opendelight

A Rapid Application Development Framework for PHP

by Ashwini Rath

If you do an internet search for PHP frameworks, you may be overwhelmed by the number of frameworks and their features. So, after the first glance at this article, you may say: Oh, no! Another one? I would ask you to reserve judgment until you finish this article. Let's start with a few facts before moving on to a detailed introduction to this rapid development framework.

REQUIREMENTS

PHP: *5.1+ with PDO enabled*

Other Software: *MySQL 5.0+*

Related URLs:

- Offical website of Opendelight: http://www.adiipl.com/opendelight
- Documentation of Framework: http://www.adiipl.com/opendelight/docs
- PHP Data Object http://www.php.net/pdo

An application developer seeks to start the application development with a bang, to focus more on the business logic and the usability of the application, and to manage the life-cycle of the application with a multitude of change requirements from clients - the same is the case with a PHP developer. The developers feel restricted from unleashing their creativity fully because they divert their attention from implementing actual business requirements whenever there are repeated demands to develop application architecture, code convention, and file structure. This gives rise to the need for a development framework that takes care of architectural requirements, leaving developers to focus on writing business logic that reflects the client's exact requirements and develops a good user interface (UI). If developers have access to a rapid application development tool for the chosen framework, things become much easier. The Opendelight framework provides exactly that, and it is free open source software!

Opendelight framework comes with a browser-based *Integrated Development Environment (IDE)* to manage all aspects of application development, a set of libraries to implement routine tasks, and a fully functional skeletal application structure to start with. All of this comes with a well laid-out file structure and a simple, robust architecture to ensure a high performing, scalable, and secured application.

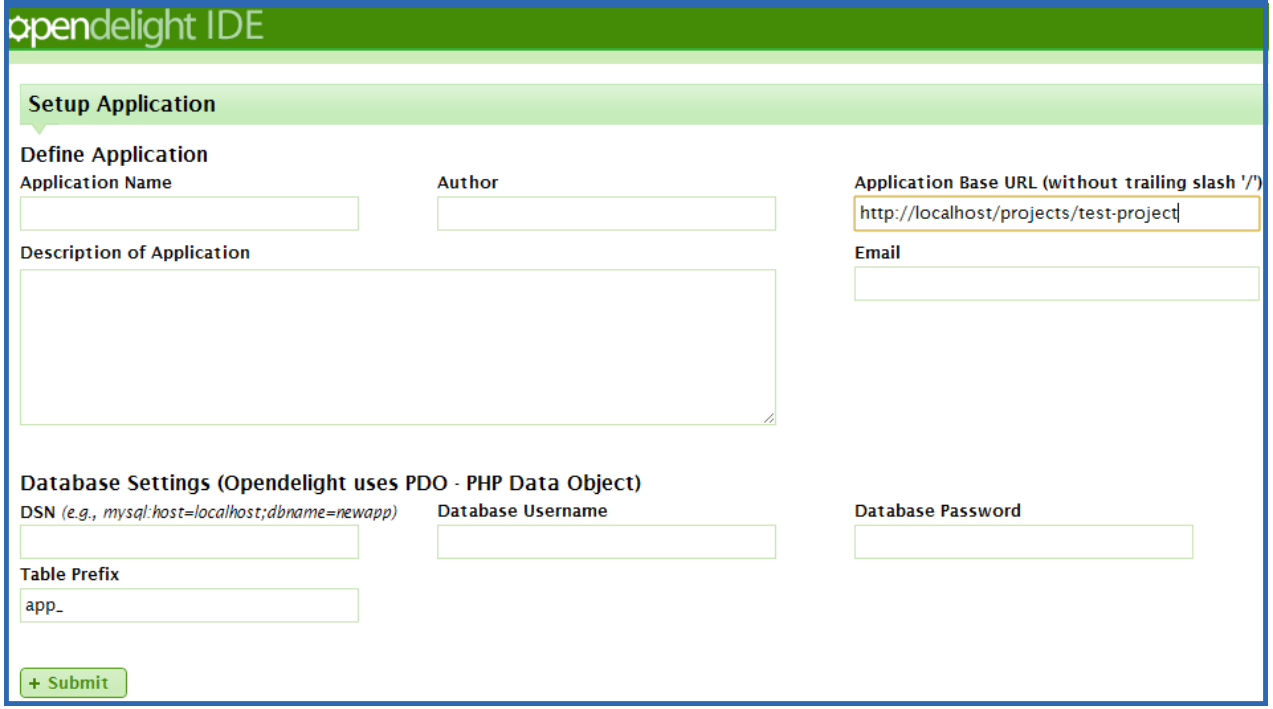
Getting Started with Opendelight – Hello World!

Developing and publishing a working application is quick and can be done in three simple steps:

download the framework from the website http://www.adiipl.com/opendelight/download.php, decompress the framework to your development location, and install the framework.

After you download and decompress, point your browser to **application-base-url/delight-ide/install.php** (**application-base-url** is the location where you decompressed the downloaded files). Click *Setup Application*, and you will be taken to a page to fill the details of your new application: *Application Name*, *Author Name*, *Application Base URL*, *Description of Application*, *Email Address* and *Database Settings* (See Figure 1). If you need further help during the installation, please refer to the documentation at http://www.adiipl.com/opendelight/docs/installation.php. MySQL is used for the example application. Note that Opendelight uses *PHP Data Objects (PDO)* for database access.

FIGURE 1



Successful installation will provide you access to the newly created application (**application-base-url**) and Opendelight IDE (**application-base-url/delight-ide**) with login information for both (Note that the **admin** user, with the access role **Administrator** created at installation, is the same for both the application and IDE). Point your browser to **application-url/sign.php**, which will show the *Sign In* form of the application (See Figure 2). If you enter the admin login credentials, you will see a "Hello World!" page (See Figure 3). At this point, you have a working application even though it is just a skeletal structure. You can now begin implementing your actual business requirements from within Opendelight.

It is important, though, to be aware of what happened after the installation completed. A brief inspection of the file structure in the root folder of application will reveal a folder named **delight-ide** - the entire IDE resides here. You can delete this folder anytime without affecting the application. Other folders are important for the application. The folder **/lib** carries classes added to the framework to facilitate common logic implementation during development. The **/model** folder contains two subfolders: **class** and **script**. The first one is meant for you to store the classes that you will write as part of the business logic implementation. The second one will be used to store any legacy code. The folder **/view** carries UI related files in respective subfolders. For more details about these folders and others, please refer to the documentation at http://www.adiipl.com/opendelight/docs/application-file-structure.php

Opendelight currently has seven database tables that come with the installation. These tables store information about the application, its controllers, events and other details. There is no need to worry about the data in these tables nor their manipulation during the application development process as it is taken care of by the framework itself. However, you may like to refer to the documentation at http://www.adiipl.com/opendelight/docs/application-database-structure.php to learn more.

Now to look at the way Opendelight treats different aspects of an application, its development, and its manageability. During the discussion, any application created with the Opendelight framework is referred to as the *application* and the one just installed as the *example application*.

FIGURE 2

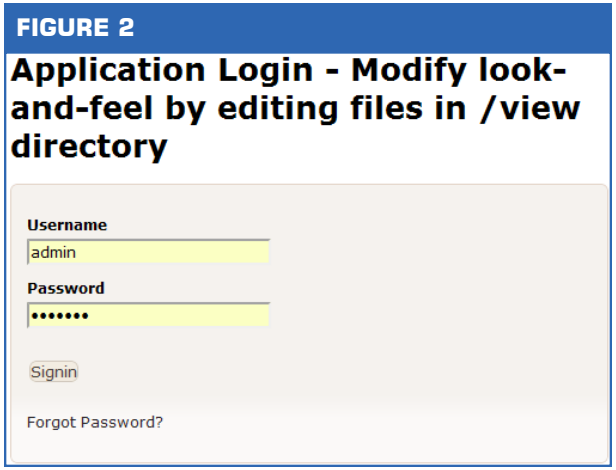
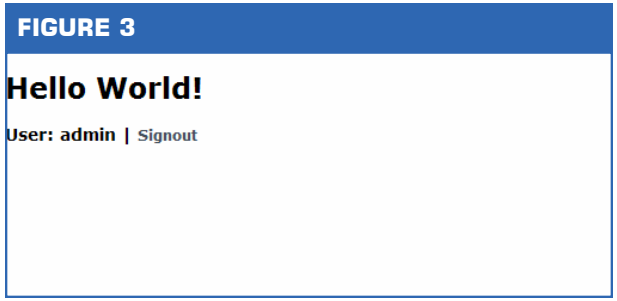


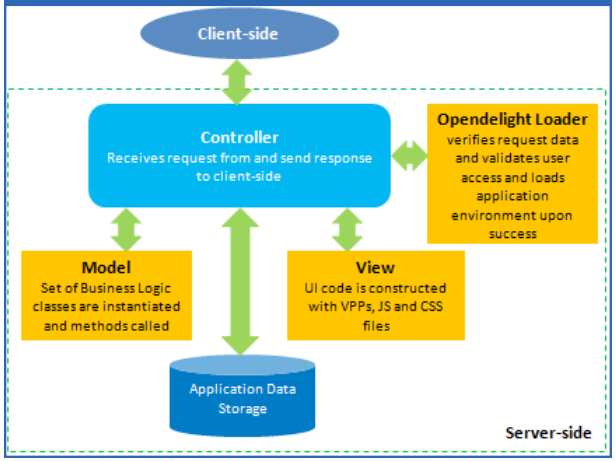
FIGURE 3



Behind The Scenes - Technical Architecture of Applications

The Opendelight framework exploits the multitier architecture of web applications and uses the *Model-View-Controller (MVC)* design pattern at the center of application architecture. This architecture facilitates

FIGURE 4



an event-driven approach to data manipulation by the application, and uses the *Role-Based Access Control (RBAC)* mechanism to manage user access control to the application. (Figure 4)

The HTTP-request from the client-side instance is built within the basic URL format: `application-base-url/controller-name?ID=N`. Note that the framework also supports using other query string parameters in the URL apart from allowing URL beautification with the help of *Apache htaccess* or *IIS URL rewrite*. The first part, `application-base-url`, is already known. The next part, `controller-name`, is the file name along with the path from the root of the application and can be managed through the IDE. In the example application (See Figure 5), there are two controllers: `sign.php` and `index.php`.

A controller provides a communication end point for other client and server systems. The first controller in the example is responsible for functionality such as signing in, signing out and password retrieval. It readily provides the common functions of an application, can be enhanced by writing new classes, and its UI can be changed by modifying the **View Page Parts or VPPs**. The second controller, `index.php`, manages the protected part of the application. Any number of controllers can be added or deleted. Note that `controller-name` is a file in a subdirectory, and the entire path from the root of the application is included.

The last part of the URL format is the query string `?ID=N`, where `ID` is the **Event ID** of the application instance and `N` is the value. Any controller events can be managed through the IDE. Figure 6 shows the

FIGURE 5

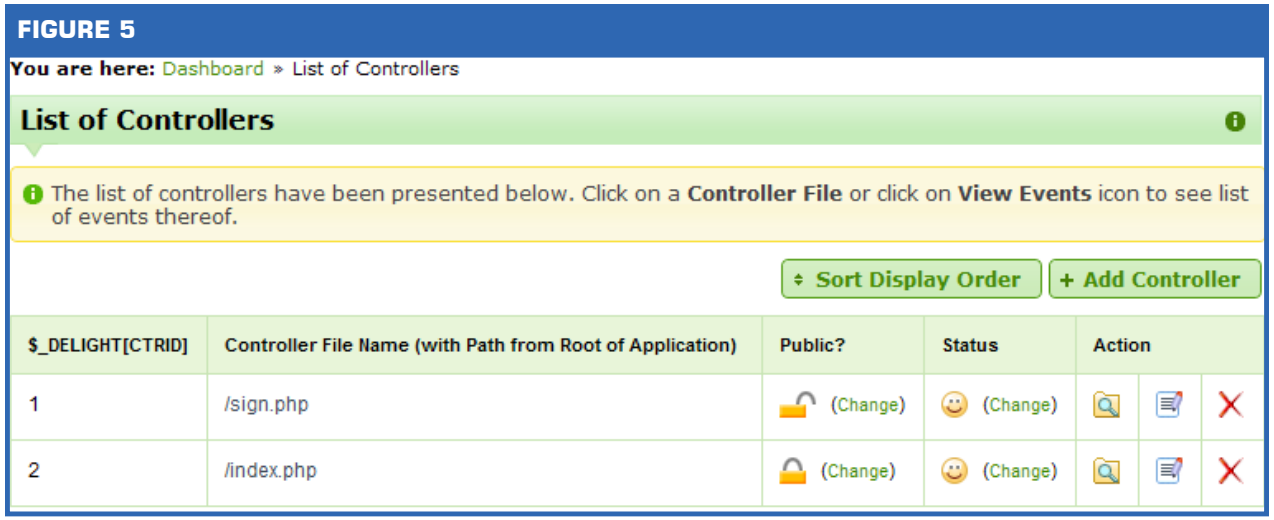
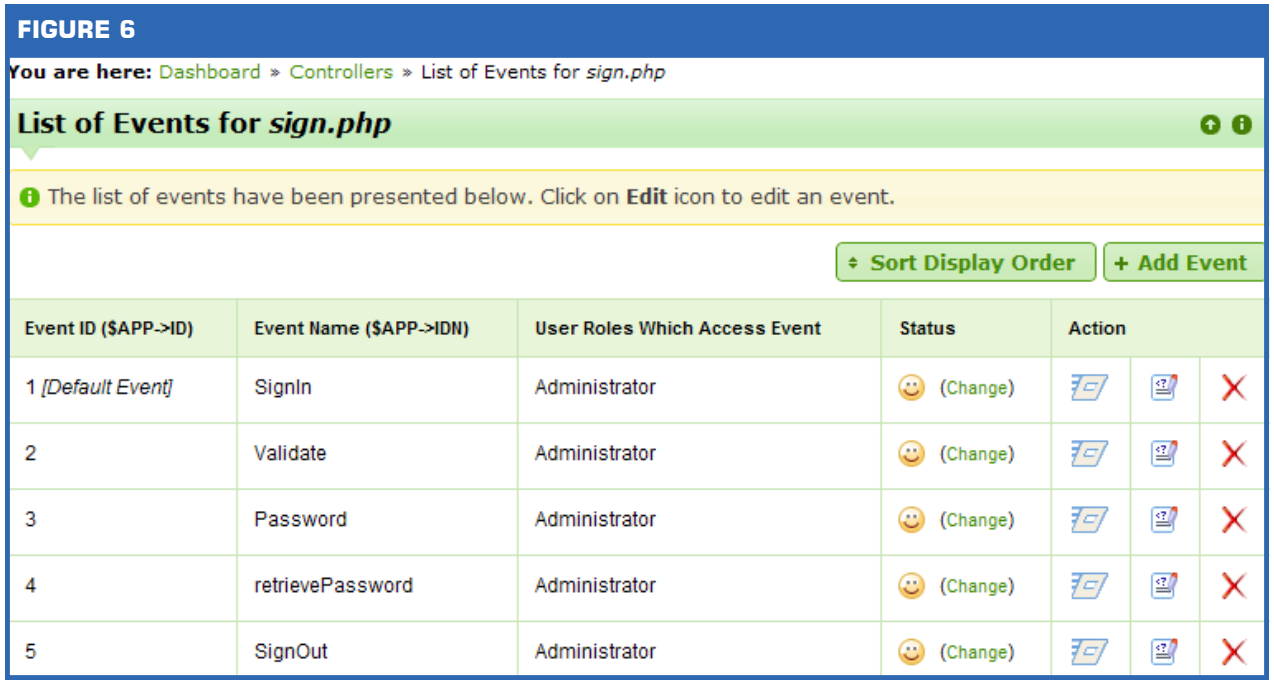


FIGURE 6



> Opendelight IDE provides tools to manage access control, user roles, business logic, and user interface code.

list of events created for the controller `sign.php` by default (see the lone event for controller `index.php` in Figure 7).

By encouraging an event-driven approach in the programming, the framework introduces the concept of an event occurring when the application is called through an HTTP-request. Any event within the application is uniquely identified with an integer and is called an Event ID regardless of which controller it belongs to. On another note, the application uses the default event for a controller, specified through the IDE's *Edit* page, if the query string does not specify an `ID` in the request URL.

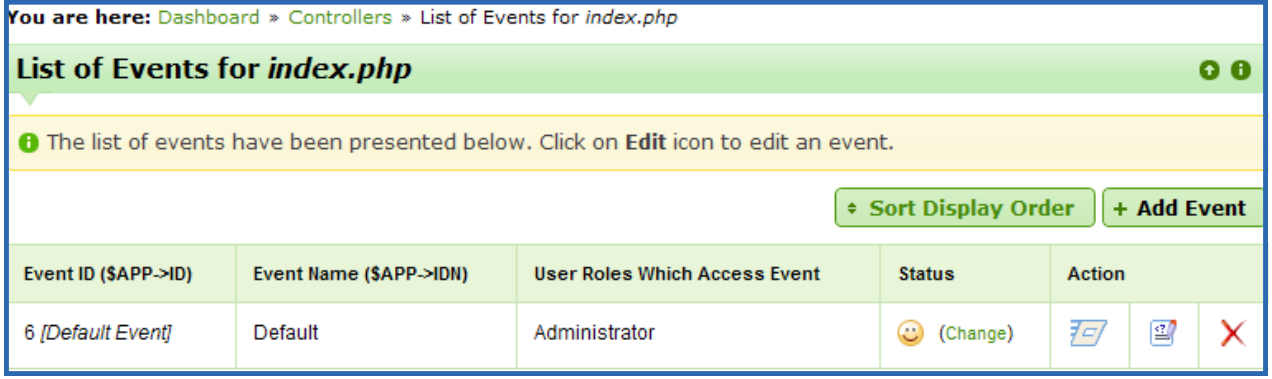
On the server-side, the controller receives the request and loads the **Opendelight Loader**. This is a file in the root folder of the application, `load.delight.php`, that performs the task of sanitizing and verifying the request data and then instantiating native framework objects, called **Opendelight Objects**. It is important to know that the user verification and user session tracking are done at the framework level, and these do not need to be worried about while developing the application apart from determining appropriate business logic objects and their respective UIs.

After user identity and access verification, the controller calls the model to execute the business logic of the application. Later, the outputs are used to construct the view. Finally, the controller sends an HTTP-response back to the client-side instance, completing the cycle.

Enforcing User Access Control in Applications

Opendelight uses an RBAC scheme to implement access control in the application. This determines user roles (a role can be shared among multiple users,

FIGURE 7



whereas one user cannot have multiple roles), and access to any particular instance of the application is determined as per its association with the corresponding event. Creating new roles, assigning a role to users and associating roles with an event are done in the IDE. To see this in action, add a new event to the controller, **index.php**, and then create a new role by choosing **index.php** as the *default* controller and the newly created event as the *default* event. Finally, create a new user by assigning this role. The landing page for this new user is visible after signing into the application - it is different from where the admin user landed. With this new user session, you will fail to access the URL of "Hello World!" until you add this role to the event responsible.

In addition, Opendelight introduces an optional verification through a custom conditional expression called **Event Verifier**, which can be set for any event through its *Edit* page in the IDE.

Object Model and Writing Business Logic

Three Opendelight Objects are created by default with each event: **$USER**, **$APP** and **$DB**. **$USER** is only available for a *private* controller, for example, **index.php**. For *public* controllers, this object does not get instantiated as no user authorization is required to access it. A public controller is responsible for serving the public pages of an application, such as **sign.php**.

Both the objects **$APP** and **$USER** adhere to an *active record* pattern and store all the data of an application instance and user, respectively. Through the IDE, you can assign different integers to be used in the application for variables called **Configuration Variables** and to use them through **$APP** later. The third object, **$DB**, is basically the instance of PDO and is instantiated with the database settings as entered during the installation process. The use of PDO gives two advantages: faster database access and relative freedom in the choice of database. The other important aspect of PDO is its support for prepared statements, thus preventing SQL-injection issues that occur within applications.

These objects are used as global keywords within the new classes that are created as part of writing business logic. This may look a bit odd for framework users in other languages who use *dependency injection* to use and extend native objects. However, this is in sync with the simplicity of usage and is a usual occurrence in PHP. Moreover, it does not contribute any disadvantage to the application over the other method.

With this knowledge about the framework, let's look at the "Hello World!" scenario. When the *Sign In* page of the application is requested, the default event **SignIn** is called since no "ID" value is supplied in the URL query string. Similarly, when the form is submitted with user login information, the event **Validate** is called (**?ID=2**) and validates the supplied user information with the information stored in the application database. This process is part of writing business logic:

```
if ($_REQUEST["hidStatus"])
{
   $oSign = new Delight_Sign();
```

> The Opendelight framework uses the MVC design pattern at the center of the application architecture and uses an RBAC mechanism to manage user access control to the application.

```
    $oForm = new Delight_Form($_POST, $APP->FORMRULES);
    $aForm = iterator_to_array($oForm);

    $sErrMsg = $oSign->signin($aForm, $oSign-
>sLoginToken);
}
```

The definition of an event allows the specifying of *Form Rules* for sanitizing and validating form input and statements calling business objects or procedural code that you have written, with appropriate conditions (called different **states of an instance of application**). Examples of states can be understood by referring to the display of any *Edit* page in a real-world application and what happens when a user saves after changing data in the *Edit* form – two states of instance of the application with an *Edit* event. In the example above, the conditional **`if ($_REQUEST["hidStatus"])`** allows two states.

View – Application User Interface

Notice the few fields on the *Edit* page of an event: a text box for View Page Parts (VPPs) and a few text boxes for **View Page Variables (VPVs)**. The former is a comma-separated list of files (with **`.tpl.php`** extensions) with client-side code interspersed with PHP print statements or simple loops/conditionals. They are called sequentially, making the order in the comma-separated list important. The client-side code is divided into multiple files as some parts are common, such as the header and the footer of a page. Note that this view also involves Cascading Style Sheets (CSS) and JavaScript, and they can be managed from within the IDE.

In constructing a view, a few things change in a VPP, even if it is common to other events. This creates the requirement of storing event-specific data in variables called VPVs. Three VPVs are already defined with the installation: **`TITLE`**, **`HEADERTEXT`**, and **`BREADCRUMB`**. More variables can be added through the IDE (**`$APP->PAGEVARS[VARIABLENAME]`**), and their respective input fields will appear on *Edit* page of the events.

Note: a recent version of *jQuery*, along with *jQuery UI*, is included in the application generated at installation. Learning jQuery is not a prerequisite to use the Opendelight framework.

How to Approach Real-World Scenarios

Creating applications requires grasping business requirements and creating a *data model*. It would take another article to explain the concepts of *data modeling*, which is outside the scope of this one. However, the end result of *data modeling* is a set of objects with well-defined interactions and relationships between their respective database tables.

Now the database for the application can be designed and the business logic classes can be created. The IDE assists in creating events with controllers for each of the public class methods. The UI and associated lists of VPPs for each event will also need to be created. Each event and its states can be tested from within the IDE. After the full application is developed, it can be moved to a production environment by moving the files and database to the destination and changing the database settings (refer to **`sys.inc.php`** in the application root folder and **`application-base-url`** in the **`*od_sys`** table).

Opendelight framework currently provides a few libraries (classes) for grid based data management, form handling and CAPTCHA apart from the sign functionality discussed in the example. Third-party tools for applications as needed or wanted, with their respective code stored in the **`/ext`** folder.

Opendelight IDE provides tools to manage access control, user roles, business logic, and user interface code. The Dashboard of the IDE provides a list of items recently edited, as well as recent blog posts and documentation links. A log of application usage can be viewed in the IDE to track errors and test applications.

Conclusion

Opendelight framework is quickly evolving with new features and libraries added on a regular basis. The framework is based on mainstream features and the simplicity of PHP, making it easy to adopt for any web application development plan. Understanding the conceptual details and effective use of the IDE will enable you to create an enterprise-grade application quickly and with ease.

ASHWINI is the CEO at ADII Research & Applications Pvt. Ltd. (http://www.adiipl.com), a technology company founded by him. His main interest lies in the application of artificial intelligence techniques and algorithms in the areas of mission-critical business solutions, application security, self-healing information architecture and semantic web.
Learn more: http://ashwinirath.com
Contact: ashwini.rath@adiipl.com