Xhwlay Tutorial

Version 1.1

author: msakamoto-sf@users.sourceforge.net

Copyright(c) 2007 msakamoto-sf@users.sourceforge.net

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.*

Contents

2007/10/09 : Version1.0 created
2008/02/14 : Version 1.1 created

0. preface

This tutorial consist of 3 parts.

Part1: making simple login/logout page flow.

Part2: making simple wizard page flow.

Part3: making page-action mapping oriented page flow and stateless pages.

The source codes shown in this tutorial are all included in Xhwlay. see "sample" directory in Xhwlay archive.

Before you go to Part1, please read following section and setup your tutorial environment.

0.A. NOTICE : Xhwlay's version and browsers

By developer(msakamto-sf)'s mistaken, 0.9.0's sample code include "<button>" tags which doesn't work at Internet Explorer correctly. Please use any other browsers if you confirm behaviour of sample code in Xhwlay-0.9.0.

This problem was fixed in Xhwlay-0.9.1, and this document was updated too (See Part 2)

0.B. Targeted reader and Ready for tutorial

Are you familiar with Apache and PHP Web application ? If you were not, you should make a trip around beginner documents for PHP and Apache Web application.

This tutorial assumes you to be familiar with creating PHP web application in Apache Web Server.

$ 1st, setup Apache's VirtualHost settings.

In this tutorial, we use "http://xhwlay-tutorial/" as script's top url. So we setup VirtualHost settings like below:

```
NameVirtualHost *:80
<VirtualHost *:80>
ServerName Xhwlay-tutorial
DocumentRoot /your/Xhwlay/tutorial/path
```

```
</VirtualHost>
```

(assumed that your server has been already setup Apache and PHP.)
See also Apache's VirtualHost Documentation: http://httpd.apache.org/docs/2.0/vhosts/

$ 2nd, setup your client pc's "host" file.

Add "Xhwlay-tutorial" host entry to your client pc's host file like below line.

```
192.168.0.1 Xhwlay-tutorial   # 192.168.0.1 is your server's ip address.
```

Where's host file ? Typically host file is :

- Windows XP :
C:\WINDOWS\system32\drivers\etc\host

- UNIX, Linux :
/etc/host

Or ask your system administrators.

$ If you can't modify Apache configurations:

Don't worry. just replace "http://xhwlay-tutorial/" url to your own url.

Okay, put some html/php file to your tutorial directory, and access from your web browser.
If you can see your html/php file, settings is okay.

1. Part 1

This part describes basic usage of Xhwlay by creating simple login/logout stateful page flow.

You can obtain complete source codes and files from "sample" directory of Xhwlay archive.

(Notice : complete source includes some codes for combination with wizard application main.php (described in part2).)

1.A. Designing your page flow

1st, we create **page flow** of application.

By the way, what's the ″page″ ? Is it HTML file ? No. In Xhwlay, the ″page″ is the alias of ″state″. Okay, you'll find What ″page″ is, and How to define or use the ″page″, little by little through this tutorial.

$ Page

Now, we consider login/logout simple application's page flow outline.

- **"login" page** : This page will show login screen including "username" and "password" input form and submit button.

- **"logout" page** : This page will show logouted message. This will be shown when client request "logout".

Where to go when logined ? so, we add a page which will show "Now you are logined.".

- **"main" page** : This page will show "Now you are logined.".

Okay. Next, we define "Start" and "End" point page.

In Xhwlay, A page flow is compared to A book. A book (page flow) has one "start" page and multiple "end" pages.

Now, we define "login" page as start, and "logout" page as end.

```
″login″ page (start) ----> ″main″ page ----> ″logout″ page (end)
```

$ Event

Next we consider events for transit page to page. ″Event″ is user request which triggers page transition.

In this application, it's obvious that there are two event : "login" and "logout".

- **"login" event** : If login event occurs at "login" page, check username and password, then go to "main" page if check is okay. We name this event "onLogin".

- **"logout" event** : If logout event occurs at "main" page, then goto "logout" page. We name this event "onLogout".

```
                ″onLogin″ event      ″onLogout″ event
                      |                  |
                      |                  |
                      V                  V
″login″ page (start) ---> ″main″ page ----> ″logout″ page (end)
```

$ Guard

At "onLogin" event, we needs to check requested username and password is correct.

If correct, application transit to "main" page from "login" page. If not, application must not transit "main" page, and rollbacks to "login" page.

In Xhwlay (or any other Finite State Machine), these feature is called **"Guard"**. "Guard" is a gate keeper at transition.

Now we define **"validateLogin"** guard at "onLogin" event.

```
″login″ page (start)
 ↓
(″validateLogin″ guard) ← ″onLogin″ event
 ↓
″main″ page
 ↓← ″onLogout″ event
″logout″ page (end)
```

Okay. Now we get login/logout application page flow.

Next, scratch a Xhwlay application script and embed this page flow in it.

1.B. Scratch a Xhwlay application

Now, we make a Xhwlay application script.

1st, let's make a skeleton script. Write following code, and save it as "/your/xhwlay/tutorial/path/login.php".

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');


// {{{ requires
require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Serialize.php');
require_once('Xhwlay/Renderer/Include.php');
// }}}
// {{{ Bookmark Container and Page Flow (Story) Configurations
$bookmarkContainerParams = array(
  "dataDir" => $__base_dir.'/datas',
  "gc_probability" => 1,
  "gc_divisor" => 1,
  "gc_maxlifetime" => 30,
);
$configP = array(
  // Yet empty array.
  );


// }}}
// {{{ main scripts


$renderer =& new Xhwlay_Renderer_Include();
```

```
$config =& new Xhwlay_Config_PHPArray($configP);

$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);

echo $runner->run();
?>
```

Let's look details.

requirements:

- Xhwlay/Runner.php : Page flow control engine.
- Xhwlay/Bookmark/FileStoreContainer.php : Bookmark file store engine (described later).
- Xhwlay/Config/PHPArray.php : for defining page flow as php array.
- Xhwlay/Renderer/Include.php : Simple php script rendering engine (described later).

settings:

- $bookmarkContainerParams : Bookmark Container configuration. This will be described later.
- $configP : Page Flow configuration. This will be implemented later.

main scripts:

```
$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);
```

Create renderer instance and config instance. Config instance is initialized with empty page flow configuration array.

```
$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);
```

Create Xhwlay_Runner instance.

Next, BookmarkContainer class name and its constructor parameters are set. (Described later.)

Then, renderer and config instance are set.

```
echo $runner->run();
```

Execute page flow, retrieve output data, and echo it.

Sure, it is an empty skeleton. It doesn't do anything.

$ Embedding page flow

Let's define and embed page flow into above script. See following code:

```
$configP = array(
  "story" => array(
    "name" => "Login Example",
    "bookmark" => "on",
    ),
  "page" => array(
    "logout" => array(
      "bookmark" => "last", // indicate this page is end page.
      ),
    "main" => array(
      "event" => array(
        // Acceptable Event list.
        // key is event name, and value is guard name.
        // If guard name is null, guard is not defined and not invoked.
        "onLogout" => null,
        ),
      ),
    // "*" means start page, equals, "login" page.
    "*" => array(
      "event" => array(
        "onLogin" => "validateLogin",
        ),
```

```
      ),
    ),
  "event" => array(
    "onLogout" => array(
      "transit" => array(
        // key is string which returned by event handler,
        // value is page name which defined above "page" configuration.
        "success" => "logout",
        ),
      ),
    "onLogin" => array(
      "transit" => array(
        "success" => "main",
        ),
      ),
    ),
  "guard" => array(
    "validateLogin" => array(
      ),
    ),
  );
```

Xhwlay's default configuration engine use simply php array.

Surely, you can extend your own customized configuration engine (for ex. using yaml, xml, and databases).

Okay. page flow definition is done ... but incomplete!! "What to be done" (other words, **"Action"**)is not defined or implemented yet.

$ Page Action Skeleton

Let's define "Actions" at page, event, and guard.

1st, let's define "login" page action.

```
function demo_page_login(&$runner, $page, &$bookmark, $params) {
}
```

Yes, it is skeleton yet. Real codes are filled in later.

How to associate page action with page ?

Add "user_function" entry in "login" page definition.

```
"*" => array(
  "user_function" => "demo_page_login", // This line is added.
  "event" => array(
    "onLogin" => "validateLogin",
    ),
  ),
```

In the same way, let's write "main" and "logout" page action.

```
function demo_page_main(&$runner, $page, &$bookmark, $params) {
}
function demo_page_logout(&$runner, $page, &$bookmark, $params)
}
```

Okay, now we describes some details of arguments.

- **&$runner** : This is object reference of Xhwlay_Runner instance. You can retrieve renderer and configuration instance from this instance.

- **$page** : This is string value of current page name.

- **&$bookmark** : This is object reference of current Bookmark. You can write/read your own application datas in it. Described later.

- **$params** : This is mixed array. For example, "*" page action retrieves this parameter like following array:

```
array(
  "user_function" => "demo_page_login",
  "event" => array( "onLogin" => "validateLogin" )
)
```

You can define your application's specific configuration parameters in it.

$ Event Action Skeleton

2nd, let's define event actions(of course, these are skeleton yet).

```
function demo_event_onLogin(&$runner, $event, &$bookmark, $params) {
```

```
  return "success";
}
function demo_event_onLogout(&$runner, $event, &$bookmark, $params) {
  return "success";
}
```

Return value is transit name which is defined in page flow. If return value is not defined value in page flow, Xhwlay doesn't transit page.

Now we define event action in page flow :

```
"event" => array(
  "onLogout" => array(
    "user_function" => "demo_event_onLogout",
    ,,,
    ),
  "onLogin" => array(
    "user_function" => "demo_event_onLogin",
    ...
    ),
  ),
```

Okay, now we describes some details of arguments.

- **&$runner** : This is object reference of Xhwlay_Runner instance. You can retrieve renderer and configuration instance from this instance.

- **$event** : This is string value of current event name.

- **&$bookmark** : This is object reference of current Bookmark. You can write/read your own application datas in it. Described later.

- **$params** : This is mixed array. For example, "onLogout" event action retrieves php array like below:

```
array(
  "user_function" => "demo_event_onLogout",
  "transit" => array( "success" => "logout" )
)
```

You can define your application's specific configuration parameters in it.

$ Guard Action Skeleton

3rd, let's define guard actions(of course, these are skeleton yet).

```
function demo_guard_validateLogin(&$runner, $event, &$bookmark, $params) {
  return true;
}
```

If guard action returns true, Xhwlay invokes Event action. If returns false, Xhwlay doesn't invoke Event action, roll back to current page.

Now we define guard action in page flow :

```
"guard" => array(
  "validateLogin" => array(
    "user_function" => "demo_guard_validateLogin"
    ),
  ),
```

Okay, now we describes some details of arguments

- **&$runner** : This is object reference of Xhwlay_Runner instance. You can retrieve renderer and configuration instance from this instance.

- **$event** : This is string value of current event name.

- **&$bookmark** : This is object reference of current Bookmark. You can write/read your own application datas in it. Described later.

- **$params** : This is mixed array. For example, This guard action retrieves php array like below:

```
array(
  "user_function" => "demo_guard_validateLogin"
)
```

You can define your application's specific configuration parameters in it.

$ "View Name" , and preparing skeleton templates.

Actions are prepared. Now, we must prepare **"View"** to create HTML for the browser.

In Xhwlay, **The return value of "Page Action" is called "View Name"**.

"View Name" is passed to rendering engine.

"View Name" is resource identifier for rendering engine. Resource is something datas to create output data.

Resource may be HTML Template File Name, Database Key, or other else. These are depends on rendering class implementation.

(Note: some renderers which provided by Xhwlay ignore "View Name". ex : "Serialize", "VarDump" renderer)

For this tutorial, we use Xhwlay_Renderer_Include rendering class which includes html/php text file specified by "View Name".

Okay, now we prepare "View Name" and resources for each pages.

- "login" page : View Name is "templates/login_login.html"
- "main" page : View Name is "templates/login_main.html"
- "logout" page : View Name is "templates/login_logout.html"

Let's implement these view name and template files.

$$ Modify Page Action

Add "return <View Name>" to page actions.

In Xhwlay_Renderer_Include, view name is relative path against application script.

Now we make "templates" directory, relative path becomes "templates/...".

```
.../login.php
    templates/
                xxxx.html
```

Okay, add "return" code to actions.

```
function demo_page_login(&$runner, $page, &$bookmark, $params) {
  return "templates/login_login.html";
}
function demo_page_main(&$runner, $page, &$bookmark, $params) {
  return "templates/login_main.html";
}
function demo_page_logout(&$runner, $page, &$bookmark, $params) {
  return "templates/login_logout.html";
}
```

$$ Prepare skeleton HTML template files

Now we create directory "templates" in script's directory, make 3 html files in it.
See complete source codes and files in "sample" directory of Xhwlay archive.

Okay. Now, we prepared executing page flows. Start your web browser and access following url:

```
http://xhwlay-tutorial/login.php
```

You'll confirm "templates/login_login.html" is displayed.

Now we confirmed Xhwlay execution, but page/event/guard action implementation is incomplete yet.
And we can't invoke event yet. Next, we implement some code pieces about invoking events.

1.C. Build a Xhwlay application

First, we must decide "How to hold **'Bookmark Container ID'**?".
Okay, try to describe summary about Xhwlay's "Bookmark" and "Bookmark Container" concepts.

$ "Bookmark"

"Bookmark" is compared to real world bookmark exactly.
A "Bookmark" is associated a book, in Xhwlay, called **"Story"**.
"Story" equals page flow.
A "Bookmark" holds a page name at which currently am I.
Optionally, A "Bookmark" can holds some user datas.

Okay, then how we and Xhwlay identifies bookmarks ? Is there any Id associated with it ?
Yes, **"Bookmark Container Id" is identification of bookmarks**.

$ "Bookmark Container" and "Bookmark Container Id (BCID)"

At our first access to Xhwlay application, Xhwlay publishes a new "Bookmark Container Id (BCID)".

A BCID is associated with a **"Bookmark Container"**.

A "Bookmark Container" holds and manages some bookmarks.

Remember, a bookmark is associated a "Story" (page flow).

Yes, **A BCID is associated with multiple stories. We execute multiple page flows by one BCID at same time.**

(Notice: Xhwlay CAN'T associate multiple bookmarks with same page flow by one BCID.)

Okay, Now we knows:

- A bookmark holds "Where am I ?" and some user datas.

- A Bookmark Container holds multiple bookmarks. (But bookmarks must be different page flow for each other.)

- A BCID is associated a bookmark container.

Humm.... then, how we can holds a BCID and how to tell a BCID to Xhwlay ? And more, How to tell "Event has occurs" to Xhwlay ?

$ "HOOK" and "setup", "terminate" hook

"Implement as you like" is one of Xhwlay's base concept. Many features which constructs one web application are not in Xhwlay. Those are implemented by developers as they like.

How to tell a BCID to Xhwlay ? How to tell event occurrs to Xhwlay ? Yes, Xhwlay doesn't provide any typical means about these questions.

Why not ?

Because Xhwlay and developers of Xhwlay never imagine completely where these informations come from. URL Query ? POST parameter ? XML-RPC parameters ? or any other special format which developers of Xhwlay don't know ? ...e.t.c.

So, developers of Xhwlay doesn't (can't) provide concrete implementation, but provide **"HOOK"** point for users to implement as they like. This "HOOK" point is called **"setup" and "terminate" hook**.

You may be familiar with "HOOK" (other word, "Plugin Point", "Plugin", ...e.t.c.) concept if you already have used to be any other frameworks.

Okay, "A practical source code is worth a thousand words", let's see how to use Xhwlay's HOOK.

Edit login.php like below:

```
....
$config =& new Xhwlay_Config_PHPArray($configP);
// append start
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");

$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");
// append end
$runner =& new Xhwlay_Runner();
...
// add this function
function demo_hook_setup_session($hook, &$runner) {
  session_start(); // starts the session.

  $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
  $event = isset($_REQUEST['_event_']) ? $_REQUEST['_event_'] : "";

  Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
  Xhwlay_Var::set(XHWLAY_VAR_KEY_EVENT, $event);
}
function demo_hook_terminate($hook, &$runner) {
  $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
  $sid = session_id();
  if (!empty($sid)) {
    $_SESSION['bcid'] = $bcid;
  }
}
```

(Xhwlay_Var is static class which provide simple getter/setter method. This class is alternative of global variable in Xhwlay.)

Xhwlay_Hook::getInstance() returns Xhwlay_Hook instance which holds a stack of callbacks associated with specified hook name (in this case, "setup" and "terminate").

Now we prepared "demo_hook_setup_session" function as XHWLAY_RUNNER_HOOK_SETUP hook callback and "demo_hook_terminate" function as XHWLAY_RUNNER_HOOK_TERMINATE hook. So we call pushCallback() method with function name.

"setup" hook is executed before Xhwlay starts main routine. When Xhwlay starts main routine, Xhwlay retrieves event and BCID information from Xhwlay_Var. So, If you notify event occurrence or requested BCID, you should create hook functions and setup Xhwlay_Vars at "setup" hook.

"terminate" hook is executed after Xhwlay ends main routine. When Xhwlay ends main routine, Xhwlay sets actual BCID at XHWLAY_VAR_KEY_BCID of Xhwlay_Var. So, you can use "terminate" hook if you want to retrieve BCID certainly.

Above implementation, BCID is retrieved in "terminate" hook and saved into $_SESSION. This process is executed with each page request. In "setup" hook, BCID is set to Xhwlay_Var from $_SESSION. Event occurrence retrieves from "_event_" parameter at POST or GET method.

Okay, now we can pass event and BCID information to Xhwlay. Next, edit 1st page html. Try to show current BCID.

```
templates/login_login.html
...
<ul>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
</ul>
...
```

How can we set BCID to html ?

Xhwlay_Renderer_Include(and other renderer classes) provides setter method to use variables in their view resource.

```
Xhwlay_Renderer_Include::set("name", $var);
```

This is not static method, so we must get instance of renderer.

You can use variables which set by using above method as $GLOBALS['template']["name"].

And, where to call this setter() methods? Where can we retrieve renderer instance ?

Event, Page, Guard actions retrieve Xhwlay_Runner instance as 1st argument, "&$runner".

You can retrieve renderer instance reference like below:

```
$renderer =& $runner->getRenderer();
```

In this tutorial example, we call this method in page action.

Edit demo_page_login() like below:

```
function demo_page_login(&$runner, &$bookmark, $event, $params) {
  $renderer = $runner->getRenderer();
  $renderer->set('BCID', Xhwlay_Var::get(BCID));
  ...
}
```

Okay, Now we can access BCID in html by $GLOBALS['template']['bcid'].

In Xhwlay_Renderer_Include rendering engine, we can use assigned variable through $GLOBALS['template'] global variables.

Let's insert above 2 lines to other 2 files (main.html, logout.html) and 2 functions.

1.D. Implement Login/Logout and other features

Finally, let's implement login/logout mechanism and other remaining features.

$ Login/Logout

Okay, let's implement login/logout feature.

$$ Implementing "demo_guard_validateLogin" guard action

1st, we must implement "demo_guard_validateLogin" guard action. We've prepared skeleton code, now let's write actual code.

Here is full-actual demo_guard_validateLogin() code:

```
function demo_guard_validateLogin(&$runner, $event, &$bookmark, $params) {
  $_user_name = @$_REQUEST['user_name'];
  $_password = @$_REQUEST['password'];
  if (empty($_user_name) || empty($_password)) {
    return false;
  }
```

```
  // Store into Bookmark user data area.
  $bookmark->set("user_name", $_user_name);
  $bookmark->set("password", $_password);

  $sid_old = session_id(); // save old sid.
  session_regenerate_id(); // generate new sid.
  $sid_new = session_id(); // save it.
  session_id($sid_old);    // now, set current as saved old sid.
  session_destroy();       // destroy current (equals old sid).
  session_id($sid_new);    // re-set current as new sid.
  session_start();         // re-start session.

  return true;
}
```

Okay, let's see details one by one.

1st, retrieving user name and password from $_REQUEST variables.

2nd, if user name or password is empty, returns false.

3rd, saves user name and password in Bookmark data.

4th, regenerates session id for preventing session fixation attacks.

Remember, guard action returns false, Xhwlay doesn't transit page. Just invoke "current" - bookmarked page action. Above code, remember this guard is invoked when "onLogin" event occurrs. At this time, we are in "login" page.

When guard action returns false (equals user name or password is empty), Xhwlay doesn't invoke "onLogin" event action, just only execute "login" page action. And our bookmark remain at "login" page.

If guard returns true, Xhwlay invokes "onLogin" event action, retrieves transit name ("success"), transit to page which mapped with transit name, and bookmarked page is updated to new page name, finally execute new page action,

This is tutorial example, so we don't need strict authorization. So we only check user name and password is not empty. Actually, you will implement DB/LDAP access and validation process.

Next attention, user name and password is saved in bookmark. these are referred in "onLogin" event action. This is a demonstration of how to use bookmark user datas.

After saving login information in bookmark, now this request is authenticated, so we regenerates

session id for preventing session fixation attacks.

Finally returns true. If guard action returns true, Xhwlay invokes "onLogin" event action.

$$ Implementing "demo_event_onLogin" event action

Now we implement actual code of "onLogin" event action.

Here is full-actual demo_event_onLogin() code:

```
function demo_event_onLogin(&$runner, $event, &$bookmark, $params) {
  // These are stored in demo_guard_validateLogin().
  $user_name = $bookmark->get("user_name");
  $password =$bookmark->get("password");

  // demo codes.
  $user_id = md5( $user_name . $password );
  $bookmark->remove("user_name");
  $bookmark->remove("password");

  // stores user_id into a session variable.
  $_SESSION['user_id'] = $user_id;

  // demo counter.
  $_SESSION['count'] = 0;

  return "success";
}
```

1st, retrieves user name and password from bookmark.(see above section)

2nd, it's demo code. calculates "user_id" by md5(), and stores into $_SESSION['user_id'].

User name and password are no longer needed, So these are removed by calling $bookmark->remove().

And, $_SESSION['count'] is initialized to zero. This variable is used later.

$$ Implementing "function demo_event_onLogout" event action

Finally, we implement actual code of "onLogout" event action.
Here is full-actual demo_event_onLogin() code:

```
function demo_event_onLogout(&$runner, $event, &$bookmark, $params) {
  session_destroy();
  return "success";
}
```

Simply, calls session_destroy().

$ Implement remaining features

Okay, now, we are ready for implementing remaining features.

$$ demo_page_main()

At "main" page action, now we implement simple session counter.
Here is full-actual demo_event_onLogin() code:

```
function demo_page_main(&$runner, $page, &$bookmark, $params) {
  $user_id = $_SESSION['user_id'];
  // count up demo.
  $count = $_SESSION['count'];
  $count++;
  $_SESSION['count'] = $count;

  $renderer =& $runner->getRenderer();

  $renderer->set('page', 'main');
  $renderer->set('user_id', $user_id);
  $renderer->set('count', $count);
  $renderer->set('bcid', $bookmark->getContainerId());
  return "templates/login_main.html";
```

```
}
```

1st, we retrieve "user_id" from $_SESSION, and pass it to renderer.
2nd, retrieve "count" from $_SESSION, increment it, update $_SESSION, and pass it to renderer.

$$ templates/login_main.html

Last, let's display value of session counter in main view.
Here is full-actual templates/login_main.html code:

```
<html>
<body>
<ul>
<li>Page Position : <?php echo $GLOBALS['template']['page']; ?></li>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
<li>Session Count : <?php echo $GLOBALS['template']['count']; ?></li>
<li>User ID : <?php echo $GLOBALS['template']['user_id']; ?></li>
</ul>

<hr>
<h3><a href="main.php">Go to Wizard Example.</a></h3>
<hr>
<form action="" method="POST">
<button type="submit" name="_event_" value="onLogout">Logout</button>
</form>
<a href="?_event_=onLogout">Logout</a>

</body>
</html>
```

$ Access Test

Okay, now we are ready for accessing this applications and login/logout action.
1st, access http://xhwlay-tutorial/login.php.

Okay, we are start of page flow, so 1st login html page is displayed.

Input user name and password as you like (there's no authenticate features in this tutorial, so, anyone can login/logout:)) and press "login" button.

Now, we confirms "login" event occurrs and demo_event_login() was invoked, user id (md5 of user name and password) was stored in session, and assigned to renderer, finally, main.html is displayed.

Okay, now we push "reload" button of browser. you may see "Re-POST Confirmation "Message Box. push "OK".

Humm...yes, now we are at "main" page. So, reloading this page lead to displaying main view. And we can confirm that session counter is counted up every time we reload this page.

Next, click "logout" button. We can confirm "logout" html.

Then reload logouted page, now we can get "login" page again. Because logout page is the end of this story, bookmark is destroyed.

So, When we reload this page, Xhwlay create new bookmark about this story and invoke start page action.

1.E. Next tutorial

Now, we learned basics of building Xhwlay web application.

Next, we try to build a simple wizard application which has 5 pages for accepting and confirming user input.

2. Part 2

This part describes simple wizard application using Xhwlay.

You can obtain complete source codes and files from "sample" directory of Xhwlay archive.

2.A. Designing your page flow

In this tutorial, we try to make simple wizard application which has 5 pages.

This wizard provides typical application form. Users input their name, email, zip-code, address, telephone number, age, hobby. Input form is separated to multiple pages. This application also has two pages, one for users to confirm their all input datas, two for application to action something and show result of action (ex. sending mail, database registration).

Here, we design pages like below:

- page0 : Start page. Name, E-Mail input form.
- page1 : Zip Code, Address, Telephone number input form.
- page2 : Age, Hobby input form.
- page3 : Confirmation page.
- page4 : End page. Something was done.

Page 0 has a button to page1.

Page 1 - 3 has two buttons to previous/next page.

Page 4 has a button to back to page0.

Next, we consider how event does this page flow needs.

In consideration, all events can aggregate to one event. All events between page0 and page1, page1 and page2, page2 and page3 excepting page3 and page4, only store requested form datas to bookmark and transit. Between page3 and page4, because of demonstration sample, we doesn't need to implement "actions". So we can reuse a same event described above.

This time, we don't implement any guards. It's not needed to demonstrate essence of Xhwlay's wizard application.

And we now decide view names.

- page0 : templates/main_page0.html

- page1 : templates/main_page1.html
- page2 : templates/main_page2.html
- page3 : templates/main_page3.html
- page4 : templates/main_page4.html

Sorry, we now skip explanation of these htmls. These contents is similar to htmls of part1, but little differenct. The differences are described following section, but detailing entire htmls is not essence of this text. Please refer actual code in Xhwlay's archive.

2.B. Scratch a Xhwlay application

Here is scratch source code : main.php.

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');


require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Include.php');

$bookmarkContainerParams = array(
  "dataDir" => $__base_dir.'/datas',
  "gc_probability" => 1,
  "gc_divisor" => 1,
  "gc_maxlifetime" => 30,
);

$configP = array(
  "story" => array(
    "name" => "Wizard Example",
```

```
    "bookmark" => "on",
    ),
  "page" => array(
    "page4" => array(
      "user_function" => "demo_page_page4",
      "bookmark" => "last",
      ),
    "page3" => array(
      "user_function" => "demo_page_page3",
      "event" => array(
        "onSubmitPage4" => null,
        "onBacktoPage2" => null,
        ),
      ),
    "page2" => array(
      "user_function" => "demo_page_page2",
      "event" => array(
        "onSubmitPage3" => null,
        "onBacktoPage1" => null,
        ),
      ),
    "page1" => array(
      "user_function" => "demo_page_page1",
      "event" => array(
        "onSubmitPage2" => null,
        "onBacktoPage0" => null,
        ),
      ),
    "*" => array(
      "user_function" => "demo_page_page0",
      "event" => array(
        "onSubmitPage1" => null,
        ),
      ),
```

```
    ),
  "event" => array(
    // go to next page
    "onSubmitPage1" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page1")),
    "onSubmitPage2" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page2")),
    "onSubmitPage3" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page3")),
    "onSubmitPage4" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page4")),

    // back to previous page
    "onBacktoPage2" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page2")),
    "onBacktoPage1" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "page1")),
    "onBacktoPage0" => array(
      "user_function" => "demo_event_onSubmit",
      "transit" => array("success" => "*")),
    ),
  );

$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);

// setup "setup" hooks (executed before Xhwlay)
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
```

```
$h1->pushCallback("demo_hook_setup_session");

// setup "terminate" hooks (executed after Xhwlay)
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");

$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);

echo $runner->run();

function demo_hook_setup_session($hook, &$runner) {
  session_start();

  $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
  // get Event from request parameters.
  $event = '';
  foreach ($_REQUEST as $_k => $_v) {
    if (preg_match('/^_event_(¥w+)$/', $_k, $m)) {
      $event = $m[1];
    }
  }

  Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
  Xhwlay_Var::set(XHWLAY_VAR_KEY_EVENT, $event);
}

function demo_hook_terminate($hook, &$runner) {
  $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
  $sid = session_id();
  if (!empty($sid)) {
```

```
    $_SESSION['bcid'] = $bcid;
  }
}

function demo_page_page4(&$runner, $page, &$bookmark, $params) {
  return "templates/main_page4.html";
}

function demo_page_page3(&$runner, $page, &$bookmark, $params) {
  return "templates/main_page3.html";
}

function demo_page_page2(&$runner, $page, &$bookmark, $params) {
  return "templates/main_page2.html";
}

function demo_page_page1(&$runner, $page, &$bookmark, $params) {
  return "templates/main_page1.html";
}

function demo_page_page0(&$runner, $page, &$bookmark, $params) {
  return "templates/main_page0.html";
}

function demo_event_onSubmit(&$runner, $event, &$bookmark, $params)
{
  $vars = array(
    "name", "email", // input at "*"
    "zip", "address", "telephone", // input at "page1"
    "age", "hobby", // input at "page2"
    );

  foreach ($vars as $_k) {
    if (isset($_REQUEST[$_k])) {
```

```
        $bookmark->set($_k, $_REQUEST[$_k]);
      }
    }
    return "success";
}
?>
```

Most part of this source code is similar to login.php (part1), But demo_hook_setup_session() has little changes.

We describe detail of event action first, second describe demo_hook_setup_session()'s little changes and html differences between Part 1 and Part 2.

$ Event action of Wizard application example

In above page flow, all the event actions commonly use **"demo_event_onSubmit"**.

So we define form names as array "$vars", and loop it against $_REQUEST, retrieve value, and save it in bookmark.

And this event action only returns "success". Actual transit is switched at page flow configuration.

For example, page1 accepts "onSubmitPage2" and "onBacktoPage0" event:

```
"page1" => array(
  "user_function" => "demo_page_page1",
  "event" => array(
    "onSubmitPage2" => null,
    "onBacktoPage0" => null,
    ),
  ),
```

See "onSubmitPage2", "success" is mapped to "page2".

```
"onSubmitPage2" => array(
  "user_function" => "demo_event_onSubmit",
  "transit" => array("success" => "page2")),
```

If "onBacktoPage0", "success" is mapped to "page0", so Xhwlay transit to page0.

```
"onBacktoPage0" => array(
```

```
  "user_function" => "demo_event_onSubmit",
  "transit" => array("success" => "*")),
```

$ demo_hook_setup_session() and template html's little changes

Review demo_hook_setup_session() between Part1 and Part2 again.

Part 1(login.php)

```
function demo_hook_setup_session($hook, &$runner)
{
  ...
  // get Event from request parameters.
  $event = isset($_REQUEST['_event_']) ? $_REQUEST['_event_'] : "";
  ...
}
```

Part 2(main.php)

```
function demo_hook_setup_session($hook, &$runner)
{
  ...
  // get Event from request parameters.
  $event = '';
  foreach ($_REQUEST as $_k => $_v) {
    if (preg_match('/^_event_(¥w+)$/', $_k, $m)) {
      $event = $m[1];
    }
  }
  ...
}
```

Why these modification is needed at Part 2, main.php ? Because most pages of Part 2 has 2 buttons, "Back to previous", "Go to next" page button, so we must distinguish requests from two or more buttons by setting "name" attribute of '<input type="submit">' button "_event_[Event Name]" like following html.

templates/main_page1.html

```
...
```

```
<input type="submit" name="_event_onBacktoPage0" value="&lt;&lt; Page 0" />
<input type="submit" name="_event_onSubmitPage2" value="&gt;&gt; Page 2" />
...
```

Now, we loop requested parameters key, and if key matches "_event_[Event Name]" pattern, retrieve "[Event Name]" part, and use it.

In Xhwlay-0.9.0's sample and document uses "<button>" tags. But this was developer(msakamoto-sf)'s mistake. msakamoto-sf is foolish, didn't know multiple "<button>" tag is not supported correctrly in Internet Explorer. So, 0.9.0's sample doesn't work correct in Internet Explorer.

This problem is solved by upper codes in Xhwlay-0.9.1

2.C. Build a Xhwlay application

Now, we build remaining parts of source code.

$ Implementing page actions

For this wizard example, page action is simple. It only set user data stored in bookmark to renderer.

Here is demo_page_page0() code:

```
function demo_page_page0(&$runner, $page, &$bookmark, $params) {
  $renderer =& $runner->getRenderer();
  $renderer->set('f_name', $bookmark->get('name'));
  $renderer->set('f_email', $bookmark->get('email'));
  return "templates/main_page0.html";
}
```

Other page actions (page1 - 3) are similar to this. See actual source code included in Xhwlay archive.

$ Against "Reload" problem.

Now, we can use wizard application. Playing with it, if you click browser's "Reload" button, "RE-POST" dialog box is shown.

Okay, let's append sending "Location" header code against this dialog box.

Edit demo_event_onSubmit() like below:

```
function demo_event_onSubmit(&$runner, $event, &$bookmark, $params) {
  $vars = array(
    ...
    );
  foreach ($vars as $_k) {
    ...
  }
  // append start
  header("Location: http://Xhwlay-tutorial/main.php");
  $runner->wipeout();
  // append end
  return "success";
}
```

wipeout() forces Xhwlay not to invoke page action and view rendering. This leads outputting no HTTP body entity.

So, browsers redirects to main.php by GET method.

Let's start wizard.

- page 0 → page1 : okay.

- page 1 → page2 : okay.

- page 2 → page3 : okay.

- page 3 → page0 : ?? where's page4?

Remember page 4 is a end page. So, "onSubmitPage4", bookmark is cleared, and, send "Location" headers, then, bookmark is regenerated and finally page0 is invoked.

How to solve this problem ?

As one of many ways to solve it, we add "send_location_header" key to "onSubmitPage4" event and set its value false.

```
"onSubmitPage4" => array(
  "user_function" => "demo_event_onSubmit",
  "transit" => array("success" => "page4")),
→
"onSubmitPage4" => array(
  "user_function" => "demo_event_onSubmit",
```

```
  "send_location_header" => false,
  "transit" => array("success" => "page4")),
```

And edit demo_event_onSubmit() like below:

```
header("Location: http://Xhwlay-tutorial/main.php");
$runner->wipeout();
→
if (!isset($params['send_location_header']) ||
  $params['send_location_header'] == true) {
  header("Location: http://Xhwlay-tutorial/main.php");
  $runner->wipeout();
}
```

When "onSubmitPage4" is invoked, $params arg includes "send_location_header" entry and value is false. So, header() and wipeout() is not called, page4 is displayed.

When other events occurrs, $params arg doesn't include above entry, so header() and wipeout() is called, GET method occurrs.

2.D. Appendix : login/logout combination.

Okay, now, let's combine login.php and main.php. Append login/logout feature to main.php by redirecting to login.php when user_is is not found in $_SESSION vars.

Append "demo_hook_setup_auth()" function to main.php :

```
function demo_hook_setup_auth($hook, &$runner) {
  if (!isset($_SESSION['user_id'])) {
    // If not logined yet, send "Location" header and ...
    header("Location: http://Xhwlay-tutorial/login.php");
    // and, restrain continuous page action invoking, terminate.
    $runner->wipeout();
  }
}
```

Then, where to call above function ? We decide to call it just after session starting.

Yes, Add this function to "setup" hook.

```
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");
$h1->pushCallback("demo_hook_setup_auth"); // add this line
```

Okay, we get login/logout feature.

2.E. Next tutorial

Now we can build our own Xhwlay application. But there are some tips and features of Xhwlay which are not described yet. For example, How to call class method as action ? What happens when we set "bookmark" in story configuration as "false" ?

Part 3 include these topics. And, demonstrate that you can also use Xhwlay as "action-page mapping" old-type page controller.

3. Part 3

This part describes remaining feature of Xhwlay.

Xhwlay provides not only event driven oriented stateful page flow, but also typical "action-page mapping" page controller. But remember it, Xhwlay is mainly aimed to build event driven page flow application. "Action-page mapping" feature which described later in this tutorial is merely optional feature in Xhwlay.

You can obtain complete source codes and files from "sample" directory of Xhwlay archive.

3.A. Typical mechanism of "Action-page mapping" application

First, let's review typical mechanism of "Action-page mapping" application in PHP.

It is simple.

1st, an application has one entry point php file (like index.php).

2nd, a user request above entry point with an "action" parameter.

3rd, a requested entry point invokes controller logic and get "mapping" which associates "action" with actual class.

4th, the controller loads actual class and invokes method which name is decided by controller.

5th, next, controller invokes "View" component (like plain HTML/PHP file, Smarty or else other template engine.).

6th, a user get response from controller.

The main concept of "Action-page mapping" is "mapping" of "action" and actual class. What action should be invoked depends on user's request. So, this mechanism can be called "stateless".

Xhwlay provides unique feature about it. Yes, "Bookmark" is that. Xhwlay can store, save, and load "In which page am I ?".

You can use "Action-page mapping" in "stateful" by using Xhwlay.

Surely, you can also build "stateless" application by Xhwlay.

3.B. Designing your page flow

You can design page flow in same way as you do at tutorial Part 1, 2.

Some differences is here:

- You don't have to consider about "Event" actions.

- You must substitute "Barrier" actions for "Guard".

- You can limit page transition which should be transit from current page.

Okay, let's design page flow for this tutorial.

We now consider 3 pages as following:

- default : default page. from this page, user can transit only to page1.

- page1 : from this page, user can transit only to page0 and page2. When user transits to page2, "Barrier" action invokes. This "Barrier" action checks whether the value of "barrier" request is "pass" or not.

- page2 : from this page, user can transit only to page1 and page3.

- page3 : end page.

To simplify demonstration, we don't do anything special in each page actions. Each page actions only assign current page name to renderer instance.

Each page actions share a template file which only show which page am I and some other informations.

We now decide view names.

- all pages : templates/pages_default.html

Sorry, we now skip explanation of these htmls. These contents is similar to htmls of part1, and describing details of htmls is not essence of this text.

Please refer actual code in Xhwlay's archive.

3.C. Scratch a Xhwlay application

How to defining "action-page mapping" page flow in Xhwlay ?

Okay, here's the answer : pages.php.

```
<?php
$__base_dir = dirname(__FILE__);
$__include_path = ini_get("include_path");
ini_set("include_path", realpath($__base_dir . '/../') . PATH_SEPARATOR .
$__include_path);
session_save_path($__base_dir . '/sess/');
```

```
require_once('Xhwlay/Runner.php');
require_once('Xhwlay/Bookmark/FileStoreContainer.php');
require_once('Xhwlay/Config/PHPArray.php');
require_once('Xhwlay/Renderer/Include.php');

$bookmarkContainerParams = array(
  "dataDir" => $__base_dir.'/datas',
  "gc_probability" => 1,
  "gc_divisor" => 1,
  "gc_maxlifetime" => 30,
);

$configP = array(
  "story" => array(
    "name" => "Page Oriented Example",
    "bookmark" => "on",
    ),
  "page" => array(
    "page3" => array(
      "user_function" => "demo_page_userfunc",
      "bookmark" => "last",
      ),
    "page2" => array(
      "user_function" => "demo_page_userfunc",
      "next" => array(
        "page3" => null,
        "page1" => null,
        ),
      ),
    "page1" => array(
      "user_function" => "demo_page_userfunc",
      "next" => array(
        "page2" => "barrier_sample",
```

```
        "page0" => null,
        ),
      ),
    "*" => array(
      "user_function" => "demo_page_userfunc",
      "next" => array(
        "page1" => null,
        ),
      ),
    ),
  "barrier" => array(
    "barrier_sample" => array(
      "user_function" => "demo_barrier",
      ),
    ),
  );

$renderer =& new Xhwlay_Renderer_Include();
$config =& new Xhwlay_Config_PHPArray($configP);

// setup "setup" hooks (executed before Xhwlay)
$h1 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_SETUP);
$h1->pushCallback("demo_hook_setup_session");

// setup "terminate" hooks (executed after Xhwlay)
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");

$runner =& new Xhwlay_Runner();
$runner->setBookmarkContainerClassName("Xhwlay_Bookmark_FileStoreContainer");
$runner->setBookmarkContainerParams($bookmarkContainerParams);
$runner->setConfig($config);
$runner->setRenderer($renderer);
```

```
echo $runner->run();

function demo_hook_setup_session($hook, &$runner) {
  session_start();

  // get BCID from session variables.
  $bcid = isset($_SESSION['bcid']) ? $_SESSION['bcid'] : "";
  // get Page from request parameters.
  $page = isset($_REQUEST['_page_']) ? $_REQUEST['_page_'] : "";

  Xhwlay_Var::set(XHWLAY_VAR_KEY_BCID, $bcid);
  Xhwlay_Var::set(XHWLAY_VAR_KEY_PAGE, $page);
}

function demo_hook_terminate($hook, &$runner) {
  $bcid = Xhwlay_Var::get(XHWLAY_VAR_KEY_BCID);
  $sid = session_id();
  if (!empty($sid)) {
    $_SESSION['bcid'] = $bcid;
  }
}

function demo_page_userfunc(&$runner, $page, &$bookmark, $params) {
  $renderer =& $runner->getRenderer();
  $renderer->set('page', $page);
  $config =& $runner->getConfig();
  $bm = $config->needsBookmark() ? "on" : "off";
  $renderer->set('bookmark', $bm);
  if ($config->needsBookmark()) {
    $renderer->set('bcid', $bookmark->getContainerId());
  } else {
    $renderer->set('bcid', '(none)');
  }
```

```
  return "templates/pages_default.html";
}


/**
 * Barrier Example
 */
function demo_barrier(&$runner, $current, $next, &$bookmark, $params) {
  return isset($_REQUEST['barrier']) && $_REQUEST['barrier'] == "pass";
}
?>
```

"demo_hook_terminate" is the same source codes as tutorial 1/2.

"demo_hook_setup_session" is different.

At tutorial1/2, "demo_hook_setup_session" retrieves and set "_event_" variable, but at this tutorial, "_page_".

"demo_page_userfunc" simply set basic informations of bookmarks.

And here's template/pages_default.html source code:

```
<html>
<body>
<h1>Page oriented flow example</h1>
<ul>
<li>Current Page : <?php echo $GLOBALS['template']['page']; ?></li>
<li>Bookmark Container ID : <?php echo $GLOBALS['template']['bcid']; ?></li>
<li>Bookmark Mode : <?php echo $GLOBALS['template']['bookmark']; ?></li>
</ul>
<ul>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>">current</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page0">default</a></
li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page1">page
1</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=block">page 2 (Barrier Blocked ... finally, page
1)</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=pass">page 2 (Barrier Pass.)</a></li>
```

```
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?_page_=page3">page
3</a></li>
</ul>
</body>
</html>
```

You can play clicking links.

$ Barrier actions

"demo_barrier" is callback function for **"Barrier"**. "Barrier" action is invoked when new page is requested.

For example, imagine we are in page1. When new request page is page1, no "Barrier" actions invoke, simply page1's action is invoked.

When new request page is "page2", Xhwlay finds out "Barrier" between page1 and page2. Now, we can find "barrier_sample" is defined, and its callback function is "demo_barrier".

```
"page1" => array(
  "next" => array(
    "page2" => "barrier_sample",
    "page0" => null,
    ),
  ),
  ...
barrier" => array(
"barrier_sample" => array(
  "user_function" => "demo_barrier",
  ),
),
```

Then Xhwlay invokes barrier action, "demo_barrier". When its return value is true, then bookmark is updated with "page 2" as current page, and page action of "page2" is invoked.

Yes, "Barrier" is similar to "Guard".

Remember pages_default.html, there are two links like below:

```
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
```

```
_page_=page2&barrier=block">page 2 (Barrier Blocked ... finally, page
1)</a></li>
<li><a href="<?php echo $_SERVER['SCRIPT_NAME']; ?>?
_page_=page2&barrier=pass">page 2 (Barrier Pass.)</a></li>
```

When we are in page 1 and we click *upper* link, "barrier" request is not "pass", so "demo_barrier" returns false, finally Xhwlay rollbacks to "page1".

When *lower* link is clicked, "barrier" request is "pass", so "demo_barrier" returns true, finally Xhwlay update current page to "page2" and the an action of "page2" is invoked.

3.D. "Bookmark-OFF" mode page flow

In both "event-driven oriented" and "action-page mapping" page flow, you can invalid all "stateful" features **by setting "off" at "bookmark" value** of page flow definition.

```
$configP = array(
  "story" => array(
    "name" => "...",
    "bookmark" => "off",
    ),
```

If it is set, following features are disabled.

- "Guard", "Event", "Barrier" actions
- Bookmark Container Storing, Saving, Loading
- BCID

And, &$bookmark argument of page action will be null. This is called **"Bookmark-OFF" mode**.

"Bookmark-OFF" results that Xhwlay works simple stateless "action-page mapping" controller.

Which page action should be invoked is decided only by page parameter in the requests.

3.E. Loading Custom Class at runtime

In above examples, we only defines various actions as **"user_function"**.

How to use custom class and method ? The way is shown below:

```
"page1" => array(
  "class" => "Tutorial_PageActions_Page1",
  "method" => "staticMethod",
```

If both "user_function" and "class", "method" are defined, which will be invoked ?

Then, Xhwlay invokes "user_function".

1st, Xhwlay find out "user_function" is valid. If valid, Xhwlay invokes "user_function".

If not valid, 2nd, Xhwlay find out "class" and "method" is valid. If valid, Xhwlay invokes "class"::"method" (not instantiated).

Do we have to write "requre()" for each actions preliminarily ?

No, we don't have to.

Is there any ways to load class files which name is determined by developer's naming rules at runtime?

Yes, the way is setting **"classload" hook** which implements our own auto-loading features.

Here's full example implementing these features:

page flow:

```
$configP = array(
  "story" => array(
    "name" => "Page Oriented Example",
    //"bookmark" => "on",
    "bookmark" => "off",
    ),
  "page" => array(
    "page3" => array(
      "class" => "InnerKlass",
      "method" => "staticMethod",
      "bookmark" => "last",
      ),
    "page2" => array(
      "class" => "Tutorial_PageActions_Page2",
      "method" => "staticMethod",
      "next" => array(
```

```
        "page3" => null,
        "page1" => null,
        ),
      ),
    "page1" => array(
      "class" => "Tutorial_PageActions_Page1",
      "method" => "staticMethod",
      "next" => array(
        "page2" => "barrier_sample",
        "page0" => null,
        ),
      ),
    "*" => array(
      "user_function" => "demo_page_userfunc",
      "next" => array(
        "page1" => null,
        ),
      ),
    ),
  "barrier" => array(
    "barrier_sample" => array(
      "user_function" => "demo_barrier",
      ),
    ),
  );
```

Insert this code:

```
class InnerKlass
{
  function staticMethod(&$runner, $page, &$bookmark, $params)
  {
    // lazy job :p
    return demo_page_userfunc($runner, $page, $bookmark, $params);
  }
```

```
}

function demo_hook_classload($hook, $params)
{
  $__basedir = dirname(__FILE__);
  if (!isset($params['class'])) {
    return;
  }
  $klass = $params['class'];
  // translate PEAR-like class name to actual file path
  $klass = strtr($klass, "_", "/");
  $file = $__basedir . "/classes/" . $klass . ".php";
  if (is_readable($file)) {
    require_once(realpath($file));
  }
}
```

Okay, we are ready for autoloading "Tutorial/PageActions/Page1.php", and Page2.php.
Here's simple implementations of these files.

Tutorial/PageActions/Page1.php :

```
class Tutorial_PageActions_Page1
{
  function staticMethod(&$runner, $page, &$bookmark, $params)
  {
    // lazy job :p
    return demo_page_userfunc($runner, $page, $bookmark, $params);
  }
}
```

Tutorial/PageActions/Page2.php :

```
class Tutorial_PageActions_Page2
{
  function staticMethod(&$runner, $page, &$bookmark, $params)
```

```
  {
    // lazy job :p
    return demo_page_userfunc($runner, $page, $bookmark, $params);
  }
}
```

Okay, finally, we push demo_hook_classload() to "classload" Hooks.

Append two lines like below:

```
$h2 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_TERMINATE);
$h2->pushCallback("demo_hook_terminate");
// add start
$h3 =& Xhwlay_Hook::getInstance(XHWLAY_RUNNER_HOOK_CLASSLOAD);
$h3->pushCallback("demo_hook_classload");
// add end

$runner =& new Xhwlay_Runner();
```

3.F. End of tutorial

Now, we can build our own Xhwlay application.

You may want to know more about Xhwlay's funny features like Xhwlay's Hook(Xhwlay_Hook) and Xhwlay_Var.

Xhwlay is small, so you can easily hack the internal Xhwlay and above funny features.

Thank you for selecting Xhwlay.

If you are native english speaker and feel anger of the author's bad english, please mail to author: msakamoto-sf at users.sourceforge.net.

Copyright(c) 2007 msakamoto-sf@users.sourceforge.net