Rough Concept Publisher

Database Specifications

# Database Information

# Naming

A sort of Hungarian notation is used, items are prefixed according to what they are. The name itself should describe what the object is, names should be as short as possible but long enough to uniquely describe the object.

## Notation

* tbl: Table, eg tblContent.
* txt: Text including blobs of text and character arrays, eg txtTitle.
* n: Integer, eg nNewItem.
* f: Float, size doesn’t matter, eg fPrice.
* b: Bool value.
* dt: Date, eg dtCreated.
* qry: Query (usually in php, not in the database itself).
* id: When refering to an id in another table, it should be follow by the name of the table without the leading tbl, eg idContent would refer to the id of the item in tblContent. Also a table id field may be “id” without a name so in the table Content the id is “id” instead of “idContent”, “idContent” is acceptable but not prefered. When there is more than one id linked from a table and underscore should separate the text that distingueshes what the two ids are, eg idUser\_From, idUser\_To, both of these ids link to the table tblUser so a dinstinction must be made as to what the two references are used for.

PHP Specifications

RC Publisher uses object oriented php. Each page has it’s own class which inherits from a base class. The base class is responsible for the website layout. The class for each page only manages what is displayed in the content section of each page.

# Naming

Variable names should be very descriptive of the variable, a hungarian type notation is used.

## Notation

* n: Integer types.
* b: Bool types.
* f: Floating point types.
* s (used to be str, but depreciated): String types.
* qry: Strings that are specifically an sql query.
* m\_: Class member attributes.

Function and method names should be have each word capitalized, eg TheFunction($nValue).

Class names start with C and then have a name using the same format as functions. They should be in a file named after the class but in all lowercase letters and words separated by underscores. Only one class should be used per file. Class files themselves should not be webpages, and should be in a private folder, another php script should load the class and display the contents of it.

$\_GET and $\_POST variable names should generally be one word, and all lower case, if more than one word is needed to describe it then it should be separated by underscores. If a name might need two words, but it would make the name really long an abbreivation should be used for the name (eg u\_file instead of upload\_file).

Pages

# Main Page (index.php, index.php?content=home)

The main page displays only the newest content that has been uploaded. New comments are shown, as well as new content from the blog site. A detailed design spec has not been documented yet, but that will come shortly.

The basic idea is that the newest content, and links to all other content is presented to the viewer.

# Upload Page (index.php?content=upload)

The upload page allows new content to be uploaded to the site. Only users with the appropriate access can upload content. The upload page performs various checks to make sure that the uploaded content is formatted correctly. The upload page will upload anything, but it shows a warning if the content is not in PDF format.

# Login Page (index.php?content=login[&logout])

The login page allows an administrator to log in. The page performs authentication. It also allows logged in users to logout with the appropriate parameters.

# Content Page (index.php?content=content&id=(?))

The content page displays the content of a particular piece of content. It also allows a logged in administrator to go to the edit content page for that particular item. Non-users may also post comments about the particular piece. Administrators may also delete comments.

# Inbox (index.php?content=email[&message=(?)])

Allows a logged in user to see the messages that they have been sent, delete them, and read them.

# Contact Page (index.php?content=contact).

Allows someone to email a registered user.

User Access

User accounts have an access level. Each page is assigned an access level, with 1 being the lowest level, and all public pages have an access level of 1. If a browser tries to navigate to a restricted page, the page will be not be processed at all (POST or GET messages will not be processed). Only a message stating that the page is restricted will be displayed. User access is handled by PHP sessions with variables stored in $\_SESSION.

# Passwords

Passwords are stored in the database by their MD5 hash. Because the web server does not have an encryption certificate, the login is managed by javascript hashing the password into md5. Then a new string is created with a unique key appended to the hashed password, and that string is then hashed using sha1. In order to authenticate the user, the password is retrieved from the database, and then the same alogrithm for the sha1 hashing is applied to the retrieved password, and the two are compared. That way the actual password, and it’s md5 hash are not transmitted over the internet, it all the actual password exists only on the client, and only a hashed version of the md5 is transmitted. Not as secure as SSL but much cheaper than a certificate.

RC Publisher 2 (This Supersedes the Above)

# Specifications

RC Publisher 2 is to be a Wiki type web experience, though not in the traditional sense as it is not intended for public modification. It is meant to be an easy way for website administrators to add and delete content.

The site is divided into two main parts, with some additional features. The two main features are pages and files. The additional features include a news feed.

## Pages

Pages are the brunt of the website. A page is defined by text which contains a basic markup language. Pages may imbed certain types of files (such as images). A complete history of each page is stored in case a previous revision needs to be restored. Each page may be commented on. Pages may be deleted, but this merely makes them inaccessible, the entire history is still maintained and they may be restored. Comments may be deleted and otherwise moderated. Deleted comments cannot be restored.

## Files

Any type of file may be uploaded. Most likely image files will be uploaded. A file may also be pulled from another site and then stored locally. A file may also be replaced by a new version of the file or deleted. A history of each file is not maintained. Images and other file types may be embedded into a page. Files are stored in a special files directory, the path to a file is given by the /f/i/filename.ext (first two letters for path, followed by the full name of the filename). Files must be at least three characters long before the extension. All filenames are converted to lower case when they are uploaded. Any references to files is also converted to lower case. All files are referred to by their filename, the actual path is irrelevant and exists only to speed up loading a file from the file system.

## News Feed

The news feed is meant to be an aggregate of news stories. New news stories may be posted, and a list of news stores may be displayed on a page.

# Database Tables

## Page Table

The page table is a list of all pages that are available or ever have been available. It does not actually contain page data, instead it contains a reference to the page ID as well as the “current” version of the page. This contains the page slug.

### tblPages

* id
* txtSlug
* idVersion\_Current
* txtBodyHTMLCache (This is a formatted version of tblPageHistory::txtPage, that is convered from RC markup to html).

## Page History Table

This contains all the actual page data and page title for the entire history of all the pages.

### tblPageHistory

* id
* idPage (matches tblPages::id)
* idVersion (current version of page is tblPages::idVersion\_Current)
* txtTitle
* txtBody
* dt (date this revision was posted)

### tblNews

* id
* dt
* txtTitle
* txtArticle

### tblFiles

* id
* txtSlug
* txtDesc
* dt
* txtType (Predefined types img, pdf, txt, etc…)
* txtFileLocation (location of file should always be txtName[0]/txtName[1]/txtName)

# RC Markup

The idea behind RC markup is to create something capable of many of the features of HTML, but limited to the needs of an RC site. It is also meant to be typed more naturally without the need of tags. Many of the features of RC markup will be inspired by MediaWiki markup. Some items include:

* Italicized text is given by ‘’two single quotes’’.
* Bold text is given by ‘’’three single quotes’’’.
* Bullet points are declared with a “:\*” (That is colon star). And the newline break. Generally you’ll want to declare bullets at the beginning of a line, but other possibilities are allowed.
* Links are done with [[page Brackets for an internal link.]]
* External links are done with [[site:{{address}} Description.]]
* Files are embedded with [[file:slug {{parameters}}]]. Parameters may be different depending on the type of file. Different parameters are separated by pipes “|”. Parameters should be unique enough that it doesn’t matter what order they are in. Currently image files resolve to display the image. All other files turn into a link.
* Blog links. Links to the associated blog are given by [[blog:{{slug-url}} Description.]] The description is optional and it will be the text for the link.