Preliminary Specifications and Official Announcement
Target Version: 1.0
Developer: Tristan Harward – trisweb.com
License: GPL

Description:

zenphoto is a web-based photo gallery application designed with simplicity, elegance, and ease of use in mind. It is based on PHP and MySQL, with three main components: a public view, which displays albums and images through template functions; an administrator view, which allows for smooth management of albums, images, descriptions, and other features; and a backend behind the scenes, which automates album creation, image resizing and processing, storage and retrieval of metadata, and all other functionality.

Key Features (Planned):

- Albums are created from images in folders (within the “albums” folder). A folder with images in it is recognized as an album, and all processing/resizing is done automatically.
- Uploading images to a folder via FTP is a great way to create albums, but alternative methods also exist in the admin interface: upload individual images, a zip file full of images, or point zenphoto to a directory anywhere on the web.
- Names and titles are automatically created from filenames by default – the album’s name will be the folder’s name, the image titles will be their filenames (minus the extension).
- Images are processed quickly, without timeouts. Images are only resized when the resized image is requested by a user. After that, they’re cached, and never have to be processed at that size again. All sizes are fully customizable.
- A great admin interface that allows you to browse your album just like a user (but with special privileges) and change titles and descriptions on the fly like flickr; or batch edit the titles and descriptions of whole albums on a single page; or add/edit/delete albums/images/etc.
- Four main templatable user views: Gallery, Album, Group, and Image. Each can be fully customized through PHP. The default templates are easily modifiable PHP files and the template functions are well documented.
- Comments on individual images just like on most blog posts – Name/email/website and a checkbox to save them. Basic spam protection is included, and the admin interface includes a comment manager to easily delete/edit comments.
- Blog integration tools and widgets – aside from the ability to easily template a zenphoto gallery into your existing web site, images and albums can easily be added to blog posts and web pages. Any image can be accessed and resized on the fly simply by putting the desired size in the URL, like so: /photos/albumname/width/400/john.jpg, for example, will return john.jpg resized to a width of 400 pixels. The image will be cached so it doesn’t have to be processed on every view, just as in a zenphoto album. A blog sidebar widget is also planned, which creates thumbs and links to a few of your most recent images from any group, album, or from the entire gallery, custom sized to fit your sidebar.
- RSS Feeds – Feeds are available on many levels, including individual image comment feeds.
- EXIF Data – All available EXIF data can be displayed through template functions.
- Categorization of images by tags and/or ratings (ala flickr and del.icio.us). Images are organized into groups in which the user can view similarly tagged or rated images. Groups can be created based on any number of criteria, expressed as a set of boolean expressions.
- User-picked custom groups – the user can “hold” images (like Picasa) that they want to put into their own custom group, which can then be exported to a downloadable zip file, or sent to an e-mail address via attachment.
- Plugin Architecture – eventually, wordpress-like hooks will be included for plugin authors to add new functions, trigger on events, add to the database, and more.
Implementation Details:

Zenphoto uses an object-oriented model for representing images, albums, and groups. The template functions are simply convenient ways of accessing these objects and the data contained in them.

The three classes are (for example purposes; these are subject to change):

**Image**

*Constructor:* Retrieve image metadata from database table “images”, and create an entry if one doesn’t already exist.

*Fields (Private):* title, filename, description, containing album, comments[]

*Methods:*
- `getSizedImage(width, height)` – creates and returns the path to a resized image.
- `getFullImage()`
- `setTitle(…)`
- `setDescription(…)`
- `getComments()` – returns array of comments, each an array of comment info.
- `addComment (name, email, web, commentbody)`
- `getCommentCount()`
- `setCommentsAllowed(…)`
- `add/deleteTag(s)(…)`
- `setVisible(…)`
- `getFilename()`
- `getPrev() / getNext()` – uses the folder array to get the prev/next images

**Album**

*Constructor:* Retrieve album metadata from database table “albums”, and create an entry if one doesn’t already exist.

*Fields (Private):* title, folder, description, visible, images[], page

*Methods:*
- `setTitle(…)`
- `setDescription(…)`
- `getImages(page)` – returns array of filenames of images in the page
- `getFolder()`
- `setVisible(…)`
- `getSortOrder(…)`

**Group**

*Constructor:* Retrieve group metadata from database table “groups”. Groups are defined only in the database, so an entry will already exist.

*Fields (Private):* title, folder, description, containing album

*Methods:*
- `setTitle(…)`
- `setDescription(…)`
- `getImages(page)` – returns array of filenames of images in the page
- `getCriteria()` – specify the criteria the group is based on.
- `setVisible(…)`

**Template Functions:**

Template functions are wrappers for the classes which operate on globally defined objects specific to each page. They look something like this, and only work in the template for which they’re named. Each function which outputs a string also has a “print” version, which I’ll omit for now.

More info on the template functions and formats will be available later.

**Example: Displaying a Gallery**

When the user visits your gallery, the program goes through several steps to display it to them, and as a general rule, never does more work than is needed for displaying the current page. This works because
no processing is needed before the user requests an image, so doing it on the fly (at request time, only when needed, et al) means efficient operation and no server timeouts when new images and albums are added.

To display the process behind zenphoto, we’ll look at the example of viewing an album page, and just to make it interesting, let’s say we’re trying to view page two of an album that was just uploaded and hasn’t been viewed yet.

Let’s say the request URL is: http://www.yoursite.com/photos/myalbum/page/2/ which gets converted to http://www.yoursite.com/photos/index.php?album=myalbum&page=2 via the magic of ModRewrite (even though those ugly links will still work for those of you without that privilege).

Let’s say you’ve uploaded 25 photos into the “photos/myalbum/” directory, in whatever way you wanted to do so. They’re huge 1600x1200 files because you have unlimited web space and want people to be able to see the hi res versions. But, of course, in the album, they’ll need resizing. In index.php, zenphoto sees that the user has requested page 2 of the album. It knows the foldername from the URL (the URL album path is the folder name), so the first thing it does is look up the album in the database by its folder name. Since we’ve never seen this album before, it inserts a new entry for it using the folder name as the title, default sort order, blank description, and uses the first image in the folder for the album thumbnail (all of these can be changed later). Now that the album has an entry in the database, zenphoto makes a new Album object out of it, which has methods for updating and querying the database and retrieving images from the album by page. We then call a method of the Album object: getImages(page), and pass in “2” for the page number. This method goes to the album folder and gets an array of all images that zenphoto can handle (jpeg, gif, png, bmp, and more) and sorts them based on the set sort order (which defaults to modification time to show the most recently uploaded images first). It then uses the configured default number of images per page to truncate the array down to images only on the current page (2). Now, in the template, a simple foreach loop creates an Image object from each of the filenames in the array, and displays the linked thumbnail using the Image object’s getThumbnail() method. This method looks in the cache folder to see if the thumbnail has already been created; since it hasn’t, it makes it and caches it and returns the filename, which then gets displayed nicely in the album page.

By this method, each image size is processed only once, and the images are processed incrementally by request from the user, so the server is never bogged down too much at one time, or by you uploading a giant album of images.

A single image display works in almost exactly the same way, with the addition of a couple database calls to get the comments and extra information.

Outline of Program at Runtime (Backend)

- User requests page
  - Get album and image name from the URL
  - Viewing a single image ($get['image'] is set)
    - Create Image object based on album name and image name.
    - In template (via template function wrappers):
      - Call Image->getSizedImage(w, h) and display the result.
        - Looks up sized image in cache; if exists, return it.
        - Else, get the original, resize it, cache it, return it.
      - Call Image->getTitle, Image->getDescription
      - Call Image->getComments
        - Foreach comment, print it (in template)
      - Other template functions.
  - Viewing an Album ($get['image'] not set)
    - Create album object based on album name
    - In template:
- Print album name, description.
- Call Album->getImages(page) to get an array of images.
  - Foreach image, Image->getThumb(), and print with link.
- Print next/prev page links if necessary.

Viewing Index of Albums ($get[‘album’] and $get[‘image’] not set)
  - Get array of directories in ‘albums’ directory.
  - Foreach directory name, create an album object.
  - In template:
    - Foreach album, Call Album->getThumb()
      - Print the thumb and link to the album page.
    - Print next/prev page links if necessary.

Special Requests: custom sized images, bloglet, etc.
  - Delegated by mod_rewrite to specific scripts.

Filesystem/Database Synchronization

Unfortunately, the use of a database, which allows a great deal of functionality that a flat-file system would not, is usually hard to keep synchronized with the filesystem. This is because the nature of a web program means it only runs when requested by a user, and there is no way to monitor the /albums/ directory directly.

Because zenphoto uses only the filesystem to display the gallery structure, it is nearly immune to this problem. Moving or deleting images or albums simply creates new entries in the database, and deleting images or albums makes them invisible to the gallery. However, old images and albums that no longer exist have unneeded entries in the database that remain even after they’re deleted, and since there’s no way to tell when files are deleted on the filesystem, there’s no way to automatically remove them from the database.

An easy way around this is to do a little garbage collection. Since these extra entries don’t affect normal gallery browsing, we can intermittently garbage collect when normal album pages are requested. However, when a search or a group is requested, a garbage collection must be performed beforehand because these actions run across the database and not the filesystem. There is a minimal performance penalty; it’s simply one pass over the database checking for the existence of files. There might be a more elegant way to do this, but that’s the plan for now.

Final Words and Philosophy

Zenphoto will be built from the ground up as the best private-hosting web photo gallery application on the market today. I will consider all solutions currently available, and see what they do well and what they don’t, and I will attempt to create a quality system which provides all of the needed functionality in a small, efficient, and customizable package. It will be designed with simplicity and quality in mind at every turn. I have a vision of what the final product will be like – I imagine it with the simplicity and flexibility of PhotoStack, the user interface innovations of flickr, the power of Gallery, and the interactivity of WordPress all rolled into one great app that should be a joy to use.

I look forward to building zenphoto, and if I do it right, it might just become the definitive photo gallery software for the web. That’s my goal. Stay tuned for future news.

Tristan Harward
http://www.trisweb.com/