zenyan Login Module

zenyan
Login Module

zenyan login module

Table of Contents

Login module Essentials

A Brief Introduction

Security is an essential component to many intranet and internet applications. If you are building static web pages available for everyone then this module will not be of much value to you. If you are building a web application where you need "out of the box" security and context management coupled with a great deal of flexibility then this may be a module to consider.

This particular section will not focus on using the module. This is covered in the Admin Manual. It will be assumed that you have already implemented the login module (which is the last section in this doc, purposely placed at the end as it is something you will only be doing once).

One final comment. I struggle with the term framework and module pertaining to the login infrastructure. For me the notion of a module is that it is (or can be) modular. To me, a framework, is something that is foundational. The Login infrastructure is interesting in that it really is modular. You do not have to implement login. But, when you consider what zenyan is used for it is hard to imagine building an application that does not contain a login component. In this regard login becomes a framework. In the end I decided to call it a module. Sometime the world is gray.

Login module Functional Workflows

Login Workflow

The following is the workflow pertaining to the user login function.

Illustration 1: User Login Function
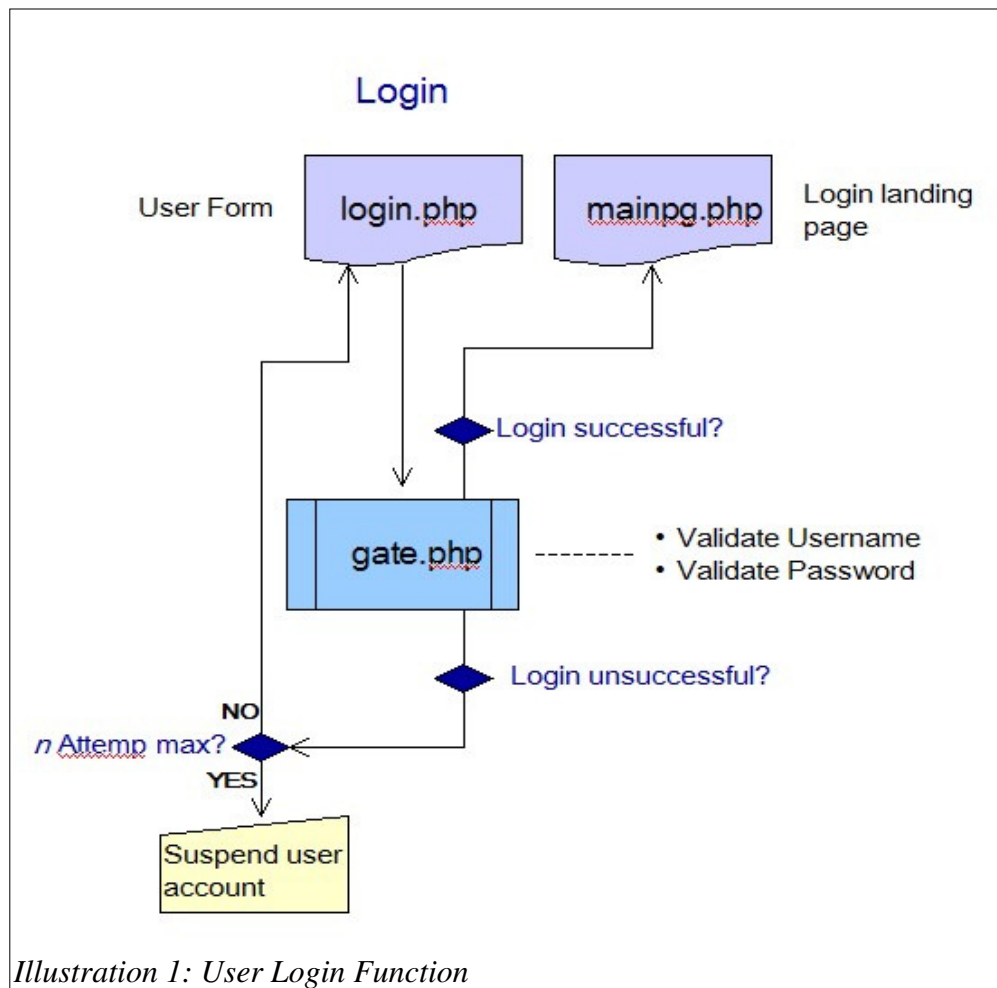
The main goal of login is user and password validation. gate.php is the login handler program. "Out of the box" it provides a minimum data subset. Refer to the implementation section for details pertaining to implementing, configuring, tweaking, and extending this function.

Login Administration

One of the biggesest changes as of zenyan release 0.96 was enhancement of the login administration function. Another big change was the introduction of the notion of groups and users

into the login equation. It is not possible to create user groups within zenyan. Further it is possible to create a singular group login that many users are attached to. There is a great deal of operational flexibility with the login module. How you choose to implement it is up to you. Of course you can still utilize the login infrastructure for single individuals to have login access.

Registration

There are three different workflows that allow for registration. They are:

1. **Users can register themselves.** This function is most applicable for internet sites where you are seeking membership. It can also be implemented in any other situation as well where you want to allow an end user to self-register.

2. **Group Admins can register users.** This function allows an end user that is designated as a group administrator to be able to register users.

3. **System Admins can register users.** This function allows system administrators to be able to register users.

The following is the workflow pertaining to each of the the user registration workflows.

User Self-Registration

User Self-Registration

regfrm.php

- Verify user is not already registered
- Validate email address
- Validate User name
- Validate answer
- Register new user
- Send email

regfinal.php

Yes   No

Register User

regfinalval.php

- User completes registration
- Create link items row
- Update registration flag

Group Admin Registration

Group Admin Registration

adm_group_loginmgmt.php

insertNewUser ()

validateNewLoginName()
validateNewPassword()
createNewUser()

Send email

usrgrplib.php

loginlib.php

System Admin Registration
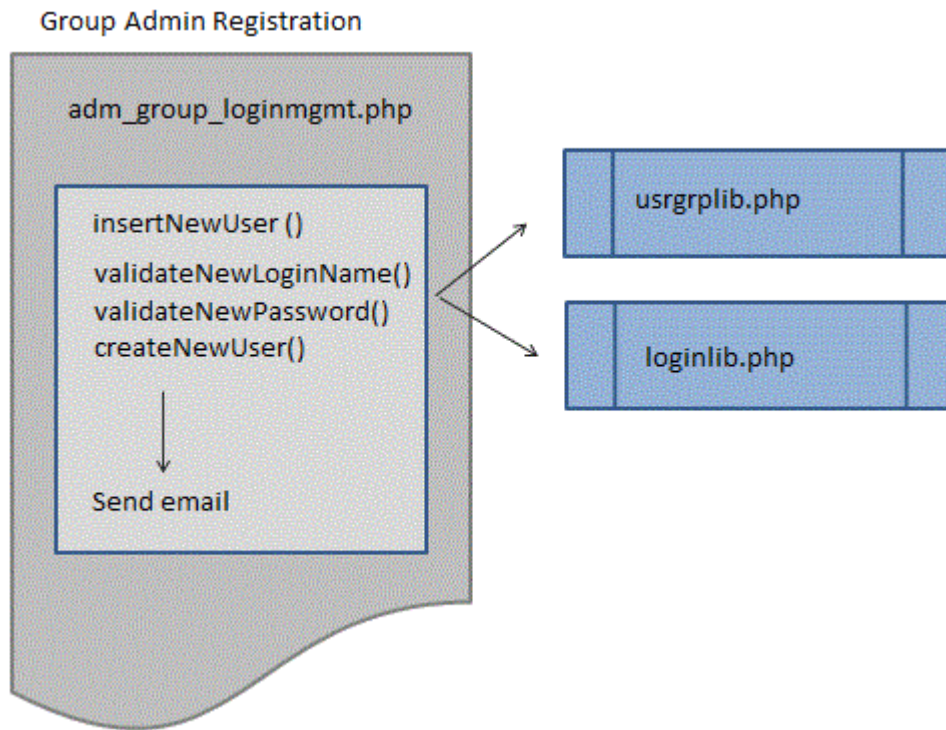
System Admin Registration
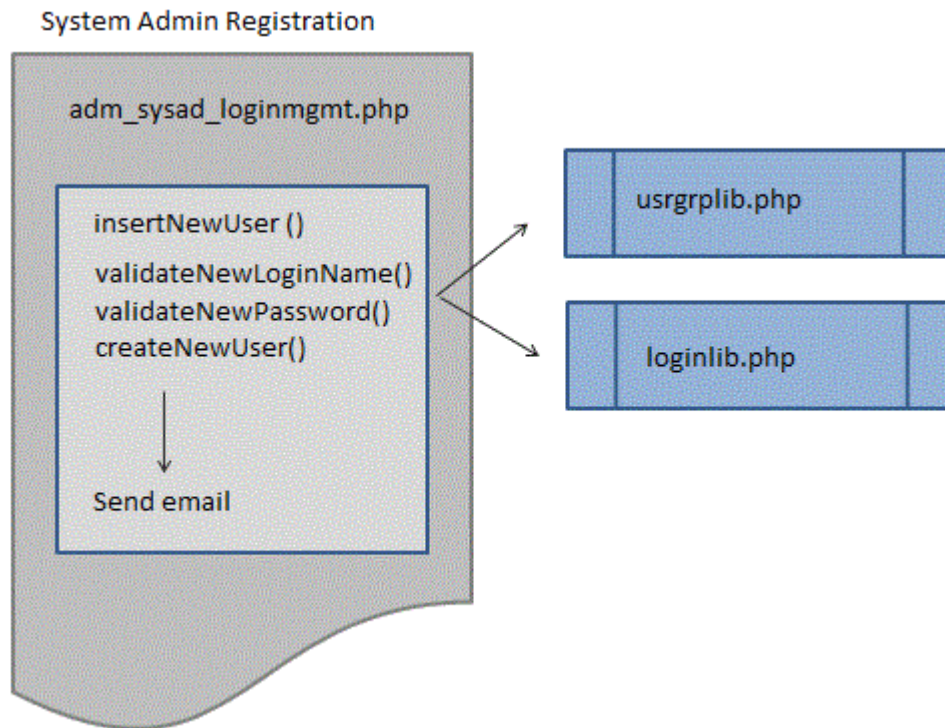
The difference between the system admin and the group admin functions is that the system admin function has the ability to add new groups, the group admin does not.

Complete List of System Administration Login Management Workflows

- Register New User/Group
- Reset User/Group Password
- Suspend User/Group
- Unsuspend User/Group
- Delete User/Group

Complete List of Group Administration Login Management Workflows

- Register New User
- Suspend User from Group
- Unsuspend User from Group

Refer to the implementation section for details pertaining to implementing, configuring, tweaking, and extending these functions.

Default Forms

You will note that zenyan forms, while functional, are very skeletal in terms of appearance. They are designed to be implemented using your styles and desired formatting/layout.

Extensibility

The login module "out of the box" provides a minimum, working login and context management framework. It was intentionally designed to be configured and extended based upon your specific needs.

Login module High Level Technical View

This is a diagram providing a high level technical view of the login module.
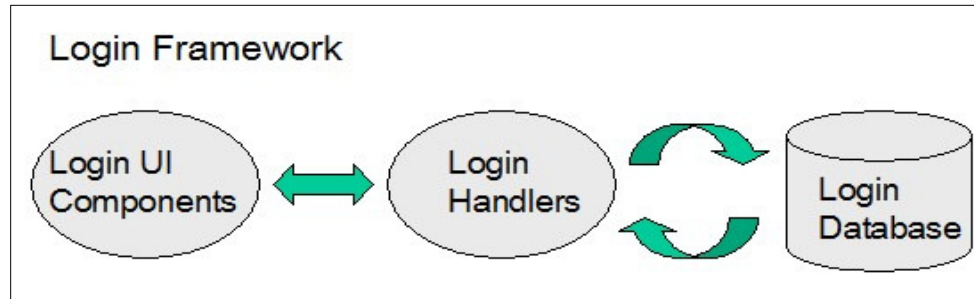
Details pertaining to each file are covered in the implementation section. Here is a list of major login module files and there location within the zenyan module.

UI Components and Handlers

Login module files (UI components and handlers) reside in the zenyan root directory. The core files pertaining to the login modules are (in alpabetic order):

- adm_group_loginmgmt.php

- adm_sysad_loginmgmt.php

- gate.php

- gatereg.php

- gatesusp.php

- gateunsusp.php

- regfinal.php

- regfrm.php

- regsusp.php

- eLib/loginlib.php

- eLib/usrgrplib.php

Configuration Parameters

Parameters for the login module reside in the following files.

`cLib/clvars.php`

`eLib/envvars.php`

Other Dependency Files Required By Login module

`eConfig/envref.php`

`eLib/applib.php`

`eLib/o_dbiomyp.php`

`eLib/loggers.php`

Database Schema

The following represents the MySQL table schemas that makeup the login module database.

`schema/zenyan_mysql_core.schema`

The database consists of three base tables.

1. gate
2. gatehist
3. gategrps

All tables must be implemented.

Data to initially load into the tables:

schema/load_gate.sql

schema/load_gategrps.sql

schema/load_gatehist.sql

Architectural Considerations – Wide Variances in Requirements

I have had a few gigs where I could reuse 100% of my security and login work from a previous gig. But, this has been the exception, not the rule. I make this point in defense of the separation of functions into their own code file. I also separate the UI (User Interface) form from the code in order to provide maximum flexibility regarding application design / look-n-feel.

I have built enough different applications to know that such flexibility is essential. Even so, and even as flexible as the zenyan framework is, it never seems flexible enough.

For example, some applications will resend the user their existing password, whereas others will generate a new temporary password. What if you are converting data from one application to another and cannot move the encrypted passwords from the old system. How do you deal with that scenario? What if your requirements dictate that users can belong to different login groups?

How do you deal with the requirement for aging passwords and requiring them to be changed over a given periodicity? The fact is, there are so many varying requirements that even the approach I have opted for, while enormously flexible, just cannot handle them all without some degree of customization.

Additionally, by using this technique new functionality can be added at any time and can be implemented (or not implemented) with minimal disruption to the existing login module.

Security Code Discussion

This section provides a discussion about the various baseline security and encryption techniques used within the login module. This section does not get into full implementation and configuration details, though it does address certain configuration details as they pertain directly to this discussion.

Encryption Techniques Used

The zenyan login module uses MD5 as its encryption standard. MD5 by itself has been cracked. But, I am using a dual salt technique that makes the login module reasonable secure.

It is always a challenge when designing a module that is open and available to everyone (including the hackers). It should also be understood that this is only a login security module and assumes that you have taken measures to secure your Apache web server and your web servers file system from external hacks and attacks.

Before continuing I want to disclose that I am by no means an expert in encryption. Far from it! I spent considerable time researching the subject. I was looking to create a secure login infrastructure. I was not looking to create something that would pass DOD muster or that could be declared hardy enough to store credit card numbers on public servers.

In doing my research I found the page http://php.net/manual.en/function.md5.php to be an invaluable resource. The discussion threads and user contributed examples were a very good learning tool for me.

Password Encryption Technique

When the user registers they are required to supply a default password that is a minimum of eight characters. The defaults are such that the password must contain at least one lower case, one upper case, and one number. The password is them concatenated with a 21 character guid that gets generated at the time of registration. This concatenation represents the first salt. A second salt is created by pulling three of the characters from the guid and also performing a concatenation. These three characters are defined at configuration time by setting the values of the variables $seed1-$seed3 to a value from 0 to 20. The following string gets concatenated and an MD5 signature is created. This is the password that is stored in the database.

Password + GUID + Seed1 + Seed2 + Seed3

Integrity of the Login System

When I approached creating this module I considered two important variables.

1. The source code would be publicly available to anyone

2. How could I create a system that was as secure as it could reasonable be from hack/crack attempts from someone that was "on the inside" and has administrative access to everything?

Creating a two stage salt that consists of a) a uniquely generate GUID; b) a three character signature that is uniquely defined by each implementation (assuming they do not leave the default settings) makes for a very secure login system in terms of being cracked from the outside.

Disabling Login

Globally

All you need to do is change the value of $Seed1 or $Seed2 or Seed3 and this will disable/break login access for **all** users!

Individual User

Note: there are functions to suspend user access. There are a number of ways to accomplish this. Some of them are...

- Modify the GUID column value
- Remove the GUID column value
- Modify the pwd column MD5 hash string value

Cracking the Encryption Technique

External Threats

In order to break the baseline encryption schema defined in the zenyan login module one needs access to the database, specifically to the guid column. Also, the cracker needs access to the values set for $seed1, $seed2, $seed3. Actually, having the values of $seed1-$seed3 and the guid value are the most important items for the cracker to have. Without these values it is not really possible to decrypt the password.

Techniques to Further Harden the Login module

In the handler gatereg.php you can change order of the right operand of the following line of code.

```
$ipwd = trim($newpwd) . trim($g) . trim($s1) . trim($s2) . trim($s3);
```

If you do make changes, you will also need to modify the following line of code in the handler gate.php.

```
$cpwd = trim($password) . $rguid  . trim($s1) . trim($s2) . trim($s3);
```

You can also extend the length of $seed1-$seed3. By doing so you increase the probability required in creating a rainbow table should someone be able to gain access to a lname, pwd and guid column, but still not have the $seed1-$seed3 values.

Finally in the file ../zenyan/eLib/envvars.php you can modify the allowable characters used in $logincharset. Just keep in mind that you can create potential problems if you change your dbms's ascii collation set or implement on servers that have different extended character sets that your users have when registering.

And... always test before implementing your customs!

Misc Other Security Techniques

Encryption alone is not enough to ensure a secure system.

Preventing Brute Force Attacks

We want to be able to protect ourselves against brute force hacking. In order to do so we limit login attempts to the value (+ one) defined in the environment variable $maxloginattemps. There are a few scenarios we need to contend with. For one, a legitimate user may have forgotten their password and are trying to guess it. After $maxloginattemps is reached the users account is suspended and they are notified to contact support to have it restored.

You will likely want to enhance this to offer the user to ability for the user to recover or reset their password and send it to the email address on record. The bare bones implementation does not require an email address so this would be an add-on (at this point in time).

This scenario is well and good but not cover someone trying a brute force attack against a compromised login name or against random login attempts.

To account for this attack scenario once ($maxloginattemps + 5) is reached the user is redirected to loginhackattempt.php which is an empty page. This should be enough to minimize brute force attacks.

IP Spoofing

It is possible for the hacker to write a program that spoof the IP/DNS name and launches a brute force attack against the zenyan login system. In order to prevent this from occurring the IP is checked against the server to which the login module is implemented. While not 100% bulletproof it will exclude a large percentage of hackers from trying to breach the system.

Future Enhancements to Login Security

 following are some ideas I have for future enhancements (just ideas, no hard and fast promises for delivery).

- ◆ Add heuristic checking that uses time to determine if entity attempting to login is human or a program.

Parting Thoughts

There is no such thing as a 100% bullet-proof system. But, your general goal is to implement security in a manner where you reduce the probability of people attacking you. I remember years ago reviewing my Apache logs and finding common attacks (for that era) for a posix server. Problem was I was running Apache on Windows. I would take some of the Attacks and do a reverse DNS lookup. On one I was able to get the guys email, home phone, and home address as well as other personal information. From what I could tell he was a college student in South Korea. that was running his own server. I sent him a rather amusing email and needless to say never

heard form him again.

My point is that decent security will protect you from the vast majority of the hacker world. As far as the real pro hackers go, it is likely unless you become eBay, Amazon, Facebook or some big web presence you are not going to even appear on these folks radar.

Implementing the Login Module

A great deal of time and consideration needs to happen in implementing the login module. Keep in mind that you can always make changes and modifications until you go-live. Once you put this module into production then it is going to be harder to make changes and implement new functionality.

Getting Started

The zenyan login module is meant to work "out of the box." I am assuming that you have already implemented the full zenyan framework, to which this is a part of. The module works on both Posix (Linux, Unix...) and Windows. As such there is nothing for you to install to the file system if you have already installed zenyan.

You will need to build the login module database in your MySQL implementation. You will find the code to do this in...

```
../zenyan/schema/zenyan_mysql_gate.schema
```

Required Configuration

After building the zenyan login database the first thing to do is open the following files and make the following configuration settings.

```
../zenyan/cLib/clvars.php
```

$Scontxttoll = 8224;

Modify this value to some number or string. My recommendation is to select a number between 0 and 100,000. This variable acts as a simple token for carrying context. If you require more complex carrying context logic you are free to add your own.

Or, inherently you can require that a php page is only called from another php page. If you look at the baseline code you will see this in the following code patterns that validate login in order to display a page.

Login Validation Pattern #1

```
if ($_SESSION['initentry'] != $Scontxttoll) {
  $_SESSION['loginerr'] = 'Login required to access page';
  header("Location: login.php");
  exit;
}
```

This pattern is less restrictive and secure as pattern #2.

15

Login Validation Pattern #2

```
//get only page name

$fullrefpg = $_SERVER['HTTP_REFERER'];

if (preg_match("/[A-Za-z0-9_-]+[.]php.*$/", $fullrefpg, $matches)) {$refpg =
$matches[0];}

$refpg = preg_replace("/^([A-Za-z0-9_-]+[.]php)(.*)$/", "$1", $refpg);

$_SESSION['refpgnm'] = $refpg;
```

What this code does is gets the referring page, strips the path information and updates the the session variable $_SESSION['refpgnm'] which can be checked in the called program. An example of this is in the handler gatereg.php.

```
// deal with potential ip spoofing

$pathchk = "";
if (preg_match("/^http:[\/][\/](.+)[\/](.+)[\/](.+)/", $fullrefpg, $matches2))
{$pathchk = $matches2[1];}

if ($pathchk == $dns or $pathchk == $ip)

{

   // validate referring page

   if ($_SESSION['refpgnm'] == 'register.php')

   {
```

Whatever you set the $Scontxttoll value to you also need to set the same value to in.

`eLib/envvars.php`

...Which is the next configuration file we are going to deal with.

We need to review and modify the following login module variables from envvars.php.

Variable	Desc	Notes
$maxloginattemps	Maximum login attempts	The maximum # of login attempts before a Users account is inactivated.
$minloginnm	Minimum login name length	Default is 6. This is the min setting for the column "lname" in the "gate" table. Keep in mind the login name must be unique as it is the primary key of the table "gate". Typically you would want to increase this value. The handler program gatereg.php will check to ensure a user registering has entered the minimum number or characters. It will also check to ensure that the name requested is not already registered.
$maxloginnm	Maximum login name length	Default is 35. This is the max setting for the column "lname" in the "gate" table. the database can handle

16

Variable	Desc	Notes
		a name up to 80 characters in length which should be large enough to store email address as the login name.
$minpwd	Minimum password length	Default is 8. What is stored in the database is a 32 character MD5 signature. You can increase or decrease this setting though I do not recommend making the number less than 6.
$maxpwd	Maximum password length	Default is 10. I would not recommend making this any larger than 12.
$seed1	Login salt	Set to a value 0 through 20. Do not use default value.
$seed2	Login salt	Set to a value 0 through 20. Do not leave as default.
$seed3	Login salt	Set to a value 0 through 20. Do not use default value. Once $seed values are set they cannot be changed without breaking login passwords.
$loghistory	Enables or disables writing to the login history table gatehist	Empty value means this table will not be written to. Logging to this table is enabled as the default.

Other Login Parameters

The following parameters are currently not used, but are reserved for the password generator that I have not yet implemented.

```
$pwdcharset1
```

```
$pwdcharset2
```

```
$pwdcharset3
```

```
$pwdcharset4
```

Externalizing Parameters

The parameters have been externalized to allow the implementor maximum flexibility. But, certain parameters have also been externalized from the programs in order to enhance security.

Login Database Decisions

$loghistory setting in ...

```
eLib/envvars.php
```

To disable logging to the gatehist table set the value to null...

$loghistory = "";

To enable logging to the gatehist table set the variable to any value.

Using the Login Database

Over the years I have seen databases utilized in ways that the developer never intended. I am not sure this is necessarily a bad thing. More often than not it happens due to limitations within the database. Before I discuss the columns available for use here are some general implementation guidelines.

General Guidelines for the Login module Database

- ◆ With all of these guidelines I have to assume that you have the requisite skills to modify and extend the database in a way that does not screw things up or leads to performance problems.
- ◆ It is okay to add additional columns to meet your data requirements.
- ◆ It is okay to increase the precision of columns with the understanding that doing so may require modifying the application code.
- ◆ It is okay to add indexes to the database.

We are now ready to get into the details of the database. In order to offer the maximum flexibility and account for different needs, the login module only implements the absolute minimum subset of columns. There are other columns in the database you may want to utilize. The following will be a description of the login module database followed by information on what you need to do to implement and use certain columns.

Tables

gate

This table contains one row for each logged in entity. The term entity is used here. Though an entity will typically be a single user, the database was designed in a manner that supports the notion of groups and the cardinality that a group can consist of many users, and users can belong to many groups.

gategrps

This table will contain one entry for each entity and is the table that supports the ability that a group that can consist of many users, and users can belong to many groups. At present (Jul 10, 2010) I have not implemented the UI code that utilizes the notion of groups and users. However, for each registered user a row is created in this table.

gatehist

This table is a replication of the gate table. It provides a historical log of activity in the gate table (inserts, updates, etc...). There is a configuration switch that allows you to disable writing to this table. The switch default is enabled. Also, as you add your own functionality to the login module it is your responsibility to also write to this table if so desired.

Table/Column Description

gate (and gatehist)

There are many additional columns that can be utilized. The following is a list and description of each.

Column	Description
lname	This is the logon user name. It is the primary key of the table and as such much be unique. In other words, every registered user must have a unique name. For internet sites storing an email address can be a good use of this field as it can prevent a user from registering multiple time (at least under the same email address).
requestdate	This is the date/time of registration.
pwd	This is the MD5 encrypted password.
firstname	
lastname	
country	
state	
chall_qst	This can be used in conjunction with chall_ans to deal with situations where the user has forgotten their password and/or user name, or as a challenge if you suspect there is a security breach in progress.
chall_ans chall_hint	See chall_qst.
curr_email_addr	If you are using email address as the lname field it constitutes the primary key of the table. A user may change their email address. Since lname is a pk you cannot update this column. Your only remedy is to delete the row and reinsert it with the updated address, or, you can extend to write lname also to curr_email_addr. This way if a user changes their email address you can update and utilize this field as your login lookup name.
confrm_email_sent_flg	A flag you can use if you are sending email confirmations or communications pertaining to registration.
inactive_flg	This flag is used to to inactivate a user. If it is "" the user is active. If set to any other value the user is inactive.inactive_flg
contributor_name contributor_url contributor_short_bio	These columns are used to support applications that allow users to also contribute content to the site you are implementing.

Column	Description
guid	This is a 21 character unique ID that is generated for each registered entity. It is also the field that is used to add salts to the encrypted password.
mbrshpstat mbrshpconfirm	Other columns you can use pertaining to the notion of membership within your implementation.
cstm1 cstm2 cstm3	Three custom flags you can use however you want.
referral	A text field you can use to store information pertaining to referrals.
entity_typ_flg	Determines if entity is a user or group or whatever designation levels you care to implement. By default this column is set to "user" for registered entities.
admin_flg	This flag was intended to identify users that might have administrative privileges to some degree to the application you are implementing. For example, can define users you want to perform certain functions like suspending and unsuspending users.

gategrps

This is an associative table. Though unlike most associate tables its links back to a single table (gate).

Column	Description
parent_guid	This is foreign key (fk) representing the guid column from gate. This column represents the parent entity. The entity defined here can either be a user or a group. Every registered group and user will always contain a base row where the parent_guid is equal to the child_guid. This is how you can always determine you are at the base parent entity in this table. This table is written to even if you are not implementing the notion of groups in your application.
child_guid	This is foreign key (fk) representing the guid column from gate. If you are implementing the notion of a group this will represent user or subgroup that belongs to the parent_guid.
inactive_flg	This flag is used to to inactivate a user or group. If it is "" the user is active. If set to any other value the user is inactive.inactive_flg. Unless you implement the notion of a group within your use of this module this column has

Column	Description
	not real function.

Using columns in default functions

In terms of updating the database it is just a matter of a) adding the columns you want to use to your UI form code, and; b) modifying the handler code to deal with these columns.

Extending the module

General Guidelines

Please note that these are general guidelines I follow! This is not a "do as I say but not as I do" sermon!

◆ Try not to modify the database in a manner that will not be backwards compatible. It you have no choice you need to factor in how you are going to interface and convert existing implementations data into the new database!

◆ Introduce new functionality in a manner that will not break existing functionality. This is easier said than done!

◆ If you are making a modification to existing functionality that others also will be using you have to do so in a manner that will not break their application use of that functionality. If you cannot guarantee that your changes will meet this rule then you need to a new instance of that function, thus preserving the old function. Be evolutionary not revolutionary.

◆ Feel free to add or modify any aspect of this or any facet of the zenyan framework within the parameters of the licensing agreements. But, do so with the understanding that you are customizing the module and as such you are the one responsible for maintaining and supporting your work.