Implementation Guide

Document Version 1.0

Freetag Version 0.202

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.

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, it is probably limited to English language tags for now. But more language support is planned.

Here are a few examples of normalization:

- Normalized: [platophilosopher] Raw: [Plato, Philosopher!]
- Normalized: [mattsrecumbentbike] Raw: [Matt's Recumbent Bike]
- Normalized: [concert] Raw: [concert]

Eventually, you should be able to define a folksonomy on various sorts of tagged objects in your database by using namespaces to partition your folksonomies around different tagged object types. The current version only supports the default namespace, but it's high on the priority list.

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. Edit freetag.class.php, and set the private variables for your database connection as needed, and also set the appropriate path to ADOdb (as bundled, or on your system) by changing $ADODB_PATH

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 = new freetag();

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 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