarity and experience working with PHP and MySQL. Experience with ADOdb for PHP is useful for playing with hacking Freetag, but is not necessary.

Tagging Databases in the Wild

Freetag was created in order to allow you to make **your** application behave the exact way that you want it. In order to achieve that goal, many different approaches to tagging were considered, from an end-user perspective, which you can configure by passing special constructors to the freetag class (in fact, it also supports connecting to different Freetag installs by switching your table prefix). Before diving into this documentation, let's review some of the big differences in implementations out there currently.

Tag Input Quoting and Separators

There are a couple different schools of thought about how to process tags coming in through a text box. The most common situation is to parse tags similar to Google searches – that is, every word as a distinct tag, ignoring all punctuation except for double-quotes. Double quotes actually signify phrases. Since this is the predominant format, this is the default situation in Freetag, but it can be tweaked somewhat through modifying normalization parameters, as described next.

Tag Normalization (and "Raw" Tags)

"Normalization," when discussing tags, refers to any operations that modify or otherwise filter user input (farther than splitting into phrases), such as: lowercasing, eliminating spaces, eliminating punctuation, etc. A "Raw" tag is an unmodified version of a tag, as the original author input it. Freetag normalizes all tags by default, by lowercasing and eliminating all non-alphanumeric characters. It goes a step further (similar to Flickr) and also stores the original "raw" tag uniquely, and stores the normalized version alongside it.

Additionally, when displaying tags, it becomes the implementor's choice whether to show the "raw" or "normalized" tags in the user interface. Sites similar to Flickr show the "raw" tag to the user who tagged it, thus maintaining the user's personal style of markup, while using the normalized tag in query, search, and display to others.

The advantages of normalizing tags like this are as follows:

1) URL's are easier to build off of alphanumeric characters. You can build http://site.com/tag/tagname/ clean URL's easier with normalized tags.

2) Even though people might tag "NY", "N.Y.", and "ny" in these different formats, normalization means that anyone searching for one of those will come up with any objects with a normalized tag matching "ny." This is a moderate advantage when querying your tag db.

3) Users can still use their own "raw" format tags to organize their content when logged in. Many users get upset when they see their own tags normalized in the tagging interface.

Either way, you can choose which path to follow through the "normalize_tags" setting, and even control the characters to filter on with the "normalized_valid_chars" string.

Tag Edit and Delete

There are two major styles of allowing users to edit and delete their tags. In the del.icio.us style, the user always interacts with their tags on an object through a single text field. This approach is easily done (albeit a bit inefficiently) through a combo of Freetag's delete_all_object_tags_for_user() and tag_object() functions, called upon submission of the edit text box. In Flickr's style, each tag must be deleted individually, yet new tags can be entered through a text box. You'd use Freetag's delete_object_tag() with tag_object() for this situation.

Multiuser Tags on Objects

One trickier topic (especially in combination of allowing "raw" tag manipulation) is whether two users can tag an object with the same tag. This completely depends on your design choices, and can be supported either way with Freetag's "block_multiuser_tag_on_object" constructor parameter. It's set to block multiuser tags on objects by default.

Tagging with Freetag

As you can see, Freetag is quite flexible in allowing you to build your own tagging system however you prefer. Plus, it's open source, so you can feel free to modify it and customize it however you'd like.

If you do see something that I've missed in configuration options, drop me a line and I'll decide whether it makes sense to include in the distribution. Most suggestions have been included so far!

Sample Application

The manual is organized around a theoretical database application that stores information about cars. This application stores information about an individual car in MySQL database records, as follows:

```
mysql> describe cars;
+-------+------------------+------+-----+---------+----------------+
| Field | Type             | Null | Key | Default | Extra          |
+-------+------------------+------+-----+---------+----------------+
| id    | int(10) unsigned |      | PRI | NULL    | auto_increment |
| make  | varchar(50)      |      |     |         |                |
| model | varchar(50)      |      |     |         |                |
```

```
| year  | year(4)           |      |     | 0000    |                |
+-------+-------------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

As you can see, the primary key is called "id", and consists of an unsigned 10-digit integer. Freetag requires some sort of unique integer reference to both users doing tagging and objects that are tagged. On that note, here's the users table schema:

```
mysql> describe users;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| id       | int(10) unsigned |      | PRI | NULL    | auto_increment |
| username | varchar(40)      |      |     |         |                |
| password | varchar(40)      |      |     |         |                |
+----------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

As you can see, the users table also has the integer primary key.

Planning

As in your database, a little bit of planning is in order, in order to better implement Freetag in your application. Here, I'll walk through the steps of implementing Freetag to allow users to tag cars in the sample application.

A good place to begin is by identifying the three main objects you'll need to think about when implementing a folksonomy around Freetag.

1. Tagged Objects – This happens to be whatever you want to tag in your system. In our example, it's going to be cars. You'll need a unique integer key to identify tagged objects uniquely.

2. Taggers – Users, members, actors, agents, whatever your database is set up to track. These 'objects' define the tags on your tagged objects. You'll need a unique integer key to identify taggers uniquely.

3. Tags – Chances are, you know what tags are. But do you know that different applications use tags differently? Freetag models each tag as a 2-tuple consisting of a "raw tag" and its corresponding normalized form. Since Freetag normalizes raw tags by stripping all non-alphanumeric content out, by default, if you need to use additional characters in your normal tag forms, you should

Here are a few examples of normalization:

- Normalized: [platophilosopher] Raw: [Plato, Philosopher!]

- Normalized: [mattsrecumbentbike] Raw: [Matt's Recumbent Bike]

- Normalized: [concert] Raw: [concert]

If you have a number of first-order objects that you'd like to tag, I'd recommend separating two instances of Freetag into their own databases, and creating the different instances by passing different parameters to the constructor.

Anyway, here's the three-object model of our example:

1. Tagged objects – these would be the cars that we want to let users tag. Our unique integer is `cars`.id.

2. Taggers – these are the individual users in our system. Our unique integer is `users`.id

3. Tags – these are the words that we let users set on the cars.

Simple enough, right?

Installation & Configuration

To install Freetag, download http://www.getluky.net/projects/freetag/freetag-latest.tar.gz. Unzip the file to an include directory outside of your public web structure.

```
tar xzvf freetag-latest.tar.gz
```

There will be two files of major interest:

- freetag.class.php - The class file, which includes the freetag api.

- freetag.sql - A SQL script file to use to create your freetag tables in a MySQL database.

You can either use an existing database, or create a new database to store the tag information.

We import the table definitions by running the freetag.sql script file:

```
mysql -u username -p databasename < freetag.sql
```

After it runs, we're ready to rumble.

Integration

This is the fun part! Sample code to go along with this document will eventually be available, but in the interest of providing you with code samples as quickly as possible, we'll just go through some basic site functions and explain common uses of Freetag.

Adding Tagging Capability

The first step that we should take is to allow users to tag our cars. Without tags in the system, we can't test the rest of the functions! So we start out on our car details page.

On that page, we show basic information about the car already – remember, this is supposed to be an existing application, even if our example is simple. We expect a lot of logic already on the page, but what we can do is add a small HTML box or form that looks like this:

```
<div class="formbox">
   <div class="title">Be the first to add some tags!</div>
      <form name="addTags" method="post">
            <input type="hidden" name="car_id" value="<? $car_id ?>" />
            <input type="text" name="tags" maxlength="40" />
            <input type="submit" name="Add" value="Add" />
    </form>
</div>
```

The form will submit to itself and stick the tags as a long string in the $_POST['tags'] variable at next page load. Freetag is configured to allow people to separate tags with spaces, and put multi-word tags in double-quotes to allow for longer or more interesting tags. Remember, though, that raw tags that are multi-word still end up in normalized form in the end.

Here's what the form handling code might look like:

```
<?php
require_once('config.inc.php');
require_once("/path/to/freetag.class.php");

session_start();
$user_id = $_SESSION['user_id'];

$freetag_options = array(
  'db_user' => '<username>',
  'db_pass' => '<password>',
  'db_host' => 'localhost',
  'db_name' => '<databasename>'
  );

$freetag = new freetag($freetag_options);

if (isset($_POST['tags']) && trim($_POST['tags']) != "") {
    $freetag->tag_object($user_id, $_POST['car_id'], $_POST['tags']);
}
?>
```

So that will let us tag the object without having to modify our existing schemas at all. Pretty neat, huh?

Okay, so now that we've got the tags in the Freetag database, we modify the original little HTML form to also display the tags that have been set. Here's another snippet of code that shows what that might look like.

```
<div class="formbox">
<?php
    $user_id = $_SESSION['user_id'];
    $tagArray = $freetag->get_tags_on_object($_POST['car_id'],  0, 0,
$user_id);
    if (count($tagArray) > 0) {
         print "<div class='tags'>";
        foreach ( $tagArray as $tag ) {
             if ( $tag['tagger_id'] == $user_id ) {
                 print  htmlspecialchars($tag['raw_tag']);
            } else {
                 print  htmlspecialchars($tag['tag']);
            }
             print "<br />";
        }
         print "</div>";
    } else {
?>
    <div class="title">Be the first to add some tags!</div>
<?php
    }
?>
      <form name="addTags" method="post">
            <input type="hidden" name="car_id" value="<? $car_id ?>" />
            <input type="text" name="tags" maxlength="40" />
            <input type="submit" name="Add" value="Add" />
    </form>
</div>
```

Do you see what we did? Basically, we run the get_tags_on_object function to get an array of arrays with all the tags on our car. If it turns out that we get some tags, we loop through the tags, and only show the raw tag to the user who originally tagged it as such. If this is another user looking at the tags on the car, they will only see the normalized form. And finally, if there are no tags yet, it shows a little title asking them to be the first to add tags.

Left as an exercise for the reader:

- Allow a user to delete their own tags (Hint: use the section of the conditional that prints out raw_tag).

- Link the tag to a page that shows all cars tagged with that page.
- Create a tag cloud like it was going out of fashion with the handy $freetag->silly_list() function!
- Telling me what bugs are in the sample code that I haven't found!

Conclusion

Out of respect for the fact that I don't know if anyone is actually using Freetag in production yet, I'll stop this quick little guide here. If you ARE using it, though, let me know what sections you'd like me to cover in more detail, and I would be happy to. It's cool watching the excitement about Freetag, and I truly hope that this implementation guide helps you understand how Freetag might fit into your PHP/MySQL application. But it would be wicked super cool to hear how you integrated Freetag into your application, and how long it took you.

Til next time,

-Gordon Luk