ethna.en Documentation

Release 2.6

Pulkit

Oct 27, 2017

Contents

Warning: To refer to the original documentation (*in Japanese*). Please check the following links.

- http://ethna.jp/doc/ (*New* Under Development)
- http://ethna.jp/old/
- News Archive (In Japanese)

CHAPTER 1

Welcome to Ethna

Ethna is a web application framework using PHP.

- MVC + Forms + many powerful Features

Ethna's slogan which translates that

"It is a framework that is written without being too complex & you can write an application with the core logic and exquisite sense of balance."

CHAPTER 2

Documentation

Contents:

Quick Start

Installation is simple using `PEAR`

```
$ pear channel-discover pear.ethna.jp
$ pear install -a ethna/ethna
```

Note: If permission errors or not able to discover then try with `sudo` like

```
$ sudo pear channel-discover pear.ethna.jp
```

Note: In `$ pear install -a ethna/ethna` we gave the option `-a` so that the dependencies are installed together. Hence, it is advicable to say **YES** when asked to install `Smarty`.

Installation

Install Using PEAR

Ethna can be installed using pear command. It is the simplest and recommended. Note that you need to have pear installed on your system.

Only for the first time, you need to register a channel for Ethna using `channel-discover` as:

Warning: If your pear version is old, then it `channel-discover` will get to discover. Therefore, please run

```
pear upgrade pear
```

```
$ pear channel-discover pear.ethna.jp
```

Once the channel of Ethna is discovered, you are ready to install Ethna

```
$ pear install -a ethna/ethna
```

From this command, the dependent library `Smarty` and `simpletest` will be installed at the same time. If you don't want to install them together then just ommit the option `-a`.

Once the installation is finished, verify `ethna`, using the following command.

```
$ ethna -v
Ethna 2.6.0 (using PHP 5.3.6-6~dotdeb.1)

Copyright (c) 2004-2011,
Masaki Fujimoto <fujimoto@php.net>
halt feits <halt.feits@gmail.com>
Takuya Ookubo <sfio@sakura.ai.to>
nozzzzz <nozzzzz@gmail.com>
cocoitiban <cocoiti@comio.info>
Yoshinari Takaoka <takaoka@beatcraft.com>
Sotaro Karasawa <sotaro.k@gmail.com>

http://ethna.jp/
```

Note: The stable Version 2.5.0 will be installed. Ethna 2.6.0 is currently in beta. If you wish to install the beta then choose the following command.

```
$ pear install ethna/ethna-beta
```

Update Ethna using PEAR

To update ethna, you may run the following

```
$ pear upgrade ethna/ethna
```

It is also possible to upgrade via downloading the package using wget.

```
$ wget -O Ethna-2.6.x.tar.gz  https://github.com/ethna/ethna/tarball/2.6.x
$ pear upgrade Ethna-2.x.y.tgz
```

Tutorial

Setup 1 Projects

This section teaches you to build your project in just a few steps with Ethna using **command line**.

Hint: Make sure once the Ethna is installed on your computer, you have made changes to your `php.ini` files. i.e. pear libraries are in the include path. On a Mac, you would do something like the following

```
$ emacs /etc/php.ini

And then add the appropriate path

; include_path = ".:/Users/YOUR-USER-NAME/pear/share/pear/"
```

Similarly, make the changes to your php.ini file on other Operating Systems.

Simple Blog Application

Here, we are going to create a simple blog application with Ethna and ethna's command line tool.

In this tutorial, there are two important learnings

1. `Ethna` flow of creating an application
2. How to set up a url

1. Application information

Before developing an Application, there are two points to keep in mind.

1. Application ID (*must be letters*) e.g. miniblog
2. Application Directory. Usually on Linux, it is `/var/www/`. Although, you may change it your directory by modifying the `/etc/apache/httpd.config` file.

In this tutorial, we are going to use `/var/www/`.

To create a new project command `ethna add-project AwesomeName`

```
$ cd /var/www
$ ethna add-project miniblog
```

The above command will create a skeleton for the application. Following is the sample output.

```
$ ethna add-project miniblog
creating directory (/var/www/miniblog) [y/n]: y
project sub directory created [/var/www/miniblog/app]
project sub directory created [/var/www/miniblog/app/action]

...

file generated [/usr/share/php/Ethna/skel/skel.view_test.php -> /var/www/miniblog/
↪skel/skel.view_test.php]

project skeleton for [miniblog] is successfully generated at [/var/www/miniblog]
```

Above application has been generated.

Verifying the Directory Structure

Let's take a peek at the directory and the files generated.

```
| - App (directory of the application)
| | - Action (Action Script)
| | - plugin (filter script)
| | - test (test scripts)
| `- View (view script)
| - bin ( command line scripts)
| - etc (configuration files, etc.)
| - lib library) (application
| - locale
| `- ja_JP
| - log (log file)
| - schema (DB schema, etc.)
| - - skel (skeleton application file)
| - template
| `- ja_JP (template file)
| - tmp (temp files)
`- www (web publishing file)
```

Common use is in the following directory.

- app/
- app/action
- app/view
- template/ja_JP

2. Setting Application's Url

Although you may see it on the localhost/miniblog/www. Also, in order to access the application from web browser, i.e. var/www/. We are going to set the **Apache Virtual Host** to http://miniblog.myhost/ like the following:

```
<VirtualHost *:80>
   ServerName miniblog.myhost
   DocumentRoot /var/www/miniblog/www
</VirtualHost>
```

Set up is complete lets try to view in the browser

```
http://miniblog.myhost/
```

If the set up is done correctly, you may should be able to see the following output

Next we are create action using ethna

Flow of creation & processing: Action, View, Template

Prior to creating a real application, we are going to add action, view and template to our miniblog.

Create action, view, template

Next we are going to create action hello, which will be accessible from the following url

```
http://miniblog.myhost/?action_hello=true
```

There are 3 things to do

- `add-action hello`
- `add-view hello`
- `add-template hello`

Let's jump to the command line and do the above 3 steps.

```
$ ethna add-action hello
file generated [/var/www/miniblog/skel/skel.action.php -> /var/www/miniblog/app/
↪action/Hello.php]
action script(s) successfully created [/var/www/miniblog/app/action/Hello.php]
```

```
$ ethna add-view hello
file generated [/var/www/miniblog/skel/skel.view.php -> /var/www/miniblog/app/view/
↪Hello.php]
view script(s) successfully created [/var/www/miniblog/app/view/Hello.php]

$ ethna add-template hello
file generated [/var/www/miniblog/skel/skel.template.tpl -> /var/www/miniblog/
↪template/ja_JP/hello.tpl]
template file(s) successfully created [/var/www/miniblog/template/ja_JP/hello.tpl]
```

Tip:

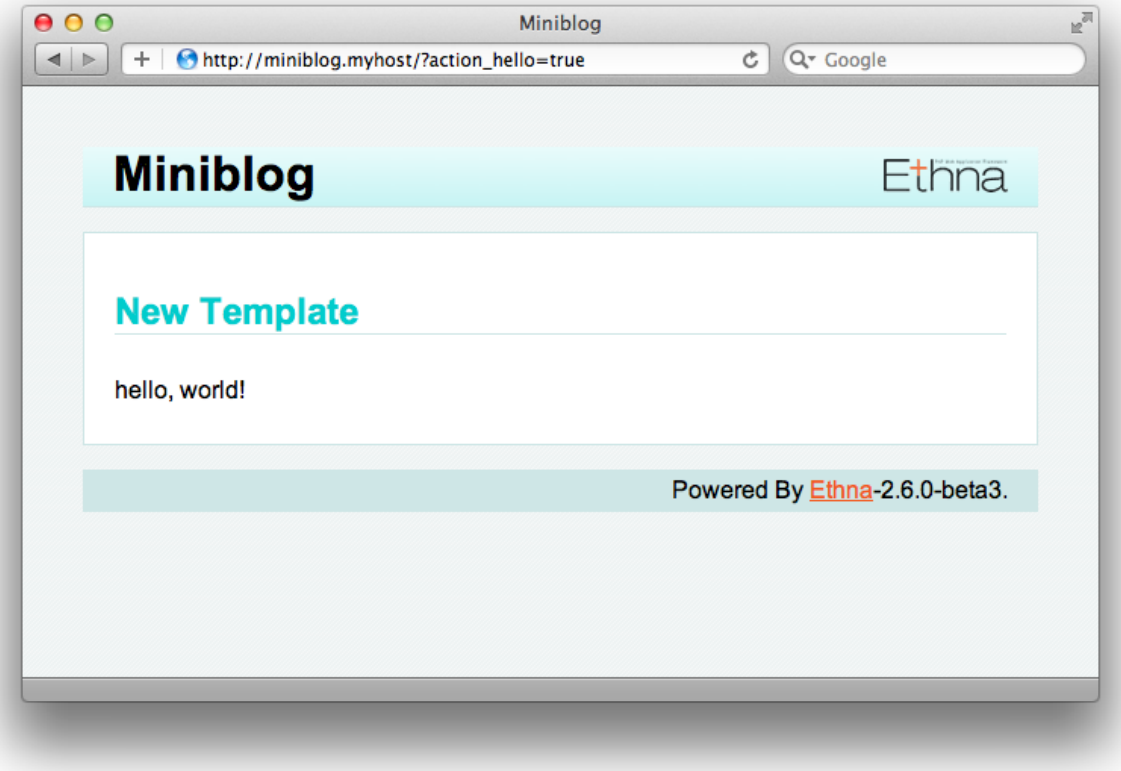
You can add a view and template in one command by adding `-t` like this

```
add-view -t hello
```

Now, the view for the added action named `hello` can be viewed in the browser with the following url

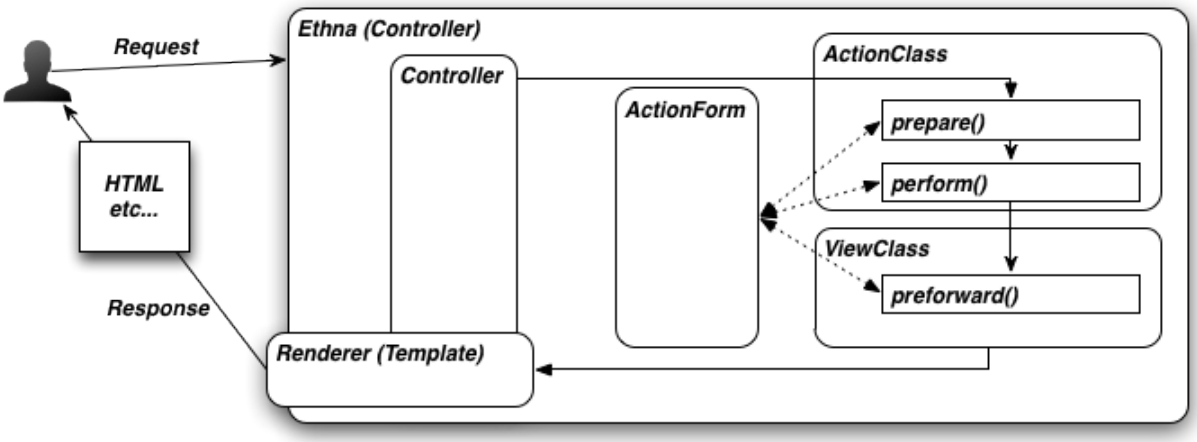
```
http://miniblog.myhost/?action_hello=true
```

The browser should display the following



Flow of processing in Ethna

This diagram illustrates the flow of processing Ethna



Action

With the command `ethna add-action hello`, `Hello.php` is created in the directory `app/action/Hello.php`. In this php file there are two classes. As a courtesy, comments are also generated, but for this tutorial I am omitting the comments. The two classes follows next.

```
<?php
class Miniblog_Form_Hello extends Miniblog_ActionForm
{
     protected $form = array(
  );
}

class Miniblog_Action_Hello extends Miniblog_ActionClass
{
    public function prepare()
    {
        return null;
    }

    public function perform()
    {
        return 'hello';
    }
}
```

`Miniblog_Form_Hello`

This is the action form class, and inherits Ethna's ActionForm class. In this class the form for POST and/or GET is defined. We will discuss more about forms later in this documentation.

`Miniblog_Action_Hello`

This class implements the Hello action. Two functions `prepare()` and `perform()` are called before the implementation transit to *view*.

View

The, view class generated by the command `ethna add-view hello`, should be in the directory `app/view/Hello.php`.

```
<?php
class Miniblog_View_Hello extends Miniblog_ViewClass
{
    public function preforward()
    {
        $message = "Hello World";
        $this-af->setApp('helloVariable',$message);
    }
}
```

This is the View class inherits from ethna's View class. View class is responsible for setting the variables that then be used in the HTML *template*. You may set your variables in the function `preforward()`, using `setApp()`.

Template

Last one is template which is created by the command `ethna add-template hello` in the directory `template/ja_JP/hello.tpl`

Lets, play with the `helloVariable` and try to print it in the template

```
<h2> New Template </h2>
<p> {$app.helloVariable} </p>
```

Implementation of Application

Lets start the implementation of creating a blog via learning following 3 things

1. Add action to handle the post, defined in a form.
2. Output the value from the form.
3. Validate the values from input form.

Add action

First is to quickly add an action named `commit` by the command `ethna add-action commit`

```
$ ethna add-action commit
file generated [/var/www/miniblog/skel/skel.action.php -> /var/www/miniblog/app/
↪action/Commit.php]
action script(s) successfully created [/var/www/miniblog/app/action/Commit.php]
```

As the message also shows, there should be a file generated in `app/action/Commit.php` Here, there are two classes,

1. `class Miniblog_Form_Commit extends Miniblog_ActionForm` that handles the input fields for the form
2. `class Miniblog_Action_Commit extends Miniblog_ActionClass` that performs actions upon input/submitting the form

Lets take a look at them one by one

Creating Form

And in `app/action/Commit.php` file.

```
<?php
class Miniblog_Form_Commit extends Miniblog_ActionForm
{
    protected  $form  =  array(
        'Comment'  =>  array(
            'type'  =>  VAR_TYPE_STRING,
            'form_type'  =>  FORM_TYPE_TEXTAREA,
            'name'  =>  'comment',
            'max'  =>  140,
            'required '  =>  true,
        ),
    );

}
```

Which is:

- Form named **Comments**
- The value type is string
- Form type is TEXTAREA
- 140 Characters
- Required Item

And the second class

```
<?php
class Miniblog_Action_Commit extends Miniblog_ActionClass
{
    public function prepare()
    {
        return null;
    }
    public function perform()
    {
        return 'index';
    }
}
```

We will talk about this second class `Miniblog_ActionClass` soon, but first lets create the form to be shown in the browser.

Display Form

Lets display the form in our `template/ja_JP/index.tpl` file

```
<h2>Post</h2>

{form name="form_comment" ethna_action="commit"}
  Post Content:
  {form_input name="comment"}
```

```
  {form_submit}
 {/form}
```

`ethna_action` defines the action to be called. It is similar to doing `action="commit.php"` in plain html form. Similarity other values are self explanatory.

Upon visiting the browser, a page similar to following should appear.

Showing the POST content

```
<h2>Comment</h2>
{$form.comment}
```

This way you should be able to see the post content after the submit button from the text area. `$form` is used to access the variables defined in the ActionForm class.

Display Validation and Error content of POST

Here we explain about the `Miniblog_ActionClass` where the validations to the form can be performed.

```
public function prepare()
{
   if ($this->af->validate() > 0) {
       return 'index';
   }
```

```
    return null;
}
```

Firstly, `prepare()` is called, here the necessary preparations can be done, like assigning values to instances, performing checks etc.

`$this->af` is the action form object, while `validate()` executes the method that performs validations on the input value. The point here is that if there is a trouble with the validation, the next method `prepare()` will not be called and instead the `index` will be returned (rendered).

Displaying Errors

Error content for the form are handled by the variable `$error`.

```
<h2>Posting Comment</h2>
{if count($errors) > 0}
There was an error !
{/if}

{form name="form_comment" ethna_action="commit"}

 Post Content:<br />
 {message name="comment"}<br />
 {form_input name="comment"}

 {form_submit}

{/form}
```

Where the output of the above code would be the following, if the characters in the text area are more than 140 or 0.

Connection to Database

In this section, we talk about the following things

1. Setting Database
2. SELECT, INSERT, UPDATE i.e. basic stuff

Setting Database

Setting database is pretty straight forward with ethna. Settings are defined in `etc/miniblog-ini.php`. To connect to the database uncomment the `dsn` like the following.

```
<?php

$config = array(
    // site
    'url' => '',

    // debug
    // (to enable ethna_info and ethna_unittest, turn this true)
    'debug' => true,

    // db
    // sample-1: single db
```

```
    // 'dsn' => 'mysql://user:password@server/database',
    // ...
```

Replace the

- `user` with your `DB username`
- `password` with your `password`
- `server` with `localhost` or `127.0.0.1`
- `database` with your `database_name`

Save it and good to go

Connecting to Database in Action Class

The following connects to the database in `app/action/Hello.php`

```
function perform()
{
    $db = & $this->backend->getDB();
    $db->db->setFetchMode(DB_FETCHMODE_OBJ);
}
```

SELECT, INSERT, UPDATE with `$db`

SELECT

```
$sql = "SELECT * FROM users where id > ?";
$params = array("1");
$stmt = & $db->db->prepare($sql);
$res = & $db->db->execute($stmt,$params);

echo "<pre>";
$i = 0;
while ($data[$i] = $res->fetchRow()) {
   if ($i){
       var_dump($data);
   }
   $i++;
}
echo "</pre>";
```

INSERT

```
$sql = "INSERT INTO users (username, email, password, osid, gameid,signup) VALUES (?,?
↪,?,?,?,now())";
$params = array(
     "kevin",
     "dummy@gmail.com",
      sha1("foo"),
      "1111",
      "gamers"
      );
$stmt = & $db->db->prepare($sql);
$res = & $db->db->execute($stmt,$params);
```

```
//Note that this message only exists in case of DB error
if (method_exists($res, 'getMessage')){
    var_dump($res->getMessage());
}
```

UPDATE

```
$sql = "UPDATE puzzle SET playedby=?,solved=?,score=score +100 WHERE id=?";
$params = array("johny", '1',  "111");
$stmt = & $db->db->prepare($sql);
$stmt = & $db->db->execute($stmt,$params);
```

Tip: Check the other class methods quickly by `var_dump(get_class_methods($res))`

Session

Methods are defined in Ethna's ActionClass. We will go through them one by one.

Authenticate

```
function authenticate()
{
   if ( !$this->session->isStart() ) {
      return 'login';
   }
}
```

If the session has not started yet the user will be drawn to the `login` page

Starting Session

Lets say you have a login page and the user enters the password using the web form In that case, we can start a session as:

```
function perform()
{
    $password = $this->config->get('password');

    if ( $password == $this->af->get('password')) {
         $this->session->start();
    }
}
```

Tip:

- Session can be destroyed using `$this->session->destroy()`
- Regenerate Id `$this->session->regenerateId()`

Its All Romantic

In this section we will talk about ethna, smarty, localhost & webserver (Heroku) The goal upon this section is to set up ethna on a heroku. Ethna being lightweight we will not install it on heroku but rather just copy the required libs to the server.

Setting app on Heroku

Go to Heroku and set up your account. First step here is to set up an app on heroku, lets name it `ethna-web`. Upon creating the app, go to settings and get the **Git URL:** `git@heroku.com:ethna-web.git`

Tip: Also install the **Heroku Toolbelt** based on your operating system if you haven't done that already. https://toolbelt.heroku.com/

Back to Terminal

Now on the terminal, make a new Directory lets say `mkdir ~/Documents/Ethna-Web_Heroku` And go to that directory.

Lets quickly create a file in that directory,

```
$ echo "<?php echo 'Hello Heroku'; ?>" >index.php
```

Initialize the git

```
$ git init
$ git add .
$ git commit -m "Committing the index.php"
```

Add the Remote

Because we have already created the app named `ethna-web`. Lets just add a remote for the existing app using the terminal

```
$ heroku git:remote -a ethna-web
```

Push

Note: Note that we added `index.php` which also indicates heroku that its a PHP application

---

Test on your url (e.g. http://ethna-web.herokuapp.com/) provided by heroku for your app if `hello` is rendered on the web browser.

Getting Ethna to work

While in the Dir `~/Documents/Ethna-Web_Heroku`. Lets do the following to add a project.

```
$ ethna add-project ethnaweb
```

Now if you have installed Ethna on your local machine, we need to copy the libraries to heroku server.

First copy,

```
$ cp -a ~/pear/share/pear/Smarty ethnaweb/app/
$ cp -a ~/pear/share/pear/Ethna ethnaweb/app/
```

Tip: You may also want to copy the DB files

```
$ cp -a ~/pear/share/pear/DB ethnaweb/app/
```

All Done

Push the changes to heroku's repo

```
 $ git add .
 $ git commit -m "pushing ethna"
 $ git push heroku master

Ethna should render the welcome page on the url http://ethna-web.herokuapp.com/
↪ethnaweb/www/
```

Setting up Database

Setting the database (DB) requires following 2 steps:

1. Create a DB on Heroku
2. Set it up in Ethna

To create a DB on Heroku, I am going to use the free one **Heroku Postgres**. Note that the previous tutorial about DB uses Ethna's Backend to call a query to MySQL, but we are not restricted to just using Ethna's, misc DB can also be easily integrated.

Getting Heroku username & password

```
$ heroku config | grep HEROKU_POSTGRESQL

  ...HEROKU_POSTGRESQL_MAROON_URL: postgres://himvd

$ heroku pg:credentials MAROON
```

The above will output the username & password like the following

```
"dbname=abcdefg host=****.amazonaws.com port=5432 user=**** password=****␣
↪sslmode=require"
```

Setting PDO in Ethna

Setup the username and password in `etc/ethnaweb-ini.php`

```
$config = array(
 // site
 'url' => '',

 // debug
 // (to enable ethna_info and ethna_unittest, turn this true)
 'debug' => false,

 // db
```

```
 // sample-1: single db
 // 'dsn' => 'mysql://user:password@server/database',
 //    'dsn' => 'dbname=d5thfeu7cb8dms host=ec2-54-227-252-82.compute-1.amazonaws.com␣
↪port=5432 user=himvdmapqkjhav password=zGN3cprl66dNc1Qh-HzEsTwez7 sslmode=require',
 'pghost'    => '***82.compute-1.amazonaws.com',
 'pgdbname'  => '*****',
 'pguser'    => '*****',
 'pgpassword'=> '*****',
```

```
<?php

 function prepare()
 {
   //Access the config array
   $host =  $this->config->get('pghost');
   $dbname = $this->config->get('pgdbname');

   $user = $this->config->get('pguser');
   $pass = $this->config->get('pgpassword');
   //  $dbh = new PDO('pgsql:host=localhost;dbname=[YOUR_DATABASE_NAME]');
   //  $db = new PDO('pgsql:host='.$host.';'.'dbname='.$dbname.';user='.$user.';
↪password='.$pass);
   $db = new PDO('pgsql:host='.$host.';'.'dbname='.$dbname, $user, $pass);
```

All Done ! Create the table in heroku using terminal and do the queries from ethna

Tip: CLI interface to connect to heroku's postgresql

```
$ psql -h HOSTNAME -U USERNAME DBNAME
```

Config `etc/***-ini.php`

The default location of the configuration file is in `YOURAPP/etc/YOURAPP-ini.php` The configurations are defined in `$config`

We already discussed previously when setting up the Database like the following

```
<?php
$config = array(
 'debug' => false,
 'dsn'   => 'mysql://user:pass@unix+localhost/dbname',
);
```

Now, to access the config items in ActionClass of your app you can call using the following

```
<?php
class Sample_Action_Index extends Ethna_ActionClass
{
    function prepare()
    {
        return null;
    }
```

```
    function perform()
    {
        //Will get you the settings for ``dsn``
        $dsn = $this->config->get('dsn');
    }
}
```

Similarity we can also define the memcached configuration in the same file as

```
<?php
$config = array(
// sample-1: single (or default) memcache
'memcache_host' => 'localhost',
'memcache_port' => 11211,
'memcache_use_pconnect' => false, //persistent connection
'memcache_retry' => 3,
'memcache_timeout' => 3,
 );
```

It is also possible to define more than one configs for memcache as

```
<?php
$config = array(
// sample-2: multiple memcache servers (distributing w/ namespace and ids)
'memcache' => array(
     'namespace1' => array(
         0 => array(
             'memcache_host' => 'cache1.example.com',
             'memcache_port' => 11211,
         ),
         1 => array(
             'memcache_host' => 'cache2.example.com',
             'memcache_port' => 11211,
         ),
     ),
  ),
 );
```

Writing Logs

In the configuration file e.g. `appname/etc/appname-ini.php`

Add the following lines for log in config

```
<?php
$config = array(
    'url' => '',
    'debug' => false,


    'log_facility'          => 'default',
    'log_level'             => 'warning',
    'log_option'            => 'pid,function,pos',
    'log_filter_do'         => '',
    'log_filter_ignore'     => 'Undefined index.*%%.*tpl',
```

```
    'session' => array(
        'handler'                => 'files',
        'path'                   => '../tmp', //THIS IS IF YOU WANT TO SET THE SESSION␣
↪FILES PATH
        'check_remote_addr'      => true,
    ),

 'log' => array(
     'file' => array(
         'level'           => 'notice',
         'option'          => 'pid,function,pos',
         //'filter_do'     => '',
         //'filter_ignore' => 'Undefined index.*%%.*tpl',
         'file'            => '../tmp/error.log', //Complete path of the error file
  //     'dir'             => '/tmp/', //DONOT MENTION DIR here
         'mode'            => 666, //SET THE MODE TO 666 for writable and NOT 777
         ),
     ),
  );
```

Now to write to a log

```
<?php
function prepare()
{

    $logger = $this->backend->getLogger();
    $logger->log(LOG_NOTICE, "Testing Dir. File Should be Created in appname/tmp/
↪error.log");

    //ALSO a session file sess_**** should also be created in appname/tmp/sess_*****
    $this->session->start();

}
```

Memcached

Step 1: Set up `your_ethna_project/etc/php.ini`

```
<?php
'memcache_host' => 'localhost',
'memcache_port' => 11211,
'memcache_use_pconnect' => false,
'memcache_retry' => 3,
'memcache_timeout' => 3,
```

Step 2: Creating Memcache object in Class

```
<?php
class Php_Action_SomeAction extends Php_ActionClass
            public function __construct($backend)
    {
            $this->cache =& $backend->plugin->getPlugin('Cachemanager', 'Memcache');
            $this->cache->setNamespace("some_action");
```

```
            parent::__construct($backend);
    }
```

In the above code, you can set `setNamespace` for your memcache. In the sense it adds a prefix to all the cache keys that will be set by `$this->cache` in this class. Its better that ways if you are going to be dealing thousands of keys on a large system.

Step 3: After Memcache Object has been set, then you are able to make your own methods within the class to store data.

```
<?php
private function setCache($urls, $thumbs)
{
     $data = array('urls' => $urls, 'thumbs' => $thumbs);
     $this->cache->set($this->member, $data, 3600); // 24 hours expiration
}

/**
 * Cache key from Member Name e.g. Johny
 * @return array cache value from key
 */
private function getCache()
{
          $cache = $this->cache->get($this->member); //e.g Johny
     if (PEAR::isError($cache)) {
          return null;
     } else {
          return $cache;
     }
 }
```

Step 4: Verifying ?

```
[bash] memcached-tool localhost:11211 dump|grep Johny
       add some_action::Johny 3
```

Tip: In the above example of getCache, I have used `PEAR::isError`. However, if you look closely in your `usr/share/pear/Ethna/class/Plugin/Cachemanager/Ethna_Plugin_Cachemanager_Memcache.php`, you may find that it is throwing Ethna:: Error instead of PEAR. Its better to know anyways that if you wanna change that just edit every where it appears. Example:

```
146     function get($key, $lifetime = null, $namespace = null)
147     {
148         $this->_getMemcache($key, $namespace);
149         if ($this->memcache == null) {
150             return PEAR::raiseError('memcache server not available', E_CACHE_NO_
↪VALUE);
```

Internationalization

http://www.ethna.jp/ethna-document-dev_guide-i18n.html

App Object

Misc References

Preventing Double POST

A neat little trick for preventing duplicate POST on form submission using Ethna's Util class static method `isDuplicatePost()`.

```
<?php
function perform()
     {
        if (Ethna_Util::isDuplicatePost()) {
           //If it is a duplicate post
           return 'regist_done';
        }
     }
```