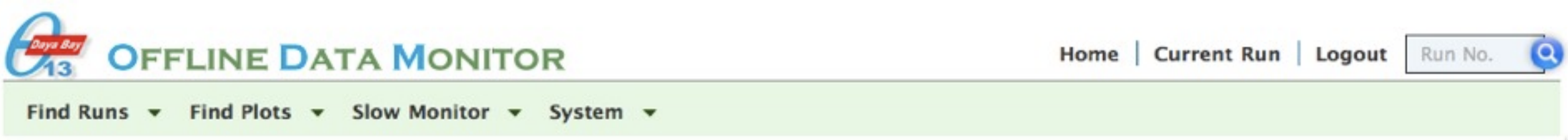


Dive Into Offline Data Monitor

An Introduction to Web Programming for Physicists

Chao Zhang, CalTech

Table of Content

This ‘tutorial’ is meant for physicists who are interested in but do not yet know where to begin with web programming.

I will use the recently developed ODM site as an example and try to make the ‘tutorial’ less technical and fun to read.

By the end of this ‘tutorial’, I hope you will be inspired enough to start writing your own web-based applications.

Introduction to Offline Data Monitor

What is and isn't ODM

- Offline Data Monitor is:
  - a central place for finding run-related information
  - a place for searching and retrieving desired list of runs or automatically generated figures
  - a place for prompt analysis in order to quickly identify existing or potential problems
  - technically, a convenient front end of various databases and flat files

- Offline Data Monitor is not:
  - a replacement of online monitors
  - a replacement of detailed offline analysis

A lot of the ODM ideas are borrowed from my KamLAND colleagues.
Hats off to them!

Why Do We Need This When We Already Have ...

- The Almighty Database

  - We users don't care about the database. We just need the information.

    Why should I care about how all the books are stored in Amazon's huge databases when all I want is the price of Harry Porter?

  - What about those cryptic database fields that nobody knows the meaning?

??

runNo	AdNo	detectorId	sourceIdA	zPositionA	sourceIdB	zPositionB	sourceIdC	zPositionC	duration	ledNumber1	ledNumber2	ledVoltage1	ledVoltage2	ledFreq	ledPulseSep	ltbMode	HomeA
6381	2	98	1	3048	1	3036	1	-78	20	4	0	-8000	0	500	0	0	1

  - Who can remember those long query commands?

```
SELECT DaqRunInfo.runNo, DaqRunConfig.intValue AS triggerType
FROM DaqRunInfo
INNER JOIN DaqRunConfig
ON DaqRunInfo.schemaVersion=DaqRunConfig.schemaVersion
AND DaqRunInfo.dataVersion=DaqRunConfig.dataVersion
AND CONCAT(DaqRunInfo.runType, 'Mode')=DaqRunConfig.objectID
WHERE DaqRunConfig.className='LTB_variableReg'
AND DaqRunConfig.name='LTB_triggerSource'
AND DaqRunInfo.runNo=4865
```

→

runNo	triggerType
4865	4354

... Why don't you just shoot me?

Why Do We Need This When We Already Have ...

- The Wonderful PHPMyAdmin
  - Doesn't the name already tell you that it's for ADMIN?
- The Master dry run list Excel files
  - It's just a piece of hand-typed-in spreadsheet and nothing more than that.
- All the raw data on disk
  - Somebody has to look at them and it takes time. Not everyone can code as fast as *Dan Dwyer* or *Xin Qian*. But you can still beat the gurus by quickly look at ODM and be the first one to find existing or potential problems.
  - We (except the nerds) like to waste time on the web instead of on the terminal, right?
- All other monitors and websites
  - Exactly, ODM cannot do everything.

OK, so what it takes to write such a web site?

Requirements for Web Developers

A Web Developer Should Always Assume ...

It only represents my point of view

- that users are dumb
  - They don’t know anything about the backend details and they don’t want to know. Your site has to provide and only provide relevant information.
- that users are smart
  - They will try anything to break your site. Your site has to be robust.
- that users are impatient
  - They have no patience to wait more than 3 seconds for information to show up. Your site has to be optimized for speed.
- that users are lazy
  - They don’t read manuals or instructions yet they want all the features to save them from even a mouse click. Your site has to be convenient, feature-rich but foolproof.
- that users are spoiled
  - They are surrounded by all the stylish websites every day. Your site at least should not look ugly.
- that users are the gods
  - You have to respect their every little requests and constantly make improvements to your site.

The Very Basics

- SQL
  - The Structured Query Language for managing data with Relational Databases. Even though nowadays many languages provide high-level libraries or ORMs which make database handling never easier, it's still a basic skill to understand the underlining raw SQL commands.
- PHP
  - The server-side scripting language to interact with the database and to talk to the client browser. (Why not java, asp.net, ruby, python, perl ...? See later discussions.)
- JavaScript
  - The client-side scripting language to relieve some of the burdens from the server, and to improve user interactivity (but do not abuse the usage).
- CSS
  - Everybody knows how to write plain html. But to make the webpages look reasonably nice, you need to under how the Cascading Style Sheets works.

Of course you don't necessarily need to be the expert on all the aspects.

Nowadays web developers use frameworks to modulize the tasks, so you only need to know the part relevant to your interests.

The Intermediate

- MVC
  - Almost all modern websites use the Model-View-Controller design pattern. The major advantage is that it isolates the user interface from the application logic, which allows independent development, testing and maintenance.
- An MVC Framework
  - The ODM site use CodeIgniter (Why not Rails, Django, Catalyst ... ? See later discussions.)
- Ajax
  - Asynchronous Javascript and XML is the popular client-side technique to retrieve data from the server asynchronously in the background without interfering with the display of the current page (see GMAIL for example). It dramatically increases the interactivity and the dynamic interface of your site.
- An Ajax Library
  - The ODM site use JQuery (There are not too many debates on this one. JQuery simply is the best for now. Besides Ajax, JQuery provides many other convenience functions.)

The best (and free) place for learning the basics and intermediates is GOOGLE.

I will try to introduce a few concepts and examples now on the intermediate subject,
in an attempt to draw your interests.

The Advanced

I don’t know any advanced stuff ...
And I suppose it’s not necessary for such a small web site.

Introduction to CodeIgniter and MVC Framework

Why PHP and CodeIgniter

- Why PHP

  - The most popular and available web server setup is LAMP: Linux - Apache - MySQL - PHP.

  - Many enterprise web servers are built on java or .net, but the development and maintenance level is high. PHP programmers are mundane and cheap.

  - It's attempting to use Ruby (Rails), Python (Django) or Perl (Catalyst) based web server solution, given their superior scripting abilities, powerful libraries, feature-rich frameworks and large communities (I know you all are python lovers.) However whoever decide to go this way should realize the difficulties in the web server setups (especially if you do not own the server yourself) and the portability of your site.

  - Despite all the shortcomings, PHP is still a powerful language and has many advantages

    - easy to learn, quick to develop
    - easy to setup, platform independent, scalable
    - rich documentations, many supporting groups/forums, many existing libraries

Like the never-ending vi vs. emacs war, it's really just a matter of choice.

By the way I vote for vi. I guess I'm old school ...
Maybe I should start a poll among Daya Bayers.

Why PHP and CodeIgniter

- Why CodeIgniter

  - There are maybe too many PHP based Frameworks available on the market at this time. To name a few: Zend, Symfony, CakePHP, Yii (Chinese author), Kohana ...

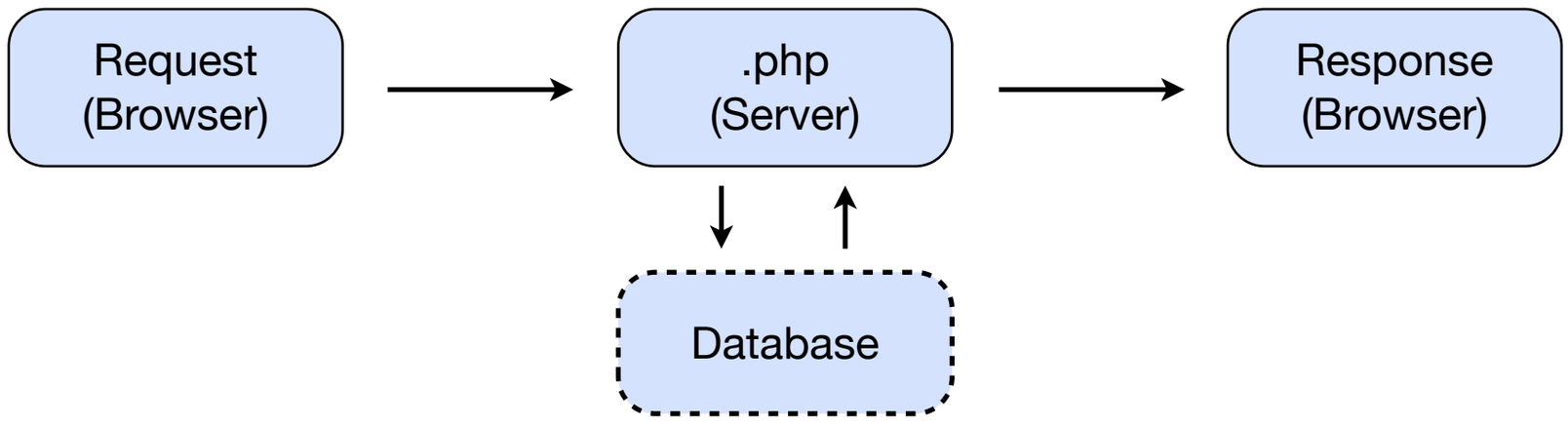
  - According to *Rasmus Lerdorf* (founder of PHP), he likes CodeIgniter '*because it's faster, lighter and the least like a framework*', and he is right.

  - I choose CodeIgniter because it is:

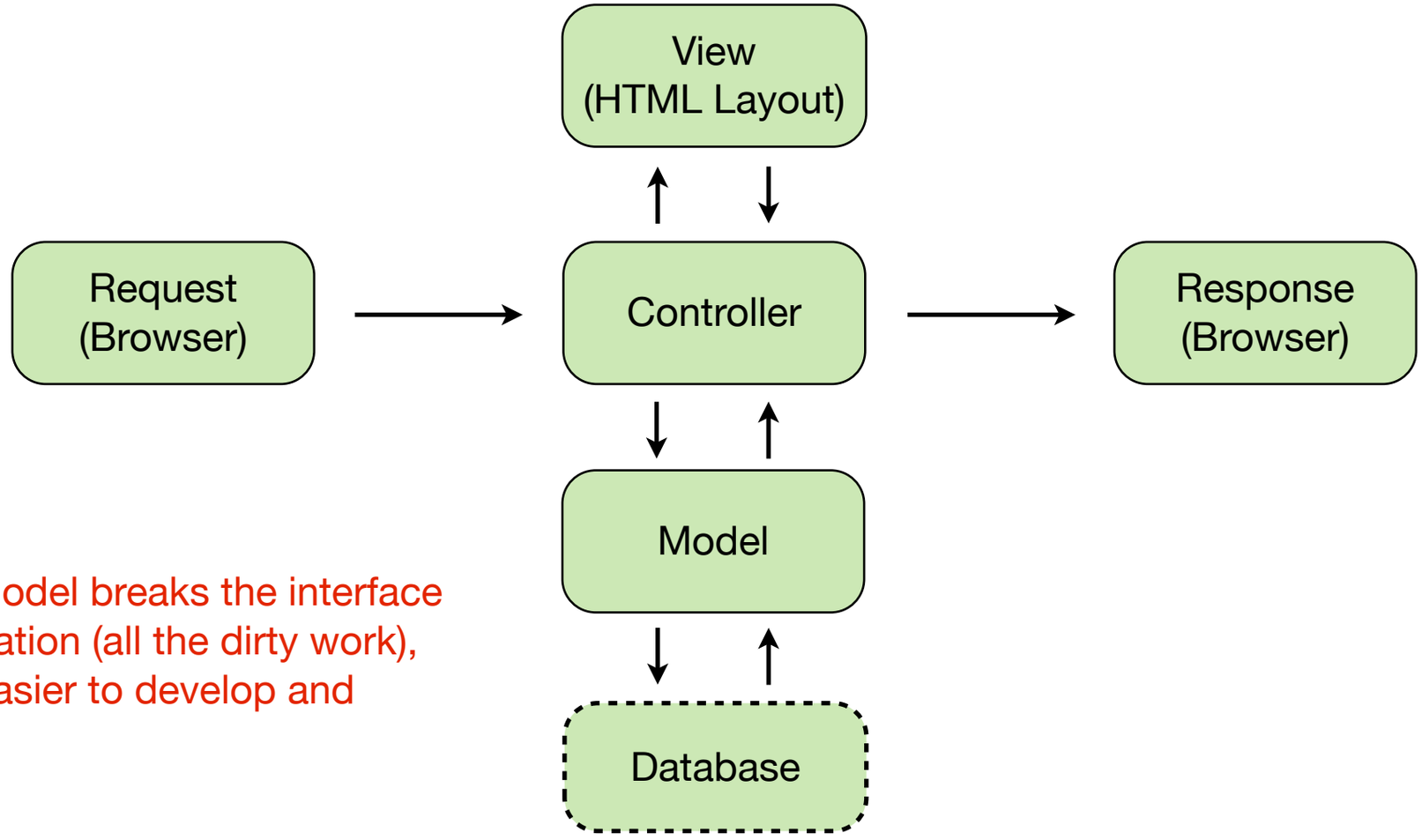
    - small (2.2MB) and portable: drag to your directory, and run! almost no configuration required.
    - loosely coupled framework. easy to learn, fast to load. no bloated core libraries.
    - well-documented, large and friendly community support, many online tutorials.
    - very customizable. scalable with many existing third-party libraries/plugins.

Model-View-Controller

- The Traditional Way:

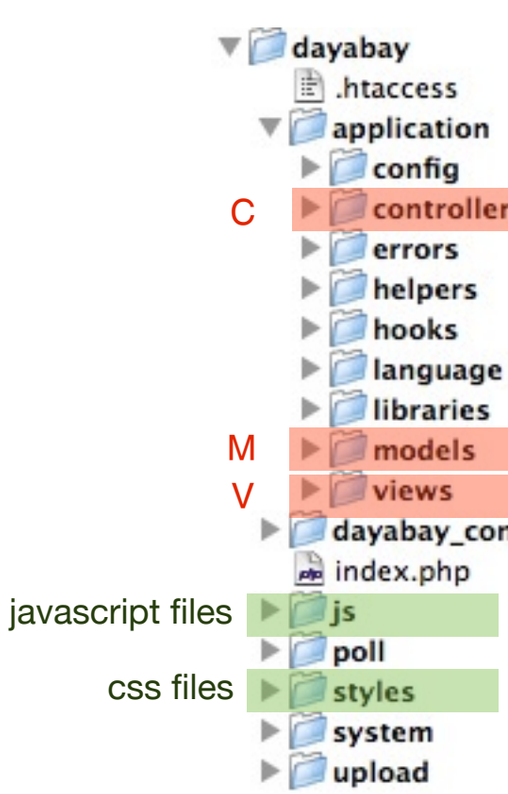


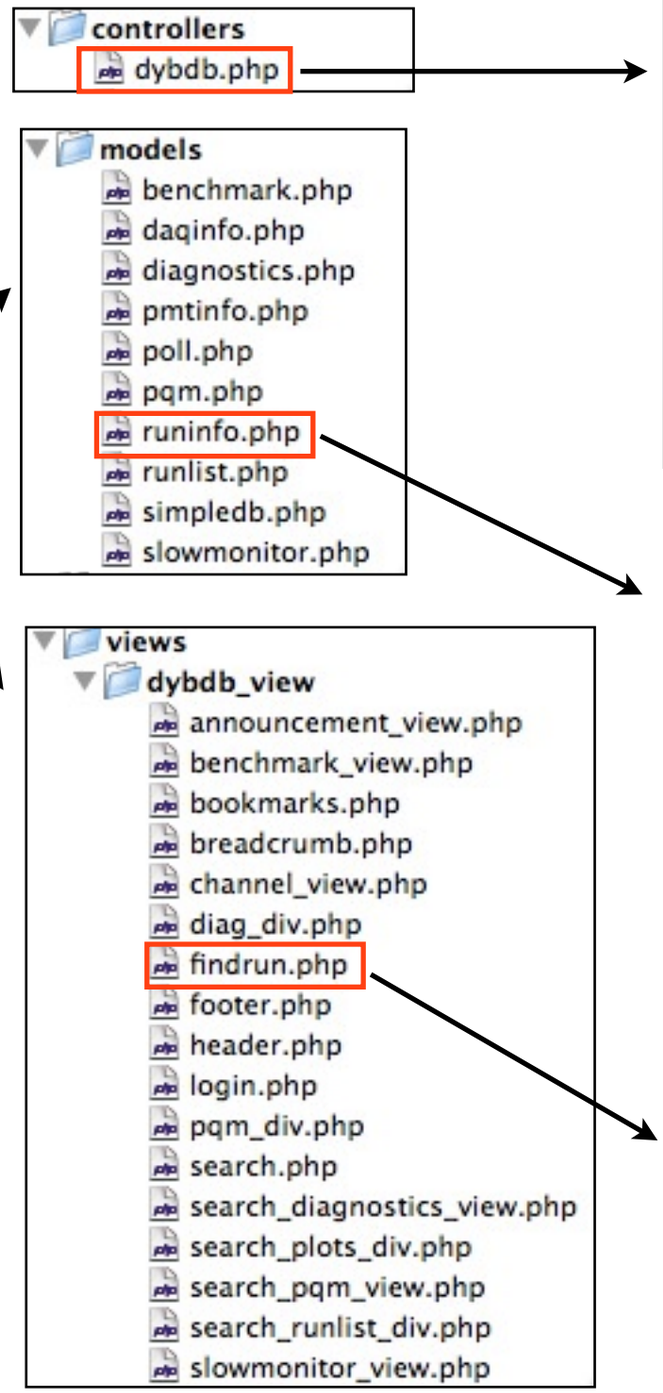
- The MVC Way:

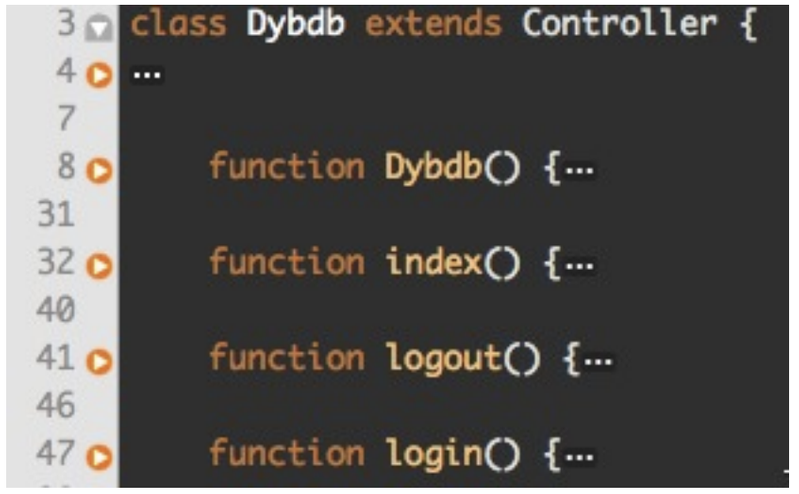


The extra layer of View and Model breaks the interface (HTML layout) with the application (all the dirty work), which makes the site much easier to develop and maintain.

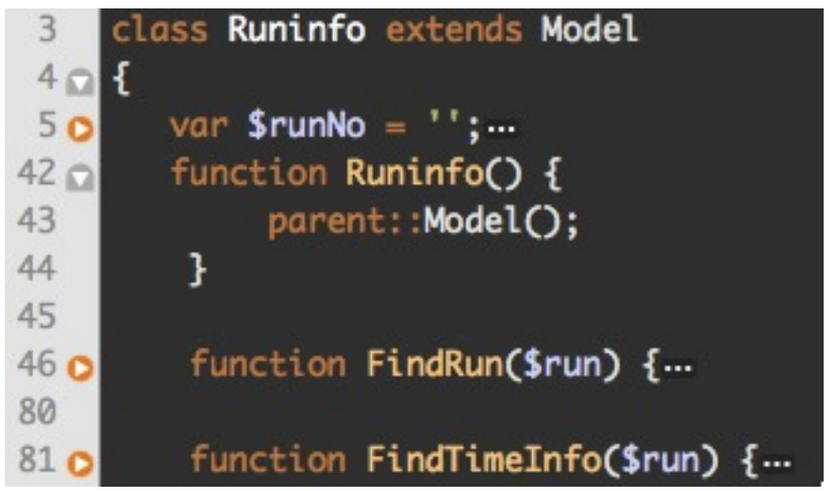
CodeIgniter's Implementation of MVC







```
3  class Dybdb extends Controller {
4  …
7
8      function Dybdb() {…
31
32     function index() {…
40
41     function logout() {…
46
47     function login() {…
```



```
3  class Runinfo extends Model
4  {
5      var $runNo = '';…
42     function Runinfo() {
43         parent::Model();
44     }
45
46     function FindRun($run) {…
80
81     function FindTimeInfo($run) {…
```

```
5  <div id="content">
6      <?php include 'search_runlist_div.php'; ?>
7
8      <?php if (isset($runlist)): ?>
9          <div class = "subject">…
76     <?php endif; ?>
```

The color coded folders are in general the only places you need to put your own code into.

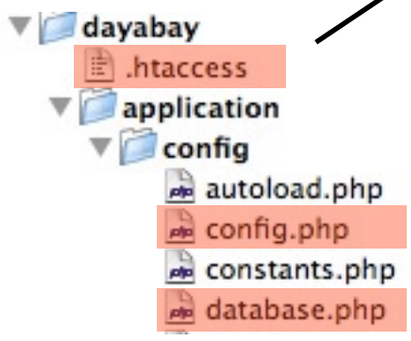
View or download all the source code at
http://dayabay.ihep.ac.cn/tracs/dybsvn/browser/people/zhang/ODM

A Few Configurations

- CodeIgniter requires minimum configurations to run. In principle you only need:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /project/dayabay/dybruns/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ index.php/$1 [L]
</IfModule>
```

Use mod_rewrite to remove the annoying 'index.php?' from the url.




```
14  // $config['base_url']  = "http://chao.local:8888/dayabay/";
15  // $config['base_url']  = "http://blinkin.krl.caltech.edu/~chao/dayabay/";
16  $config['base_url']  = "http://portal.nersc.gov/project/dayabay/dybruns/";
```

Set up the base url.
The only line of code you need to change before uploading to different servers

```
83  $db['lbl']['hostname'] = "dayabaydb.lbl.gov";
84  $db['lbl']['username'] = "dayabay";
85  $db['lbl']['password'] = "******";
86  $db['lbl']['database'] = "offline_db";
87  $db['lbl']['dbdriver'] = "mysql";
88  $db['lbl']['dbprefix'] = "";
89  $db['lbl']['pconnect'] = FALSE;
```

Set up the database you want to connect to.

CodeIgniter's Url Segments

- CodeIgniter (and many other frameworks) maps url segments into controller function calls, which makes creating dynamic webpages a piece of cake

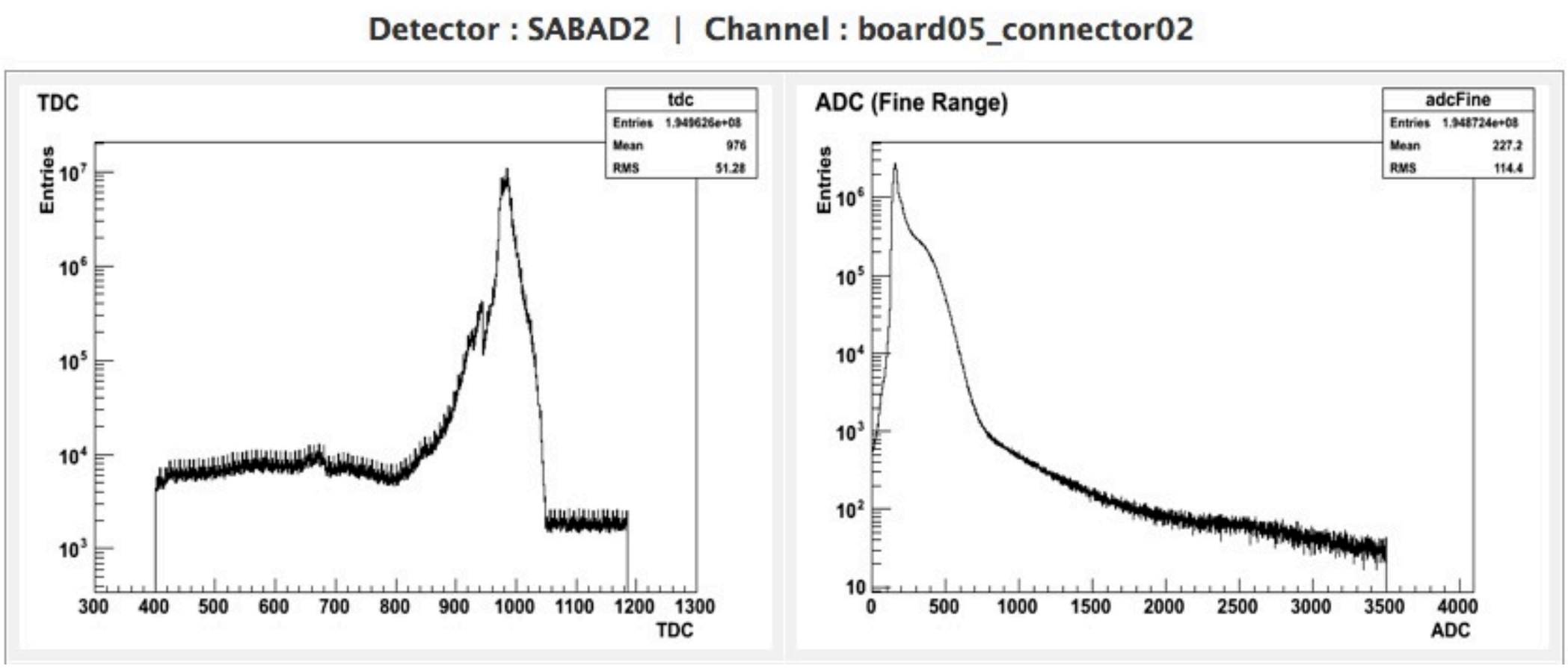
For example, this is a function defined in the Controller Dybdb:

```
448    function diagnostics($run, $detector, $channel) {…
```

The following url actually calls the function Dybdb::diagnostics(5773, 0, 1). Isn't that neat?

base url class function parameters

portal.nersc.gov/project/dayabay/dybruns/dybdb/diagnostics/5773/0/1

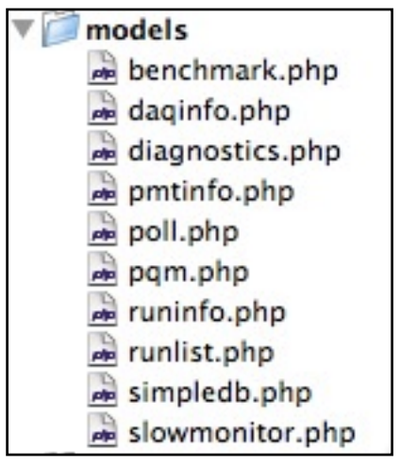


CodeIgniter's Controller

- Controller calls the Model functions and passes the data to the Views.

```
3  class Dybdb extends Controller {
4  ...
8      function Dybdb() {
9          parent::Controller();
10         ...
17
18         $this->load->model('Simpledb');
19         $this->load->model('Pqm');
20         $this->load->model('Diagnostics');
21         $this->load->model('Pmtinfo');
22         $this->load->model('Slowmonitor');
23         $this->load->model('Runinfo');
24         $this->load->model('Runlist');
25         $this->load->model('Daqinfo');
26         $this->load->model('Benchmark');
27         $this->load->model('Poll');
28         ...
30     }
```

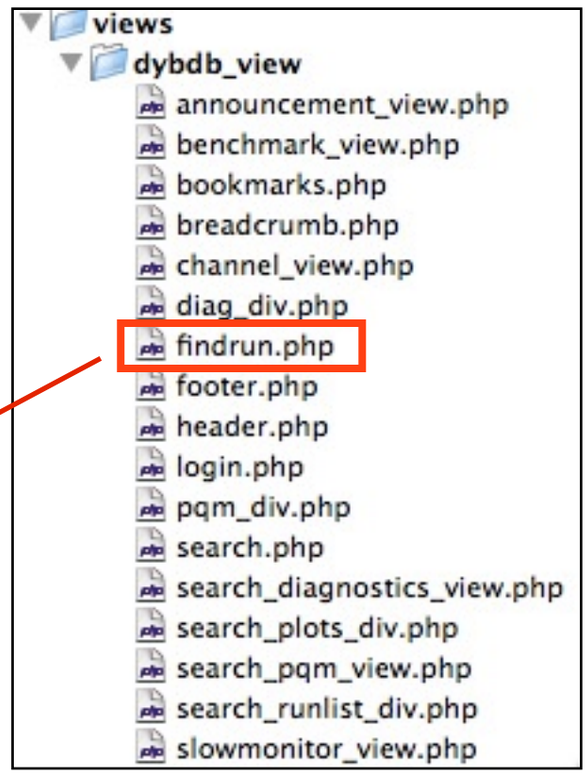
loads all your models
at the beginning



calls the function FindRun($run)
from Model 'runinfo.php'

```
156    function findrun($run, $currentrun) {
157        ...
184            $data['run'] = $run;
185            $this->Runinfo->FindRun($run);
186            $data['runinfo'] = $this->Runinfo;
187            ...
210            $this->load->view('dybdb_view/findrun', $data);
```

renders the View 'dybdb_view/findrun.php' with the array '$data'



CodeIgniter's Model

- Model does all the dirty work by interacting with the databases

```
3   class Runinfo extends Model
4   {
5       var $runNo = '';...
42      function Runinfo() {
43          parent::Model();
44      }
```

Example:
To find the calibration information for a calibration run, one need to query the DaqCalibRunInfo table.
This is done in the Model 'runinfo.php'

```
25      var $num_calib_rows = 0;
26      var $calib_array = array();
27      var $sourceID_dict = array(
28          '1' => 'LED',
29          '2' => 'Neutron',
30          '3' => 'Ge-68'
31      );
32      var $ledID_dict = array(
33          '0' => 'NONE',
34          '1' => 'A',
35          '2' => 'B',
36          '3' => 'C',
37          '4' => 'MO_BOT',
38          '5' => 'MO_MID',
39          '6' => 'MO_TOP'
40      );
```

```
101     function FindCalibrationInfo($run) {
102         $query_str = "SELECT sourceIdA, zPositionA, sourceIdB, "
103         . "zPositionB, sourceIdC, zPositionC, "
104         . "duration, ledNumber1, ledNumber2, "
105         . "ledVoltage1, ledVoltage2, ledFreq, "
106         . "ledPulseSep, ltbMode, HomeA, HomeB, HomeC "
107         . "FROM DaqCalibRunInfo WHERE RunNo = " . $run;
108         $query = $this->db->query($query_str);
109
110         $this->num_calib_rows = $query->num_rows();
111         $this->calib_array = $query->result_array();
112
113         foreach ($this->calib_array as $i => $row) {
114             $this->calib_array[$i]['sourceIdA_str'] = $this->sourceID_dict[$row['sourceIdA']];
115             $this->calib_array[$i]['sourceIdB_str'] = $this->sourceID_dict[$row['sourceIdB']];
116             $this->calib_array[$i]['sourceIdC_str'] = $this->sourceID_dict[$row['sourceIdC']];
```

CodeIgniter's View

- View writes the conventional HTML codes, except that it knows the variables passed in by the controllers.

Controller 'dybdb.php'

```
156     function findrun($run, $currentrun) {
157         ...
184             $data['run'] = $run;
185             $this->Runinfo->FindRun($run);
186             $data['runinfo'] = $this->Runinfo;
187             ...
210             $this->load->view('dybdb_view/findrun', $data);
```

Now the View has all the access to the keys inside $data

```
<div class = "subject">
   <h2>General Information Run <span id='runno'><?php echo $runinfo->runNo; ?></span></h2>
   <table border = "0" cellpadding="0" cellspacing="1", style="width:99%" class="tableborder">

      <tbody>
         <tr>
            <td><h6>Run Number</h6></td><td class='value'><?php echo $runinfo->runNo; ?></td>
            <td><h6>Start Time</h6></td><td class='value'><?php echo $runinfo->timestart; ?></td>
            <td><h6>Stop Time</h6></td><td class='value'><?php echo $runinfo->timeend; ?></td>
         </tr>
         <tr>
            <td><h6>Run Type</h6></td><td class='value'><?php echo $runinfo->runType; ?></td>
```

View 'dybdb_view/findrun.php'

Final Words on CodeIgniter

- That's pretty much all you need to know to get started with CodeIgniter
- CodeIgniter has provided many convenience classes and helpers for you. Once you have explored them, you can build up your site in no time.
- Go to CodeIgniter's User Guide (http://codeigniter.com/user_guide/index.html) whenever you need to look for something. The documentation is the best I've ever seen (clear and thorough, excellent example for software developers.) I found it's even enjoyable to read from the beginning all the way to the end.

Assuming you are not bored yet, we'll start the next big subject ...

Introduction to JQuery and Ajax

Why JavaScript

- JavaScript was considered evil in the old days. People somehow always blamed JavaScript for those annoying pop up ads ...
- Once again, not the first one, but it is GOOGLE who gave JavaScript a new boost by widely deploying Ajax technology with Gmail and Google Map (around year 2005.) Suddenly people realized that it's so convenient that one can see changes without refreshing the pages again and again.
- In general, the appropriate usage of JavaScript will
  - relieve the burden from the server. Severs are busy handling many requests from all over the world. On the other hand you own computer is likely very fast (mine is not ...) By balancing between the server-side and the client-side scripting, a better performance can be achieved.
  - increase the (apparent) response of your browser. By using AJAX, the request to the server can be sent in the background and then retrieved asynchronously. What that means is that you don't have to wait for the server to finish processing all the data in order to see the result page. Your page can be updated every time when new information arrives, without you being manually refreshing the page. Your page appears to be faster now!
  - increase the dynamics of your page. You can do all kinds of cool effects now. But be aware that abusing the 'cool effects' will annoy people. In general, you should not do 'cool effects' because '*it's cool*', you do it because it provides some convenience. Most people still prefer simplicity.

Why JQuery

- JQuery is the most popular and powerful JavaScript library today, period. Its main features include
  - Cross-browser support
  - DOM selection with CSS style selectors
  - DOM traversal and modification
  - Events handling
  - Effects and animations (together with JQuery UI)
  - Ajax support
  - Large repository of third-party plugins.

DOM stands for Document Object Model, a convention adopted by XML, HTML, XHTML documents

You will love it the moment you start using it. And I can assure you that you will never want to fall back to the old way of writing JavaScripts.

A Word On ‘Cross-browser’

- If you ask a front-end web developer about his biggest headache, 90% of the time the answer is ‘cross-browser compatibility’
- Take the four mainstream browsers for example: IE, Firefox, Chrome, Safari. They have:
  - different default style: Things look different on different browsers
  - different CSS support: Many times IE requires hacks to make certain CSS settings work. Your wonderful website could look like a mess in other browsers.
  - different JavaScript Engine: Gecko(Firefox), WebKit(Chrome, Safari), Trident(IE), which interprets objects slightly differently, but enough to screw you up if your are not coding careful enough.
- JQuery helps to resolve the JavaScript issue if you follow their (official) libraries.

Sometimes I wish all Daya Bayers agree to use only one browser.
But I know that’s not gonna happen ...

JQuery's DOM Selection and Transversal

- JQuery' first uniqueness comes from its CSS style DOM selector, which is based on the Sizzle Selector Engine. Once you start using it, you wonder why it's not yet been ported to the XML libraries in other languages.

Some HTML code (remember that HTML is a special case of XML)

```
<ol>
    <li>list item 1</li>
    <li class="hello">list item 2</li>
    <li>list item 3</li>
    <li>list item 4</li>
</ol>
<ul class="hello">
    <li>list item 5</li>
</ul>
```

JQuery Selections

```
$("li")          //selects all 5 list items
$("li:eq(0)")    //selects the first list item
$("ol li")       //selects <li> that are children of <ol> (list item 1,2,3,4)
$("ol li:even") //selects even-numbered <li> that are children of <ol> (list item 2,4)
$(".hello")      //selects elements with class 'hello' (list item 2 and the ul element)
$("li.hello")    // selects <li> with class 'hello' (list item 2)
$("li.hello").next("li") // selects the direct sibling of $("li.hello") (list item 3)
...
```

I can go on and on but you start to see how easy and versatile it is to select specific xml elements, compared with the old way of selecting (getElementsBy...) and traversing nodes . There are many other selectors and they are really the backbones of JQuery.

JQuery's DOM Modification

- After you have selected the DOM elements, JQuery provides all kinds of methods that you can perform to modify the elements. You can even chain these methods together to perform complicated operations.

Same HTML code

```
<ol>
    <li>list item 1</li>
    <li class="hello">list item 2</li>
    <li>list item 3</li>
    <li>list item 4</li>
</ol>
<ul class="hello">
    <li>list item 5</li>
</ul>
```

JQuery DOM Modification Chain

```
$("li.hello")                          // select list item 2
.next("li")                            // go to list item 3
.css("background-color": "green")      // change its background to green
.html("new list item 3")               // change its text to "new list item 3"
.after("<li>list item 3.5</li>")       // add a new list item directly below it
.prev()                                // go back to list item 2
.removeClass("hello")                  // remove its class "hello"
...
```

Again I can go on and on but you start to see how easy it is to modify the DOM dynamically with JQuery's clean grammar.

JQuery's Event Handling

- JQuery listens to many events (click, double click, hover, focus, de-focus ...), and you can easily bind specific functions to these events

Example:
All figures on the ODM site can be double clicked which then leads to its original size. This is achieved by the following 4 lines of code:

```
66 $(".img_db").dblclick(function() {
67     window.location = $(this).attr("src");
68     return false;
69 });
```

```
<img class="img_db" src="http://blinkin.krl.caltech.edu/
~chao/dybprod/diagnostics/runs_0005000/runs_0005700/
run_0005773/detector_SABAD2/channel_board05_connector01/
pedFineVsTdc.png" width=400 height=300/>
```

In the html, all <img> tags of the figures have the class "img_db"

- The JavaScript does the following things:
  - It selects all elements with class "img_db" (that means all the figures)
  - It binds an anonymous callback function (or closure as is the official term in JavaScript) to all figures' double click events $("img_db").dbclick( )
  - Inside the closure, the currently double clicked element has a reference called $(this). Its 'src' attribute (which is the long url pointing to the figure's original location) is taken and sent to 'window.location' which does the redirecting.

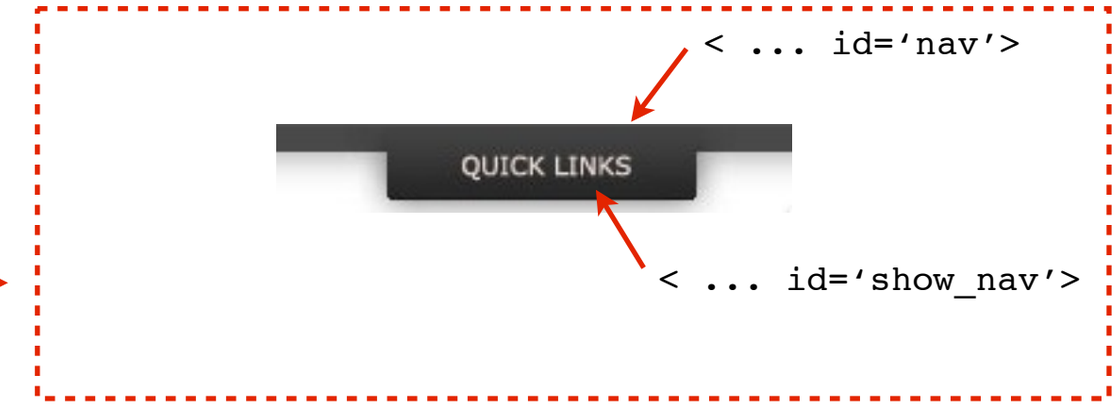
A lot of work with just a few lines of code, right?
That's exactly what JQuery's slogan says: "write less, do more" !

JQuery's Effects and Widgets

- JQuery and JQuery UI provide rich effects for animation as well as highly configurable ready-to-use widgets

Example:
The blinds effect of the bookmarks on the ODM is achieved by the following 3 lines of code:

```
2  $("#show_nav").click(function() {
3      $("#nav").toggle("blind", "", "fast");
4  });
```



Example:
The ODM site extensively use the Datepicker and Progressbar widgets from JQuery UI

Progressbar

Datepicker

JQuery's Ajax Support

- Doing Ajax with JQuery is very easy

```
$.ajax({
    type: "post",     // or "get", default is "get", in CodeIgniter we recommend "post"
    url: "test.php",    // the server-side script where your request is sent
    data: {'runno':1000, 'fig':'charge'},    // you can send some data to the server along the request
    dataType: 'xml',    // the format of the data the server sends back
                        // the format can be 'xml' / 'html' / 'json' / 'text' / ...
    async: true,    // default is true. what's the point of using AJAX but set it to synchronous??
    success: function(data) {    // called if the request succeeds
        doSomethingWith(data);   // the data returned from the server, with the dataType format
    },
    error: function() {    // called if the request fails
        doSomethingIfRequestFails();
    }
    complete: function() {    // called after success and error callbacks are executed
        doSomethingWhenRequestFinishes();
    }
});
```

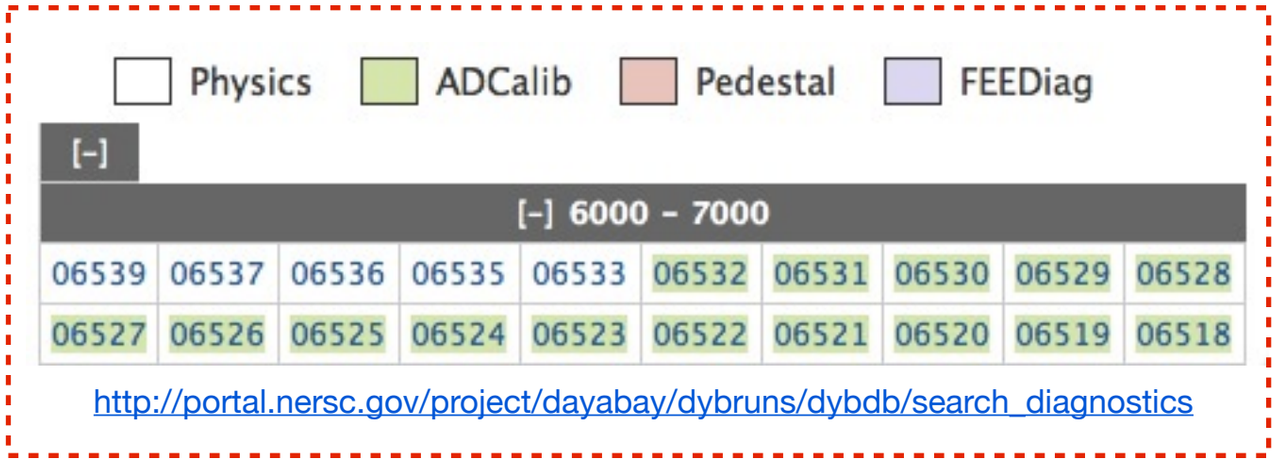
A Typical AJAX Request Streamline

- The request is sent to the server in the background (Asynchronously) through this JAvaScript.
- The page quietly waits for the server to process the request (you don't even notice it)
- The server then sends the result back, usually in the format of XML (but not necessarily. Recently the more light-weighted JSON format became popular.)
- The JavaScript then catches the result and show it dynamically on the page (no refreshing needed) by modifying the DOM.

A Final Case Study

Why Color Coding the Run Type

- On the ODM site, I color coded the run type for all the runs.
- I did this not because *it's cool* (well, it's sorta ...), but because it helps the human eyes to quickly pick up the information.



- You may notice that the table is shown first, then the color is appended to the table. That's the result of Ajax: the run type information is gathered in the background and then added to the page only after the information is ready.
- Why use Ajax? The database connection might be slow (it IS slow if the DB is in China while the web server is in US.) You want to immediately show something to the user (remember that users have no patience?) instead of showing everything all at a time only after 5 seconds of querying the DB.

I will show you how easy it is to achieve this in 3 steps
by combining all the stuff I just introduced.

First, Print out the Run List Table

Controller ‘dybdb.php’

Calls the Model

```php
function search_diagnostics() {
    $data['runlist'] = $this->Diagnostics->getarray_runlist();
    $this->load->view('dybdb_view/search_diagnostics_view', $data);
}
```

Renders the View

Model ‘diagnostics.php’

```php
function getarray_runlist() {
    $runlist = array();
    $xml = simplexml_load_file(
        'http://portal.nersc.gov/project/dayabay/dybprod/runs.xml');
    foreach( $xml->run as $run ) {
        array_push($runlist, $run->runnumber . '');
    }
    rsort($runlist);
    return $runlist;
}
```

The Model doesn’t necessarily need to interact with a database.
You can put any dirty work in the Model.
Here the Model gets the list of processed runs from an XML file.

View ‘search_diagnostics_view.php’

```php
$column_index = 1;
$per_coloumn = 10;

echo '<table>';
foreach ($runlist as $run) {
    if ($column_index == 1) { echo '<tr>'; }
    echo '<td>';
    echo anchor('dybdb/findrun/' . $run,
                sprintf('%05d', $run));
    echo '</td>';
    $column_index++;
    if ($column_index == $per_coloumn + 1) {
        echo "</tr>";
        $column_index=1;
    }
}
echo '</table>';
```

Notice that each table cell has a
run_no link

Slightly simplified from the actual code

Second, Make the Ajax Request

JavaScript ‘search_diagnostics.js’

```
$.ajax({
    type: "POST",
    url: "json_runtype",
    dataType: "json",
    success: function(data) {
        ...
    }
});
```

Request is sent to the Controller in the background

Controller ‘dybdb.php’

```
function json_runtype() {
    $this->load->database($this->session->userdata('database'));
    $this->Diagnostics->json_runtype();
}
```

Controller calls the Model which does the actual wok

Model ‘diagnostics.php’

```
function json_runtype() {
    $query_str = 'SELECT runNo, runType FROM DaqRunInfo';
    $query = $this->db->query($query_str);
    $run_array = $query->result_array();

    $json = "{\n";
    foreach ($run_array as $i=>$row) {
        $json .= ('"'
        . sprintf('%05d', $row['runNo'])
        . '":"' . $row['runType']
        . '",' . "\n");
    }
    $json = rtrim($json, "\n,");
    $json .= "}\n";
    echo $json;
}
```

JavaScript catches the result and does something with it

Firebug Plugin for Firefox, very useful for debugging

POST http://portal.nersc.gov/project/dayabay/dybruns/dybdb/json_runtype

Headers | Post | Response | HTML

```
{
"00006":"Pedestal",
"00007":"FEEDiag",
"00009":"FEEDiag",
"00010":"Pedestal",
"00011":"Physics",
```

Server returns result in a ‘JSON’ format, which is simple ‘key:value’ pairs, more light-weighted than the cumbersome XML format but still readable by human

Finally, Show The Result On The Page

JavaScript ‘search_diagnostics.js’

```
$.ajax({
    type: "POST",
    url: "json_runtype",
    dataType: "json",
    success: function(data) {
        $('#td a').each(function(){
            var run = $(this).html();
            var runtype = data[run];
            $(this).addClass(runtype);
        });
    }
});
```

If request succeeds:

Bind a closure to each link inside the table cell

Get this link’s text (the run number)

Get the corresponding run type from the JSON data

Add a class attribute to this link depending on the run type

Physics ADCalib Pedestal FEEDiag

[-]

[-] 6000 - 7000									
06539	06537	06536	06535	06533	06532	06531	06530	06529	06528
06527	06526	06525	06524	06523	06522	06521	06520	06519	06518

http://portal.nersc.gov/project/dayabay/dybruns/dybdb/search_diagnostics

CSS ‘main.css’

```
796  a.ADCalib { background: #cfeaa5; }
797  a.Physics { background: white; }
798  a.Pedestal { background: #f2c1ba; }
799  a.FEEDiag { background: #ddd3f3; }
```

Each class has a different background color, defined in the CSS file.

Done! Easy, right?

Final Words

- You now know (almost) as much as I do.
- Web programming is useful even for a physicist.
- Web programming is easy even for a physicist.
- Web programming is fun.
- However be aware that web programming could be tricky and sometimes frustrating. Again, Google is your best friend, and don’t be afraid to ask around.

Happy Web Programming!

Fin.