PHP application development with Prado

WEB HELPER

The Prado PHP development framework helps you quickly build web applications. **BY PETER KREUSSEL**

Google Docs and other online productivity tools reveal the potential for capturing the look and feel of a desktop application through the web programming model. Keeping the application on the server minimizes configuration and provides a more optimal use of system resources. But the Software as a Service model is not confined to huge and complicated applications on the servers of global giants like Google. Your software service also can be a custom web application running at the local level. Developers use languages like PHP to create small and efficient server-side programs that serve up results in a browser window.

For efficient web application development, it makes sense to use the building blocks of a development framework. Several PHP frameworks are available for Linux users – some of which were discussed in the February 2008 "PHP Scripting" issue of *Linux Magazine*.

One example of this new breed of PHP development tools is the Prado framework [1], an extensive and mature development environment for web applications. The letters of the name *Prado* stand for "PHP Rapid Application Development Object-oriented."

Prado addresses many facets of application development, providing GUI components that range from simple input fields to a wizard interface.

The XML template language included with Prado offers an easy syntax that is easily managed through normal HTML editors. All of the form components in Prado keep track of their own status without the developer having to worry about them. Prado also supports client-side validation controllers that check input data before the form is sent.

Component Design

The architecture of the Prado framework is based on the MVC model (model-view-controller). Each Prado application consists of one or more templates that define the appearance of the page.

A PHP file provides the program logic. Each template can include code files or other template files.

An example application that implements a web-based Advent calendar illustrates how easy it is to program with Prado. The doors of the calendar open with a mouse click (Figure 1). Doors that

Figure 1: A simple interactive application: the doors of an Advent calendar open with a mouse click; The prize appears in the white field below.

have already been opened remain open, so the application must keep track of the door status. The bordered box under the doors indicates the contents of the calendar compartment.

The application should reuse as much code as possible; the code for the doors, as well as the view and the controller components, should appear only once. Because of limited space, only one door per week is shown.

The First Step

To install the Prado framework on a web server, you only need to unpack Prado in the web server's root directory. Prado applications presuppose a specific directory structure in which only a central index file is accessible. The actual code resides under the *protected* directory, which is blocked by the *.htaccess* file for the web server.

A command-line PHP script is started by the developer from the directory in which the application resides, using the name of the application as a parameter. This script then creates the necessary directory structure.

A pair of files in the *protected/pages* directory – called *Applicationname.page* and *Applicationname.php* – represents the view and controller components of the new application. Listings 1 and 2 show the contents for the Advent calendar application.

The view component (Listing 1) defines an HTML form in the first line. Lines 8-14 create a door of the calendar. The template tag *<com:TImageButton ... >* (line 9) links in the picture of the closed doors.

A *div* block with the CSS Style *float: left* attribute ensures that the compartments of the Advent calendar line up correctly. Template files can contain both Prado tags (recognizable by the prefix *<com:*) and normal HTML tags.

Multiplicator

An additional *div* tag below the picture centers the date text. The included PHP code is responsible for the text, which changes by day. The framing tags *<%# ... %>* tell Prado the included text is PHP source code and not template code.

To understand this, it is necessary to take a look at the *<com:TRepeater>* tag in line 4. A repeater tag, borrowed from Microsoft ASP.Net, resembles a *foreach()* loop. That is, it iterates the enclosed text according to the number of data records from the data source linked in from the controller code. The variable *$this* in this context references the repeater.

Every Prado repeater provides the attribute *Data[]*, which is an array containing the data set from the correct iteration of the loop.

Prado repeaters offer more functionality than simple *foreach()* loops in PHP: Header and footer areas are displayed only once. Only the area between *<prop:ItemTemplate>* and *</prop:ItemTemplate>* is repeated by the framework based on the data set.

If a block is marked with *<prop:ItemAlternatingTemplate>*, the contents of *ItemTemplate* and *ItemAlternatingTemplate* alternate. This results in lines in a table that alternate color (see the first example in the Prado QuickStart Tutorial [4]). In the example, the repeater repeats lines 11-14.

Data Sources

Lines 23-27 (Listing 2) links the repeater and the data source. In both cases, *$this* references the current class, and thus the page object. Within a page object, the implemented methods are on par with Prado tags, which are indicated by the ID attribute. Thus, *$this->Repeater* points to the repeater, which is indicated in *Home.page* with the attribute *ID="Repeater"*. The attribute *DataSource* of the Prado class *Repeater* contains the data. The code assigns to it the return value of the *getData()* method.

More demanding web applications get their data from a database, which Prado links in through its Object Relational Mapper or the *ActiveRecord* calls. In the example, the data originated from an

Listing 1: View Component of the Advent Calendar

```
01 <!--- Form --->
02 <com:TForm>
03   <!--- Repeater ("Datagrid") --->
04   <com:TR ID="Repeater"
05       OnItemCommand="open_door" > <!--- all Events from the Repeater
   --->
06       <!--- Iterated code: Image, label and hidden field  --->
07       <prop:ItemTemplate>
08         <div style="float:left">
09           <com:TImageButton ID="door" ImageURL="door.png" />
10           <div align="center">
11             <com:TLabel ID="tbox"
   text="<%# $this->Data['text'] %>" />
12           </div>
13         </div>
14         <com:THiddenField ID="inside" data="<%#
   $this->Data['inside'] %> " />
15       </prop:ItemTemplate>
16       <!--- End Iterated code --->
17   </com:TRepeater>
18
19   <!--- Display area for the contents of the calendar --->
20   <div style="clear:left;border-width:2px; border-style:solid;
   background-color:#ffffff;
21   width:650px; height:100px; text-align:center; padding:20px;
   border-color:black;">
22     <com:TImage id="content"  /> <!--- Bild --->
23   </div>
24 </com:TForm>
```

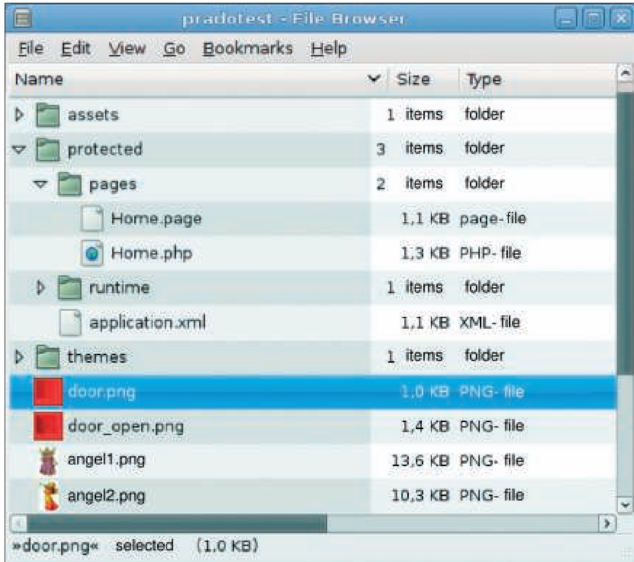

Figure 2: Always in pairs: Prado page consists of an XML template and a PHP file with the same name containing the application logic.

array. The condition *!$this->isPostBack* is only true when the application loads the page for the first time, not when it is re-loaded after a user action. The repeater thus starts via the *dataBind()* method with standard values only the first time the page is loaded. Because Prado components are typically stateful, after the initialization, the repeater remains in the same state, even when the page is loaded repeatedly. Session functionality is handled automatically by the framework, so no code is required in the view components or the controller.

The second method of the page class, *open_door()*, replaces the graphic of the closed door with an open door (Figure 1), which is done by exchanging the graphic of one instance of the door in the calendar that was replicated by the repeater (Listing 1, line 9).

If *open_door()* changes the URL, the changed variant remains constant each time the page is loaded. That is, once opened, the door of the calendar does not close again.

Who Points to Whom?

If the user clicks on a specific door, that specific door should open. One way of guaranteeing the correct reference is to differentiate the various instances of the calendar components by means of dynamically generated IDs that the *OnClick* event handler needs to receive as a parameter. However, Prado offers a simpler option: The event handler *OnItemCommand* (Listing 1, line 5) catches the events of all the control elements within the repeater.

Prado passes the object to the called function as the second parameter, called *$param* in this example. The method *getItem()* returns a reference to the loop it-

Listing 2: Model Component of the Advent Calendar

```
01 <?php
02
03 // Class for the home page
04 class Home extends TPage {
05
06 // This function provides the data
07 protected function getData()
08   {
09   return array( // text und image-URL for calendar-doors
10   array('text' => 'Sun 30th.', 'inside' => 'angel1.png'),
11   array('text' => 'Mon 1st', 'inside' => 'bells.png'),
12   array('text' => 'Tue 3rd', 'inside' => 'bug.png'),
13   array('text' => 'Wed 4th', 'inside' => 'ball.png'),
14   array('text' => 'Thu 5th', 'inside' => 'angel2.png'),
15   array('text' => 'Fri 6th', 'inside' => 'star1.png'),
16   array('text' => 'Sat 7th', 'inside' => 'star2.png'),
17 );
18
19
20   }
21 // Initialize Repeater (Data grid)
22 public function onLoad($param) {
23   if (!$this->isPostBack) { // only the first time
24     // Assign function as data source
25     $this->Repeater->DataSource=$this->getData();
26     $this->Repeater->dataBind(); // Initialize Repeater
27     }
28   }
29 // Change the image when the door is opened
30 public function open_door($sender, $param)
31   { //When the door is clicked, the eventhandler gets an object ...
32     $item=$param->getItem(); // ... that the contains the called object
33     $item->door->setImageURL('door_open.png'); // new image for the door
34     $image=$item->inside->getData(); // Image-URL from the Repeater ...
35     $this->content->ImageURL=$image; // ... show under the calendar
36   }
37 }
38 ?>
```

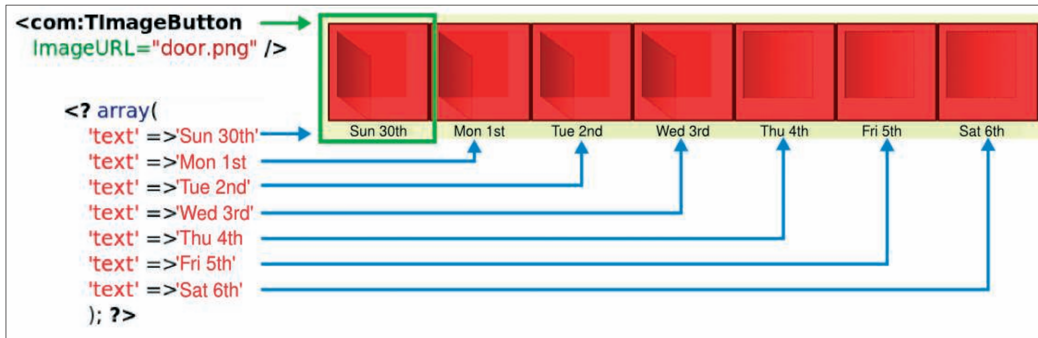

Figure 3: The Prado template language contains control structures that replicate display elements and supply the individual instances with data from arrays or lists.

eration in which the handler was called. In this way, one can refer to instances of the image object (created by the repetition) with the static ID *door* (Listing 1, line 9) by means of the return value of *getItem()* and the static ID. *$item->door->setImageURL()* always points to the graphic the user clicked. The method *setImageURL()* then exchanges the images.

During the loop iteration, repeaters pass the data to the linked objects, which they query with the call *$this->Data['Fieldname']*. After displaying the page, the data is no longer available in the repeater itself. However, the hidden field in line 14 (Listing 1) stores the passed values. This field can be addressed like the iterated images through the loop iteration object and its static ID. *$item->inside->getData()* reads its contents and *$this->content->ImageURL* sets the URL of the up-to-now-empty image tag *<com:TImage id="content" />* in line 22. The contents of the just-opened compartment appears under the Advent calendar.

A Wide Base

The Advent calendar example only gives a first impression of the extensive PHP framework. Prado supports localization, themes, and skins.

The Prado framework also provides access to SOAP web services, and Prado contains classes that ease error handling and integrate the SafeHTML Parser [5] to avoid XSS attacks.

Prado stores the state of the page in hidden form fields that the user can see and manipulate. As a countermeasure, depending on the configuration, Prado checks these forms against internal data fields and cookies based on keys stored on the server.

An integrated cache saves the parsing of the templates each time the page is loaded. With an extensive framework like Prado, PHP needs to load many Include files when a page is rendered. Relief is provided by the file *pra-dolight.php*: This file contains the complete framework in a single file, which, freed of all the code comments, is only about 2,500 KB.

The Advent calendar application does without Ajax functionality, which reloads the page with each state change. Help arrived with Prado 3.1.0 in mid-2007, which introduced Active Controls. Handling these controls is not always simple. When they are inserted individually into a template by the developer as static elements, with the exception of the saved page reload, they behave like classic Prado tags, which require a reload of the page for an update.

Is It Ripe?

The going gets difficult if the developer wants to combine Active Controls with repeaters and his own event handlers. These features do not function "out of the box" like static control elements. Instead, the developer must often initiate asynchronous server contact manually. How this works is not documented systematically anywhere on the Prado website. Many links on the tutorial pages lead nowhere. Only a search with a little luck in the forum provided the necessary information.

With approximately 4,000 registered users, the Prado forum offers valuable assistance for developers. However, the forum cannot replace one systematic document, especially for newbies. For the most part, what exists consists of automatically created documentation of the class framework. In some cases, one or only a couple of sentences explain the function of the class.

A QuickStart Tutorial is also available. The tutorial describes the most important aspects of Prado and contains an example application. To solve specific problems, especially with the Ajax elements, the tutorial does not go deep enough.

In addition to weaknesses related to the Ajax functionality that has only been available for a year and a half, the documentation clouds the otherwise overall positive impression of Prado. In contrast, Ohloh.net [7] certifies that the framework has a relatively large team with 11 core developers, in addition to well-commented source code.

Alternatives

Prado is an extensive framework that facilitates the most important areas of application development in PHP. With its understandable and efficient template language and its database abstraction layer, Prado offers a stable MVC architecture. However, the Ajax elements (alias Active Controls) are not well integrated into the framework yet. Alternatives to Prado, such as Cakephp [8] or SilverStripe [9], come with a CMS as well as an application framework. ■

INFO

[1] Prado: *http://www.pradosoft.com*

[2] Database connections: *http://www.pradosoft.com/demos/quickstart/?page=Database.DAO*

[3] Active Records: *http://www.pradosoft.com/demos/quickstart/?page=Database.ActiveRecord*

[4] Repeater: *http://www.pradosoft.com/demos/quickstart/?page=Controls.Repeater*

[5] SafeHTML: *http://pixel-apes.com/safehtml*

[6] Active Controls: *http://www.pradosoft.com/demos/quickstart/?page=ActiveControls.Home*

[7] Prado at Ohloh.net: *http://www.ohloh.net/projects/3182*

[8] Cakephp: *http://cakephp.org*

[9] SilverStripe: *http://www.silverstripe.com*