Understanding the Zend Framework, Part 5: Creating PDF files

Building the perfect reader

Skill Level: Intermediate

Nicholas Chase (nicholas@nicholaschase.com)
Developer/Writer
Studio B

01 Aug 2006

Updated 18 Jan 2011

In previous parts of this "Understanding the Zend Framework" series, you created the basic application, the Chomp online feed reader, using the open source PHP Zend Framework. In this tutorial, you use the Zend Framework's PDF capabilities to generate a PDF document based on entries the user has saved.

Section 1. Before you start

This tutorial is for developers who want to learn more about generating PDF files using the PHP Zend Framework. It discusses the overall framework and provides an example of generating new and edited documents. This functionality will be integrated into the existing Chomp feed reader created earlier in this series.

About this series

This "Understanding the Zend Framework" series chronicles the building of an online feed reader, Chomp, while explaining the major aspects of using the recently introduced open source PHP Zend Framework.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Part 1, talked about the overall concepts of the Zend Framework, including a list of relevant classes and a general discussion of the MVC pattern. Part 2 expanded on that to show how MVC can be implemented in a Zend Framework application. You also created the user registration and login process, adding user information to the database and pulling it back out again.

Parts 3 and 4 dealt with the actual RSS and Atom feeds. In Part 3, you enabled users to subscribe to individual feeds and to display the items listed in those feeds. You also discussed some of the Zend Framework's form-handling capabilities, validating data, and sanitizing feed items. Part 4 explained how to create a proxy to pull data from a site that has no feed.

The rest of the series involves adding value to the Chomp application. Here in Part 5, you will look at using the PDF format as a type of backup for saved entries. Part 6 uses the `Zend_Mail` module to alert users to new posts. Part 7, looks at searching saved content and returning ranked results. In Part 8, you create your own mashup, adding information from Amazon, Flickr, Twitter and Yahoo! And in Part 9, you add Ajax interactions to the site using JavaScript object notation.

About this tutorial

In earlier parts of this "Understanding the Zend Framework" series, you created the basic Chomp online feed reader.

In this tutorial, you will provide a way for the user to create a PDF file of his favorite feed entries. The user can then download the PDF as a backup or as a more convenient way to print out multiple entries to read later.

You'll start by creating a basic PDF document using the `Zend_PDF` component of the Zend Framework, enabling you to see the general concepts behind creating a document, adding text and graphics, and more. You will then look at integrating this functionality with the application.

To do that, you will first make it possible to save descriptions to the savedentries table, which previously held only full text (web page) entries. This will make them available for adding to the PDF document. You will then look at creating a new PDF document from within the feed controller and adding actual live data to it. You will also look at solving several issues that arise from the lack of features that are often taken for granted, such as word wrapping at the end of a line.

In this tutorial, you will take the existing code from Part 4 and enable the user to save his existing saved entries in a single PDF document, which can be read offline or printed.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Prerequisites

This tutorial assumes that you have a good familiarity with PHP programming and at least a basic understanding of how the Model-View-Controller (MVC) pattern works (see Resources). You do not have to know what the format of a PDF document looks like; the whole point of the `Zend_PDF` component is to insulate you from that depth of involvement with your data.

See Part 2 of this series for details on installing the Zend Framework and XAMPP, the easy-to-install version of Apache, MySQL, and PHP rolled into one.

Section 2. Creating a basic PDF

Let's start by creating a basic PDF file outside of the main application so you can see what's going on.

What is a PDF?

It's hard to spend much time on the Internet without running into a PDF document at some point. Created by Adobe Systems, the Portable Document Format (PDF) was designed to be just that: portable. PDF files can be read on any system for which the Adobe Acrobat Reader (or a reasonable facsimile) is available, and for the most part, it will look the same wherever you view it.

Interestingly, while these are not files that you typically edit, they are actually text-based, embedded binary data. For example, consider the snippet in Listing 1 of the actual PDF that you'll generate in this tutorial.

Listing 1. A sneak peek at the actual PDF

```
%PDF-1.4
...
1 0 obj
<<;/Type /Catalog /Version /1.4 /Pages 2 0 R >>
endobj
2 0 obj
<</Type /Pages /Kids [3 0 R ] /Count 1 >>
endobj
3 0 obj
<</Type /Page /LastModified (D:20060526190953-04'00')
/Resources <</ProcSet [/PDF /Text ] /XObject <</X1 5 0 R >>
/Font <</F1 6 0 R >> >>
/MediaBox [0 0 612 792 ] /Contents [4 0 R ] /Parent 2 0 R >>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
endobj
4 0 obj
<</Length 134 >>
stream
q
1 0 0 1 36 684 cm
72 0 0 72 0 0 cm
/X1 Do
Q
/F1 32 Tf
0.2 g
0.9 0 0 RG
3 w
BT
138 708 Td
(Chomp! To go) Tj
ET
18 774 576 -756 re
S

endstream
endobj
5 0 obj
...
```

As you can see, the file specifies various objects and their attributes. Fortunately, you do not have to get into the actual details of each of these objects because the `Zend_PDF` component of the Zend Framework takes care of it for you. Let's see how it works.

Create the document

Start by creating a document called createPDF.php. You may find it easiest to create this file in the <ZEND_HOME>/library directory, but the actual location is not important as long as it can find the Zend class files. Add the code in Listing 2.

Listing 2. Create the document

```
<?php

require_once 'Zend/Pdf.php';

$pdf = new Zend_Pdf();

$pdf->save("chomp.pdf");

?>
```

Here you include the `Zend_PDF` class files and use them to create a new PDF object. Once you have the object, you are simply saving it out to an arbitrary file.

You can execute this file by typing **php createPDF.php** on the command line. You should see a new file in the directory called chomp.pdf, but if you try to open it, you will get an error because it has no content. Let's take care of that next.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Add a page

Each page in a PDF document is an object unto itself. You can create a page of any size, using x and y values measured in points (1/72 of an inch), or you can use one of the four predetermined sizes, as you can see in Listing 3.

Listing 3. Adding a page

```
<?php
require_once 'Zend/Pdf.php';

$pdf = new Zend_Pdf();
$page = new Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER);
$pageHeight = $page->getHeight();
$pageWidth = $page->getWidth();

echo 'Height = '.$pageHeight.'\n';
echo 'Width = '.$pageWidth.'\n';

$pdf->pages[0] = ($page);

$pdf->save("chomp.pdf");

?>
```

Start by creating a new page as a standard letter-size page. The other choices are `Zend_Pdf_Page::SIZE_LETTER_LANDSCAPE`, `Zend_Pdf_Page::SIZE_A4`, and `Zend_Pdf_Page::SIZE_A4_LANDSCAPE`. Note that you can have different size pages in a single document.

Once you create the page, you can retrieve its height and width, but it still doesn't actually belong to the document. (An alternate means of creating a page, the `$pdf->newPage()` method, does belong to the document, and it reportedly has slightly better performance, but it creates a page that cannot be shared between documents.) To attach it to the document, you add it to the pages attribute, an array of page objects you can also manipulate like any other array.

Add an image

Start by adding an image to the document. As of this writing, the Zend Framework supports only JPEG images, but that is likely to change.

You start by creating an image object, as shown in Listing 4.

Listing 4. Adding an image

```
<?php
require_once 'Zend/Pdf.php';
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
$pdf = new Zend_Pdf();
$page = new Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER);
$chompImage = new Zend_Pdf_Image_JPEG::imageWithPath(
                      dirname(__FILE__) . '/chomp.jpg');

$pageHeight = $page->getHeight();
$pageWidth = $page->getWidth();
$imageHeight = 72;
$imageWidth = 72;

$topPos = $pageHeight - 36;
$leftPos = 36;
$bottomPos = $topPos - $imageHeight;
$rightPos = $leftPos + $imageWidth;

$page->drawImage($chompImage, $leftPos, $bottomPos, $rightPos, $topPos);

$pdf->pages[0] = ($page);

$pdf->save("chomp.pdf");

?>
```

Note: If you do not have the image extension enabled for PHP, you will get an error when you try to execute this file. To solve this problem, open your php.ini file and uncomment the line that says `extension=php_gd2.dll`.

Creating the actual image object is self-explanatory. You are simply loading the JPEG from a file. But from there, you have to think about positioning, which might not be entirely obvious.

A PDF document uses the same conventions as a PostScript file. That means everything is measured in "points," rather than pixels. There are 72 points to an inch, so if you want the image to be 1 inch high by 1 inch wide, you set both values to 72.

As far as the actual positioning, the origin for the coordinate system is in the bottom left-hand corner of the page. In other words, the point 0,0 corresponds to the lower left-hand corner, with coordinates going up as you get higher on the page and further to the right.

So, to place an object half an inch from the top of the page, you need to set that coordinate to the total height of the page, minus 36 points, or half an inch. Similarly, the bottom of the image will be at that point, minus the height of the image. Also, because you want it in the left-hand corner, you start with the coordinate of 36, then add the width of the image for the second coordinate.

Once you have all of that information, you can use the `drawImage()` method to add it to the actual page. The process of adding text is similar.

Add text

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Before you can add any text to the page, you need to determine what it's going to look like. You can accomplish that through the use of styles, as shown in Listing 5.

Listing 5. Adding text

```
...
$topPos = $pageHeight - 36;
$leftPos = 36;
$bottomPos = $topPos - $imageHeight;
$rightPos = $leftPos + $imageWidth;

$page->drawImage($chompImage, $leftPos, $bottomPos, $rightPos, $topPos);

$style = new Zend_Pdf_Style();
$style->setLineColor(new Zend_Pdf_Color_RGB(0.9, 0, 0));
$style->setFillColor(new Zend_Pdf_Color_GrayScale(0.2));
$style->setLineWidth(3);
$style->setFont(
            new Zend_Pdf_Font::fontWithName(Zend_PDF_Font::FONT_HELVETICA_BOLD) ,32);

$page->setStyle($style);
     ->drawText('Chomp! To go', $rightPos + 32, $topPos - 48);

$pdf->pages[0] = ($page);

$pdf->save("chomp.pdf");

?>
```

Start by creating a style object and setting its attributes. The line color, here set as an RGB value (with the red, green, and blue values being set on a scale of zero to one), mostly applies to shapes, as you'll see in a moment. The fill color, here set as a very dark gray, also applies to shapes, but more importantly, in this case, sets the color of your text. The line width determines the width of your lines. Then you set the font and the size, in points. Fourteen standard fonts are included in PDF documents and can be referenced as `Zend_PDF_Font` values. They include `Zend_Pdf_Font::FONT_TIMES_ROMAN`, `Zend_Pdf_Font::FONT_TIMES_BOLD`, `Zend_Pdf_Font::FONT_TIMES_ITALIC`, `Zend_Pdf_Font::FONT_TIMES_BOLDITALIC`, the same four varieties for Helvetica and Courier, and the Symbol and Zapf Dingbats fonts.

Once you create the style, you can set it for the page. Finally, you are ready to actually add the text to the page, specifying the text itself and its position. In this case, you want the text to be half an inch to the right of the image, and two-thirds of an inch from the top of the page.

Add a shape

The final touch to your sample page is a border around the outside (see Listing 6).

Listing 6. Adding a shape

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
...
$style->setFont(new
 Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::FONT_HELVETICA_BOLD), 32);

$page->setStyle($style);
    ->drawText('Chomp! To go', $rightPos + 32, $topPos - 48);
        ->drawRectangle(18, $pageHeight - 18, $pageWidth - 18,
                           18, Zend_Pdf_Page::SHAPEDRAW_STROKE);

$pdf->pages[0] = ($page);

$pdf->save("chomp.pdf");

?>
```

You already set a color and width of the line that makes up the rectangle, so now you're setting the top-left and bottom-right corners to be a quarter of an inch from the edges of the page, and specifying that you want just the line, as opposed to a filled shape. You have three options: `Zend_Pdf_Page::SHAPEDRAW_FILLNSTROKE`, `Zend_Pdf_Page::SHAPEDRAW_STROKE`, and `Zend_Pdf_Page::SHAPEDRAW_FILL`. The first includes a line and a fill, so if you had used it here, you would've wound up with a dark gray rectangle with a red border.

Let's look at actually implementing this within the application.

Section 3. Integrating with the application

Now that you have the general idea, let's look at what's involved in integrating this functionality with the actual application.

Adding descriptions to the view

Because you want the PDF to show both the title and description of your saved entries, you will need to save those descriptions in the database. At the moment, information only gets saved to the database if the user checks the *full text* option, and in that case, you are looking at an HTML page you don't necessarily want to include in a PDF document.

Start by adding the actual descriptions to viewChannel.php (see Listing 7).

Listing 7. Adding descriptions to viewChannel.php

```
<html>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
<head>
    <title><?php echo $this->escape($this->title);
 ?></title>
</head>
<body>
  [<a href='/'>Back to Main Menu</a>]<br>
  <h1><?php echo $this->escape($this->title); ?></h1>

  <table>
    <tr>
      <td>Channel Title (click to read)
    \
    </td>
      <td>Save entry to database
        amp;nbsp;amp;nbsp;amp;nbsp;\
        amp;nbsp;amp;nbsp;amp;nbsp;</td>
      <td>Save Full Text</td>
    </tr>
  <?php
     $feedTitle = $this->title;
     foreach ($this->rssFeed as $item) {
         $entryTitle = $item->title();
         $link = $item->link();
         $description = $item->description();
         echo "<form method='POST' action='/feed/saveEntry'>";
         echo "<input type='hidden' name='feedTitle'" .
              " value='$feedTitle'/>";
         echo "<input type='hidden' name='title' " .
              " value='$entryTitle'/>";
         echo "<input type='hidden' name='link' \
         value='$link'/>";
         echo "<input type='hidden' name='type' \
         value='rssFeed'/>";
         echo "<input type='hidden' name='description' ".
              " value='".addslashes\
              (strip_tags($description))."'/>";
         echo "<tr><td><a
 href='$link'>$entryTitle</a><br>";
         echo "$description</td>";
         echo "<td><input type='submit'
 value='save'/></td>";
         echo "<td><input name='saveFullText' " .
              " type='checkbox'/></td></tr></form>";
     }
?>
  </table>
</body>
</html>
```

The first change is to actually retrieve the description from the item itself. Once you have that, you want to add it to two places.

The first is a hidden form element (so that it is available when the user submits the form to save the item). To do that, you need to make sure you remove any possibility that the description will make this tag malformed. In other words, you need to remove any HTML tags and make sure any quotes in the text are preceded by backslashes and, thus, rendered harmless. To do all that, you apply the `php strip_tags()` function, then the `addslashes()` function.

Finally, you can display the description itself on the page for the convenience of users. In this case, you don't care about removing the HTML tags because you are displaying it in HTML, anyway.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Saving the description

Now that you have the description in the form, you need to modify the `saveEntryAction()` function to act on it. Open the FeedController.php file and make the changes shown in Listing 8.

Listing 8. Saving descriptions to the database

```
...
    public function saveEntryAction()
    {
        $filterSession = Zend_Registry::get('fSession');
        $username = $filterSession->getRaw('username');

        $filterPost = Zend_Registry::get('fPost');
        $feedTitle = $filterPost->getRaw('feedTitle');
        $channelTitle = $filterPost->getRaw('title');
        $channelLink = $filterPost->getRaw('link');
        $type = $filterPost->getRaw('type');
        $saveFullText = $filterPost->getRaw('saveFullText');
        $description = $filterPost->getRaw('description');

        if($saveFullText){
            $http = new Zend_Http_Client($channelLink);
            $response = $http->get();
            if ($response->isSuccessful())
                $fullText = $response->getBody();
            else{
                echo 'Error occurred, full text not saved, please reload.';
                return;
            }
        }

        $db = Zend_Registry::get('db');
        $row = array(
                     'Username' => $username,
                     'feedname' => $feedTitle,
                     'channelname' => $channelTitle,
                     'link' => $channelLink,
                     'entrysaved' => $saveFullText ? 'true' : 'false',
                     'entrydata' =>
                          $saveFullText ? $fullText : $description
                     );

        $table = 'savedentries';
        $rowsAffected = $db->insert($table, $row);

        if($type == 'webPage')
            $this->_redirect("/");
        else
            $this->_redirect("/feed/viewChannel?title=$feedTitle");
    }
...
```

The process is straightforward: You retrieve the description value from the form, and use it to populate the `entrydata` item in the update row array. In this case, you want to use this value only if the user did not select the `fullText` option.

Now you're ready to start dealing with the PDF itself.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Creating the new action

From the MVC standpoint, it's a toss-up as to whether creating the PDF should be part of the user, the index, or the feed. Here, you make the arbitrary decision to make it part of the feed. To that end, add the following function to the FeedController.php file, as shown in Listing 9.

Listing 9. Adding the createPdfAction() to FeedController.php

```
...
    public function createPdfAction()
    {
        $filterSession = Zend::registry('fSession');
        $username = $filterSession->getRaw('username');

    }
...
```

It is just a simple function, accessible from the browser. Start by retrieving the session from the registry and using it to get the current username.

Adding saved entries

Let's look at retrieving the information that will ultimately wind up in the PDF (see Listing 10).

Listing 10. Adding saved entries

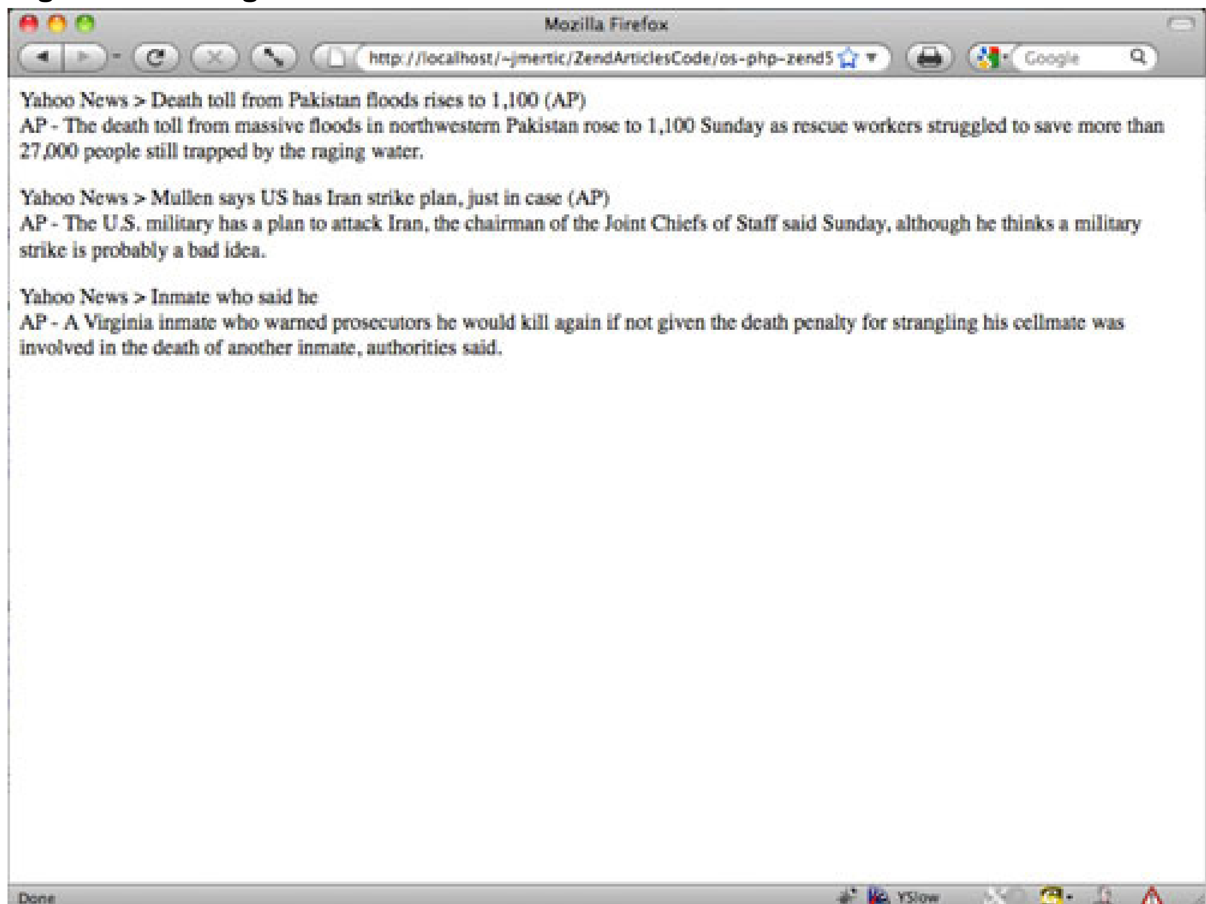
```
...
public function createPdfAction()
    {
        $filterSession = Zend_Registry::get('fSession');
        $username = $filterSession->getRaw('username');

        $db = Zend_Registry::get('db');
        $select = $db->select();
        $select->from('savedentries', '*');
        $select->where("username=?", $username);
        $sql = $select->__toString();
        $entries = $db->fetchAll($sql);
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }
            echo '<p>'.$title.'<br />';
            echo $entrydata.'</p>';

        }
    }
...
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

First, retrieve the database connection for the registry, then create a `select` statement that retrieves all the rows in the savedentries table for the current username. Execute the query, retrieving the title and data for each entry, and adding a channel name to the title if available.

If you view the results of this function by pointing the browser to `http://localhost/feed/createPdf`, you should see results similar to those shown in Figure 1

Figure 1. Adding saved entries



Creating the new document

Creating the PDF involves the same steps in the sample document (see Listing 11).

Listing 11. Creating the new document

```
...
    public function createPdfAction()
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
    {

        require_once 'Zend/Pdf.php';

        $pdf = new Zend_Pdf();
        $page = new Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER);

        $filterSession = Zend_Registry::get('fSession');
        $username = $filterSession->getRaw('username');

        $db = Zend_Registry::get('db');
        $select = $db->select();
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }
            echo '<p>'.$title.'<br />';
            echo $entrydata.'</p>';

        }
        $pdf->pages[0] = ($page);
    }
...
```

You're creating the new PDF document and a single page, then adding the page to the document after all processing has been completed.

Adding interface items

Adding the image, text, and borders should also look familiar (see Listing 12).

Listing 12. Adding a logo

```
...
    public function createPdfAction()
    {

        require_once 'Zend/Pdf.php';

        $pdf = new Zend_Pdf();
        $page = new Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER);
        $chompImage = new
                Zend_Pdf_Image::imageWithPath('E:\sw\public_html\chomp.jpg');

        $pageHeight = $page->getHeight();
        $pageWidth = $page->getWidth();
        $imageHeight = 72;
        $imageWidth = 72;

        $topPos = $pageHeight - 36;
        $leftPos = 36;
        $bottomPos = $topPos - $imageHeight;
        $rightPos = $leftPos + $imageWidth;

        $page->drawImage($chompImage, $leftPos, $bottomPos,
                                         $rightPos, $topPos);

        $style = new Zend_Pdf_Style();
        $style->setLineColor(new Zend_Pdf_Color_RGB(0.9, 0, 0));
        $style->setFillColor(new Zend_Pdf_Color_GrayScale(0.2));
        $style->setLineWidth(3);
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
        $style->setFont(Zend_Pdf_Font::fontWithName(
                            Zend_Pdf_Font:FONT_HELVETICA_BOLD), 32);

        $page->setStyle($style);
              ->drawText('Chomp! To go', $rightPos + 32, $topPos - 48);
              ->drawRectangle(18, $pageHeight - 18, $pageWidth - 18,
                                  18, Zend_Pdf_Page::SHAPEDRAW_STROKE);

        $filterSession = Zend_Registry::get('fSession');
        $username = $filterSession->getRaw('username');

        $db = Zend_Registry::get('db');
        $select = $db->select();
        $select->from('savedentries', '*');
...
```

Note that in this case, you set a specific location for the image so the application can find it.

Outputting the PDF

Outputting the PDF is a little different from in the sample file because part of the setup for the Zend Framework involves making sure that all requests go to the index.php file. That means you can't simply save the file to the server and have the user download. (Yes, there are server configuration tweaks you can make, but you're trying to keep this simple.)

The alternative is to simply output the PDF to the user's browser after it's been generated (see Listing 13).

Listing 13. Outputting the PDF

```


...
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }
//            echo '<p>'.$title.'<br />';
//            echo $entrydata.'</p>';

        }

        $pdf->pages[0] = ($page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Starting at the bottom, the `render()` function outputs the actual text that is the PDF document, but for the browser to interpret it correctly, it needs to know the text

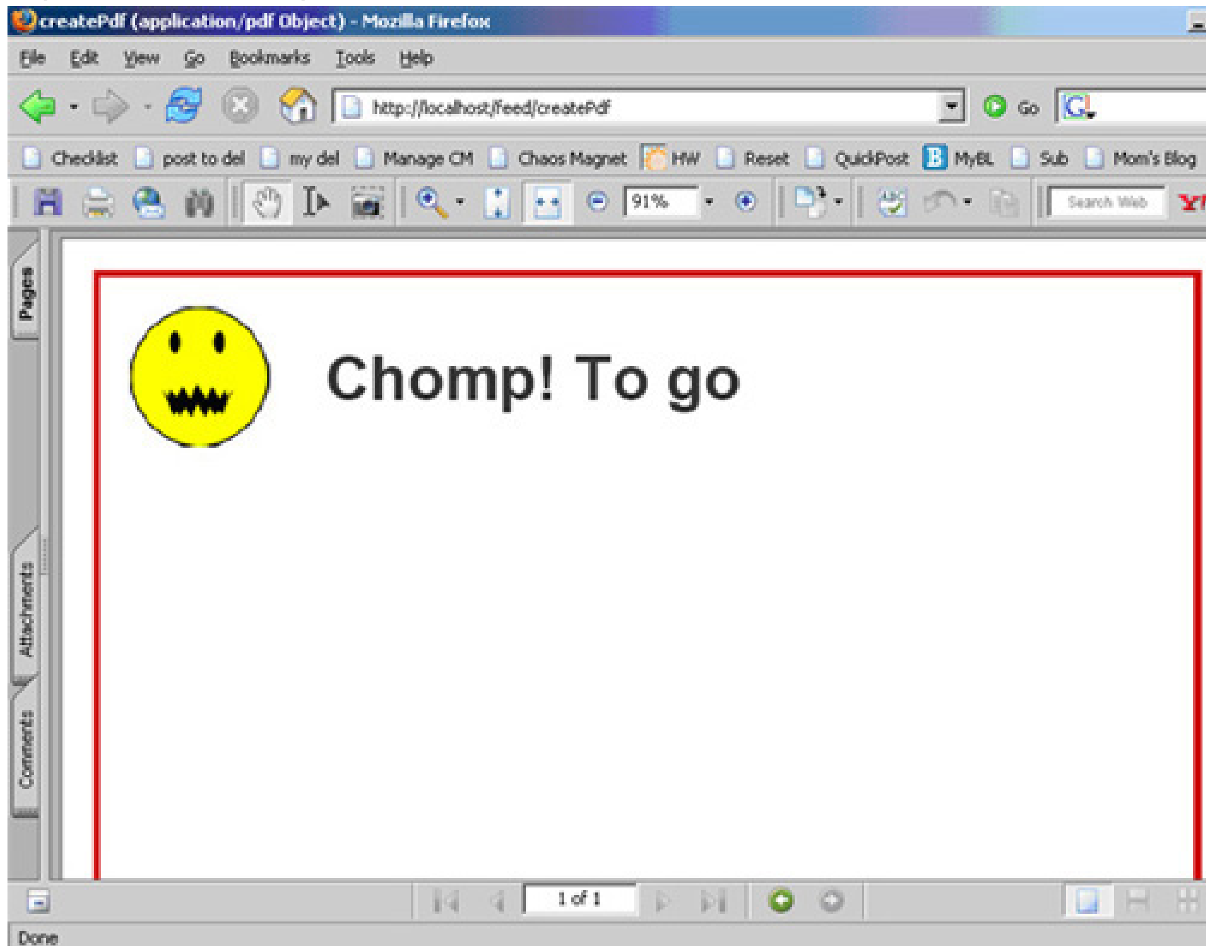
© Copyright IBM Corporation 2006, 2011. All rights reserved.

consists of a PDF file. Normally, when the browser downloads a PDF, the server sends the content type based on the file extension. Since there is no file, there is no file extension, so you have to set it manually using the headers.

Before you can use the headers, you must make sure that nothing gets output before them, so you remove the statements that were previously sent to the page.

If you refresh the browser (see Figure 2), you should see the actual PDF document (assuming you have Adobe Acrobat Reader installed).

Figure 2. Outputting the PDF



Section 4. Adding text

You have the decorations in place, so it's time to add the text.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Displaying the headlines

To add headlines to the page, you first create a new style, as shown in Listing 14.

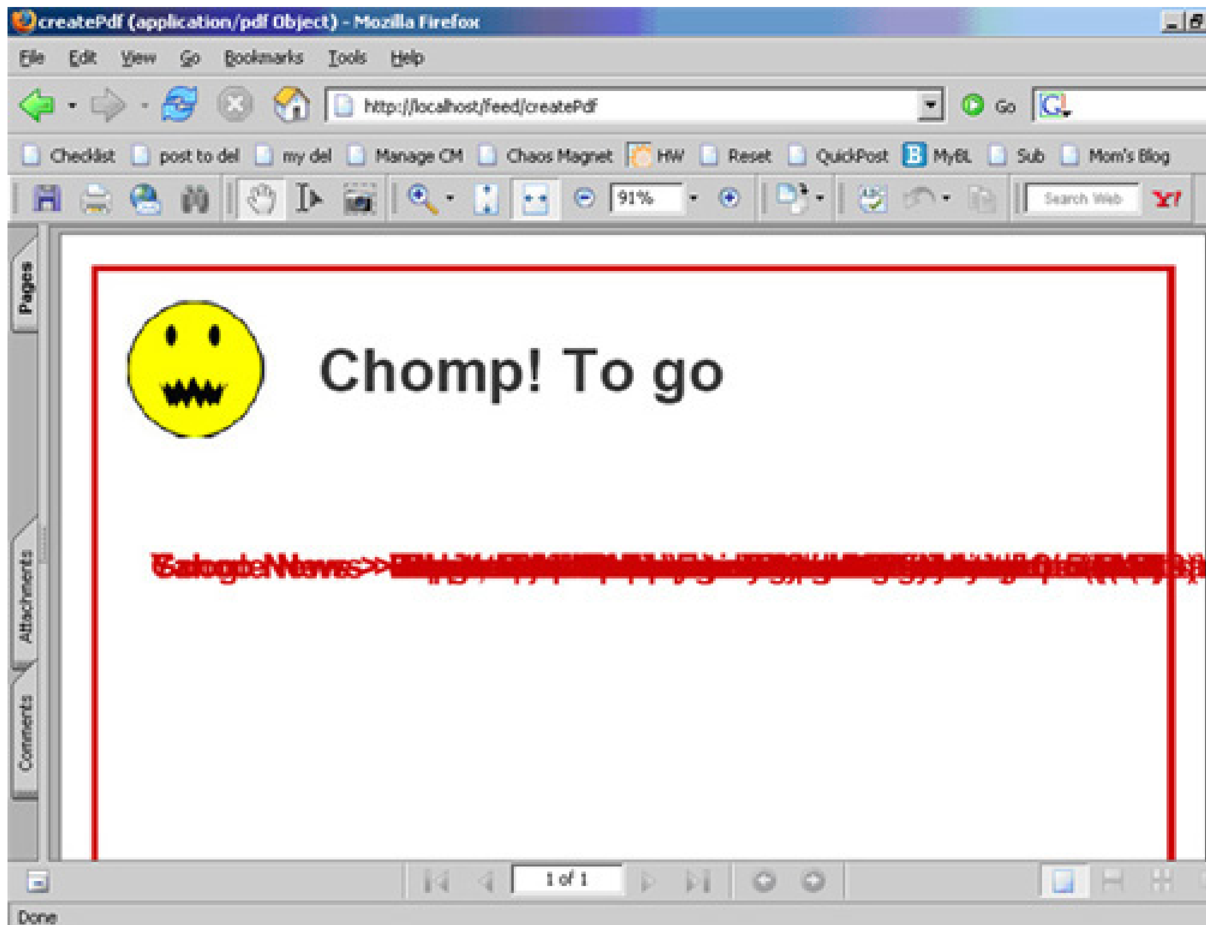
Listing 14. Displaying headlines

```
...
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(
                   new Zend_Pdf_Color_RGB(0.9, 0, 0));
            $headlineStyle->setFont(
                   new Zend_Pdf_Font_Standard(
                        Zend_Pdf_Const::\
                        FONT_HELVETICA_BOLD), 18);

            $page->setStyle($headlineStyle);
                 ->drawText($title, 48, $topPos - 148);

        }

        $pdf->pages[0] = ($page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

The creation of the style mirrors what you did before, with a few small differences. For one thing, you set the fill color to red, and you're using 18-point type instead of 36-point type. Notice also that you can set the style four-page, multiple times; the most recently set style takes precedence. Finally, you add the text. If you render the PDF, the results might not be quite what you expect (see Figure 3).

Figure 3. Displaying the headlines

© Copyright IBM Corporation 2006, 2011. All rights reserved.



As far as the PDF is concerned, you placed all the text in the same position, so rather than placing it sequentially, it just followed your instructions. You'll have to do something about that.

Positioning lines

To prevent the lines of text from overlapping, you need to calculate how far apart you want them and update the position information (see Listing 15).

Listing 15. Preventing overlap

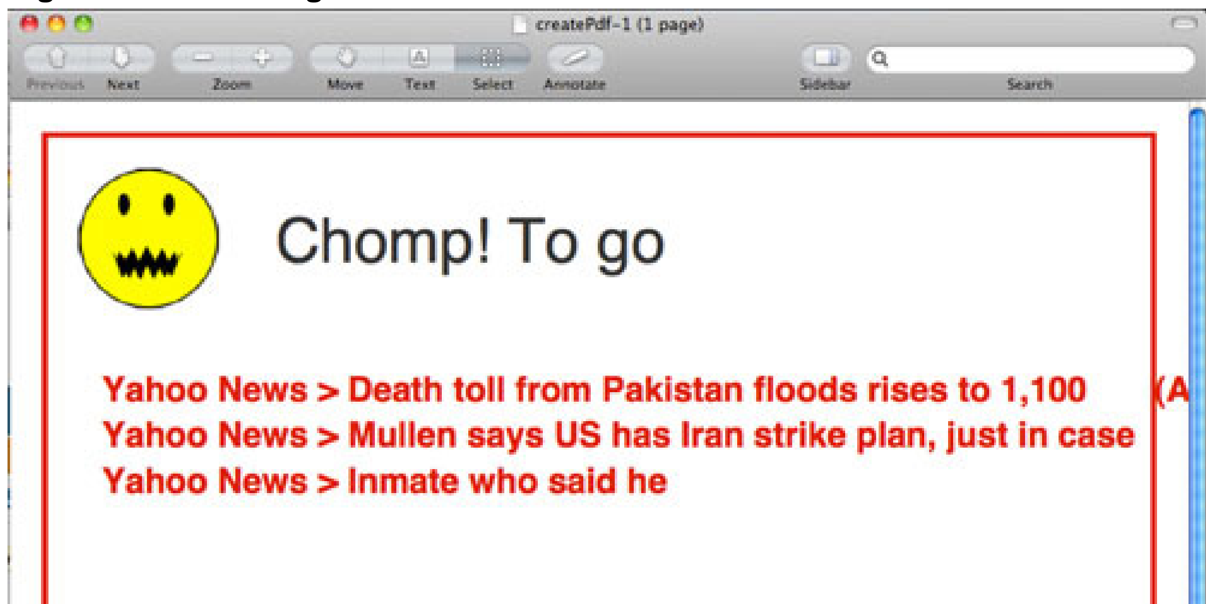
```
...
        $db = Zend_Registry::get('db');
        $select = $db->select();
        $select->from('savedentries', '*');
        $select->where("username=?", $username);
        $sql = $select->__toString();
        $entries = $db->fetchAll($sql);

        $startPos = $topPos - 120;

        foreach($entries as $row){
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(
                  new Zend_Pdf_Color_RGB(0.9, 0, 0));
            $headlineStyle->setFont(
                  new Zend_Pdf_Font::fontWithName(
                        Zend_Pdf_Font::HELVETICA_BOLD), 18);

            $page->setStyle($headlineStyle);
                 ->drawText($title, 48, $startPos);
            $startPos = $startPos - 24;
        }

        $pdf->pages[0] = ($page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Because the position is set explicitly, all you have to do is track it. You start with the position 120 points (just under two inches) below the top of the page and move the position down 24 points each time you output a line of text. Still, the output is not quite ideal (see Figure 4).

Figure 4. Positioning lines



You still have to figure out how to manually perform a line wrap for lines that are too long for the page.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Breaking text into lines

To wrap text at the edge of the page, you need to break a single line down into multiple lines and display each of them separately (see Listing 16).

Listing 16. Wrapping lines of text

```
...
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(
                   new Zend_Pdf_Color_RGB(0.9, 0, 0));
            $headlineStyle->setFont(
                   Zend_Pdf_Font::fontWithName(
                        Zend_Pdf_Font::HELVETICA_BOLD), 18);

            $page->setStyle($headlineStyle);
            $title = strip_tags($title );
            $title = wordwrap($title , 55, '\n');

            $headlineArray = explode('\n', $title );

            foreach ($headlineArray as $line) {
                $line = ltrim($line);
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 24;
            }

        }

        $pdf->pages[0] = ($page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Starting at the top, you take the title and make sure it does not include any HTML tags. Once you know that, you can use the PHP `wordwrap()` function to break the text down to lines that are no more than a specific length — in this case, 55 characters. The `wordwrap()` function simply inserts a character (in this case, the newline character) where the line would normally wrap based on this length. In other words, if a word would cause the line to be longer than 55 characters, this function inserts the newline character before that word.

This, however, does not solve the entire problem because the newline character is rendered by the PDF as a space, not as a new line. However, with the newline character strategically placed throughout the string, you can use the `explode()`

© Copyright IBM Corporation 2006, 2011. All rights reserved.

function to turn this single string into an array of strings. Once you have the array, you can loop through each member, trimming any leading spaces and moving the start position 24 points down after each line.

The results are more like one would expect to see (see Figure 5).

Figure 5. Wrapping lines of text

Displaying body text

Adding the body text for each entry is virtually identical to adding the headlines (see Listing 17).

Listing 17. Displaying the body of the entry

```
...
            $headlineArray = explode('\n', $title );

            foreach ($headlineArray as $line) {
                $line = ltrim($line);
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 24;
            }

            $articleStyle = new Zend_Pdf_Style();
            $articleStyle->setFillColor(
                  new Zend_Pdf_Color_RGB(0, 0, 0));
            $articleStyle->setFont(
                  Zend_Pdf_Font::fontWithName(
                        Zend_Pdf_Font::HELVETICA_BOLD), 12);
            $page->setStyle($articleStyle);

            $entrydata = strip_tags($entrydata);
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            $entrydata = wordwrap($entrydata, 90, '\n');

            $articleArray = explode('\n', $entrydata);

            foreach ($articleArray as $line) {
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 16;
            }
            $startPos = $startPos - 16;

        }

        $pdf->pages[0] = ($page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Again, you first create the style, this time using black instead of red and 12-point type instead of 18-point type. Because the text is smaller, you can fit 90 characters per line, rather than just 55. Also, after you display the article text, you move the pointer down an additional 16 points to set it off from the next article (see Figure 6).

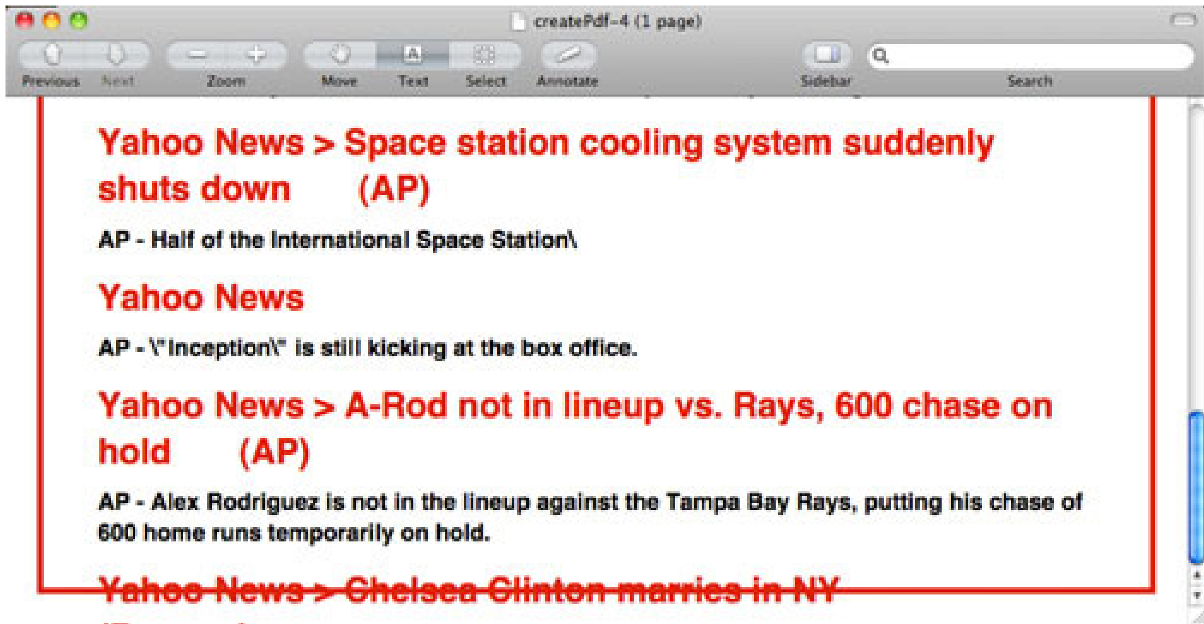
Figure 6. Displaying the body text

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Detecting the end of the page

Unfortunately, not all is well yet. Not only does the PDF not automatically wrap the text at the end of the line but it also does not automatically create a new page when you reach the end of this one (see Figure 7).

Figure 7. Detecting the end of the page



You're going to have to manually create a new page when you detect that you're getting close to the bottom of this one (see Listing 18).

Listing 18. Detecting the end of the page

```
...
        $sql = $select->__toString();
        $entries = $db->fetchAll($sql);

        $startPos = $topPos - 120;

        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            if ($startPos < 72){
                //start a new page
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(new Zend_Pdf_Color_RGB(0.9,
0, 0));
            $headlineStyle->setFont(
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
Zend_Pdf_Font::fontWithName(Zend_Pdf::HELVETICA_BOLD), 18);

           $page->setStyle($headlineStyle);
           $title = strip_tags($title );
           $title = wordwrap($title , 55, '\n');

           $headlineArray = explode('\n', $title );

           foreach ($headlineArray as $line) {
               $line = ltrim($line);
               $page->drawText($line, 48, $startPos);
               $startPos = $startPos - 24;
           }

           $articleStyle = new Zend_Pdf_Style();
           $articleStyle->setFillColor(new Zend_Pdf_Color_RGB(0, 0, 0));
           $articleStyle->setFont(
Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::HELVETICA_BOLD), 12);
           $page->setStyle($articleStyle);

           $entrydata = strip_tags($entrydata);
           $entrydata = wordwrap($entrydata, 90, '\n');

           $articleArray = explode('\n', $entrydata);

           foreach ($articleArray as $line) {

               if ($startPos < 48){
                    //start a new page
               }

               $page->drawText($line, 48, $startPos);
               $startPos = $startPos - 16;

           }
           $startPos = $startPos - 16;

       }

       $pdf->pages[0] = $page;

       header('Content-type: application/pdf');
       echo $pdf->render();

   }
...
```

Here you are performing two tests. First, before you display a new story, you make sure that there is at least an inch of page left. If not, you will start a new page. (You'll see how to do that in the moment.) Second, before you display each line of article text, you make sure that there is at least two-thirds of an inch left on the page.

Now let's look at actually creating the new page.

Creating a new page

Consider three issues when creating a new page: creating the page, adding the page to the document, and making sure everything appears properly (see Listing 19).

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Listing 19. Creating a new page

```
...
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            if ($startPos < 72){
                 array_push($pdf->pages, $page);
                 $page = new Zend_Pdf_Page(
                         Zend_Pdf_Page::SIZE_LETTER);
                 $startPos = $pageHeight - 48;
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(
                 new Zend_Pdf_Color_RGB(0.9, 0, 0));
            $headlineStyle->setFont(
                     Zend_Pdf_Font::fontWithName(
                     Zend_Pdf_Font::HELVETICA_BOLD), 18);

            $page->setStyle($headlineStyle);
            $title = strip_tags($title );
            $title = wordwrap($title , 55, '\n');

            $headlineArray = explode('\n', $title );

            foreach ($headlineArray as $line) {
                $line = ltrim($line);
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 24;
            }

            $articleStyle = new Zend_Pdf_Style();
            $articleStyle->setFillColor(
                 new Zend_Pdf_Color_RGB(0, 0, 0));
            $articleStyle->setFont(
                     Zend_Pdf_Font::fontWithName(
                     Zend_Pdf_Font::HELVETICA_BOLD), 12);
            $page->setStyle($articleStyle);

            $entrydata = strip_tags($entrydata);
            $entrydata = wordwrap($entrydata, 90, '\n');

            $articleArray = explode('\n', $entrydata);

            foreach ($articleArray as $line) {

                if ($startPos < 48){

                    array_push($pdf->pages, $page);
                     $page = new Zend_Pdf_Page(
                          Zend_Pdf_Page::SIZE_LETTER);
                     $articleStyle = new Zend_Pdf_Style();
                     $articleStyle->setFillColor(
                          new Zend_Pdf_Color_RGB(0, 0, 0));
                     $articleStyle->setFont(
                          Zend_Pdf_Font::fontWithName(
                          Zend_Pdf_Font::HELVETICA_BOLD), 12);
                     $page->setStyle($articleStyle);

                     $startPos = $pageHeight - 48;
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
                }
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 16;

            }
            $startPos = $startPos - 16;

        }

        array_push($pdf->pages, $page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Because the pages for the PDF document are simply an array, you can use the `array_push()` function to add the current page to the end of the document. You can then create a new page (using the same variable name for convenience) and reset the starting position to two-thirds of an inch below the top of the page.

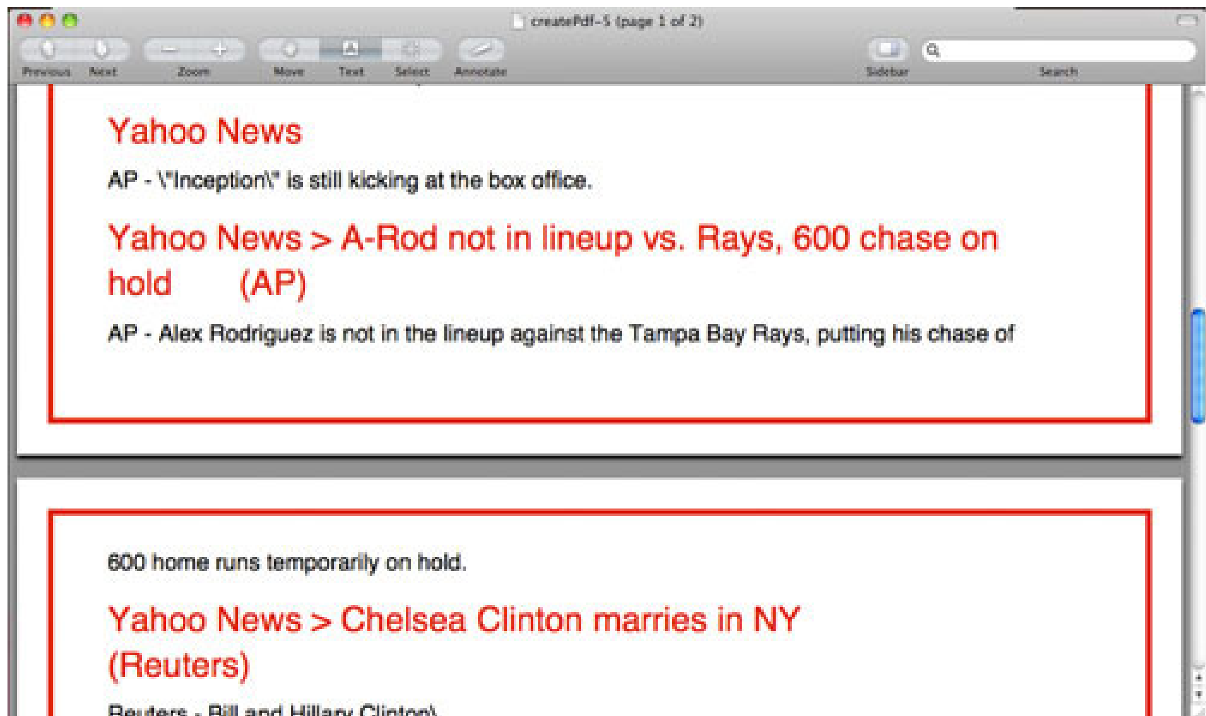
Now, that's sufficient for your headline check because you can simply set the style on the page after is created. For your check within the body of the article, it's a little more complicated because you may wind up in the situation in which you are creating a page in the middle of the array, and the new page won't have the `$articleStyle` style set properly. Therefore, in addition to adding the current page to the array and creating a new page, you also have to reset the style. Again, you reset the starting position to two-thirds of an inch below the top of the page.

Finally, you change the way the initial page gets added to the document. Before, you did all of your processing, then added the current page as the first page. In this case, you don't know if it is the first page, so you simply push it onto the array, instead.

The result is a properly paginated document, shown in Figure 8.

Figure 8. Creating a new page

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Although the page is properly paginated, it is missing a border.

Cleaning up the new pages

To make sure your pages are uniform, you can create a function and use it to create all of your pages (see Listing 20).

Listing 20. Creating a function for new pages

```
...
        $view->unsubFeeds = $unsubFeeds;
        echo $view->render('feeds.php');
    }

        private function newPdfPage(){

        $page = new Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER);

        $style = new Zend_Pdf_Style();
        $style->setLineColor(new Zend_Pdf_Color_RGB(0.9, 0, 0));
        $style->setFillColor(new Zend_Pdf_Color_GrayScale(0.2));
        $style->setLineWidth(3);
        $style->setFont(
                             Zend_Pdf_Font::fontWithName(
                     Zend_Pdf_Font::HELVETICA_BOLD), 32);

        $page->setStyle($style);

        $pageHeight = $page->getHeight();
        $pageWidth = $page->getWidth();

        $page->drawRectangle(18, $pageHeight - 18, $pageWidth - 18,
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
                                18, Zend_Pdf_Page::SHAPEDRAW_STROKE);

        return $page;

    }

    public function createPdfAction()
    {

        require_once 'Zend/Pdf.php';

        $pdf = new Zend_Pdf();
        $page = $this->newPdfPage();
        $chompImage = new Zend_Pdf_Image_JPEG(
                            'E:\sw\public_html\chomp.jpg');

        $pageHeight = $page->getHeight();
        $pageWidth = $page->getWidth();
        $imageHeight = 72;
        $imageWidth = 72;
...
        foreach($entries as $row){
            $title = $row['feedname'];
            $entrydata = $row['entrydata'];
            if($row['channelname'] != '')
            {
                $title = "$title > " . $row['channelname'];
            }

            if ($startPos < 72){
                 array_push($pdf->pages, $page);
                 $page = $this->newPdfPage();
                 $startPos = $pageHeight - 48;
            }

            $headlineStyle = new Zend_Pdf_Style();
            $headlineStyle->setFillColor(new Zend_Pdf_Color_RGB(0.9, 0,
 0));
            $headlineStyle->setFont(
 Zend_Pdf_Font::fontWithName(Zend_PdfFont::HELVETICA_BOLD), 18);

            $page->setStyle($headlineStyle);
            $title = strip_tags($title );
            $title = wordwrap($title , 55, '\n');
...
            $entrydata = strip_tags($entrydata);
            $entrydata = wordwrap($entrydata, 90, '\n');

            $articleArray = explode('\n', $entrydata);

            foreach ($articleArray as $line) {

                if ($startPos < 48){

                     array_push($pdf->pages, $page);
                     $page = $this->newPdfPage();
                     $articleStyle = new Zend_Pdf_Style();
                     $articleStyle->setFillColor(
                           new Zend_Pdf_Color_RGB(0, 0, 0));
                     $articleStyle->setFont(
                           Zend_Pdf_Font::fontWithName(
                   Zend_Pdf_Font::HELVETICA_BOLD), 12);
                     $page->setStyle($articleStyle);

                     $startPos = $pageHeight - 48;

                }
                $page->drawText($line, 48, $startPos);
                $startPos = $startPos - 16;
```

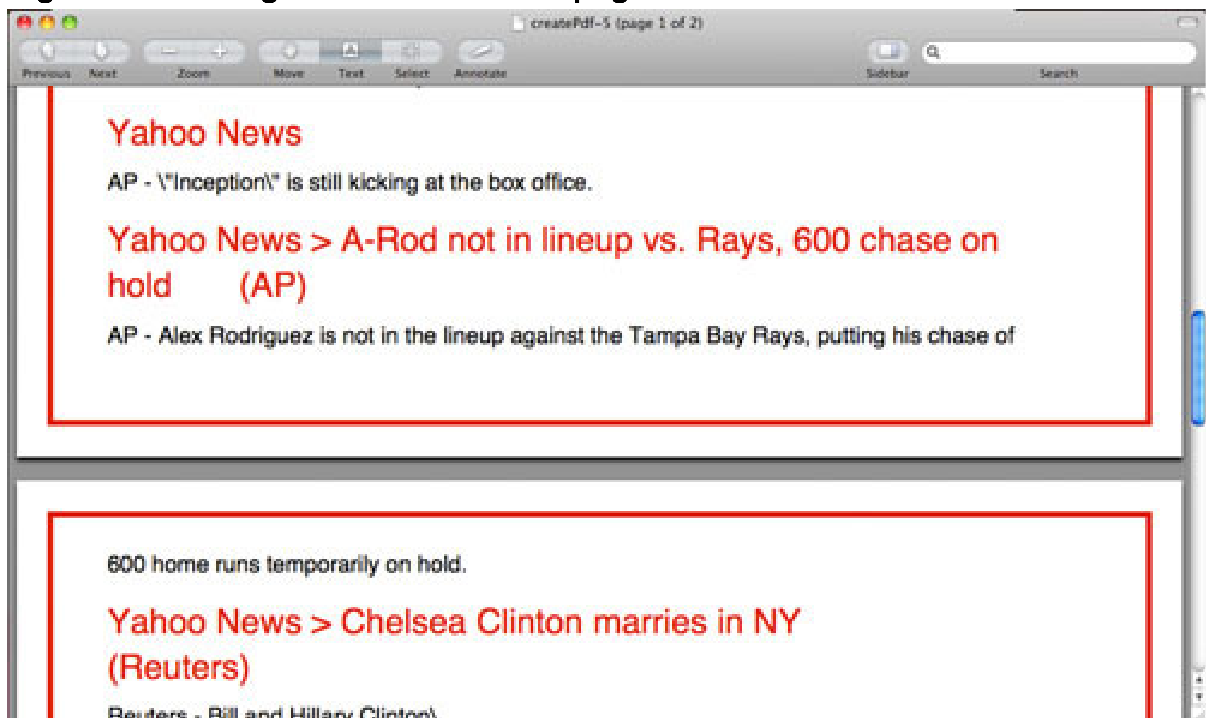

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            }
            $startPos = $startPos - 16;

        }

        array_push($pdf->pages, $page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

There really isn't any new code here. All you've done is rearrange things so that the `newPdfPage()` function includes all the common elements, such as the basic style and the border. You then replace each instance of creating a new page with a call to this function and add any additional processing (such as the image or headlines and articles) to the returned object.

Your pages are now uniform, as shown in Figure 9.

Figure 9. Creating a function for new pages



Section 5. Choosing items and reusing documents

© Copyright IBM Corporation 2006, 2011. All rights reserved.

The basic system is in place. You just need to add a couple of refinements. In this section, you'll enable users to choose which saved entries should be included in the document, and you'll enable them to add onto a previous file, rather than creating a new one.

Better item management: Adding an ID

Before you can deal with your entries in any reasonable way, you need a better way to reference them. In the first pass, you created a structure in which items were referenced by feed name, or sometimes by URL, but as you get more serious about data management, each item really needs a primary key you can reference.

To that end, log into MySQL and execute the command shown in Listing 21.

Listing 21. Adding a new column to the table

```
ALTER TABLE `savedentries` ADD `id` INT NOT NULL
AUTO_INCREMENT PRIMARY KEY;
```

This command adds a new primary key called `id` and specifies that when a new road gets added to the table, this field should provide a new, unique incremented value.

Enabling the user to choose items

The next step is to enable users to decide which items should be part of the PDF. To do that, you will add a second form to the saved entries page. Open the viewSaveEntries.php file and make the changes shown in Listing 22.

Listing 22. Adding PDF generation options to viewSavedEntries.php

```
...
         echo "<input type='hidden' name='type' value='$type'/>";
         echo "<td><a href='$link'>$title</a></td>";
         echo "<td><input type='submit'
 value='delete'/></td>";
         echo "<td><a href=
'/feed/fullText?feedTitle=$feedTitleamp;channelTitle=$channelTitle'
>$entrysaved</a></td></form></tr>\n";
      }
     ?>
  </table><br>
  <h3>Generate a PDF:</h3>
  <form method='POST' action='/feed/createPdf'>
  <table>
    <tr>
      <td>Feed > Title (Click to View)
        amp;nbsp;amp;nbsp;amp;nbsp;amp;nbsp;amp;nbsp;amp;nbsp;</td>
```

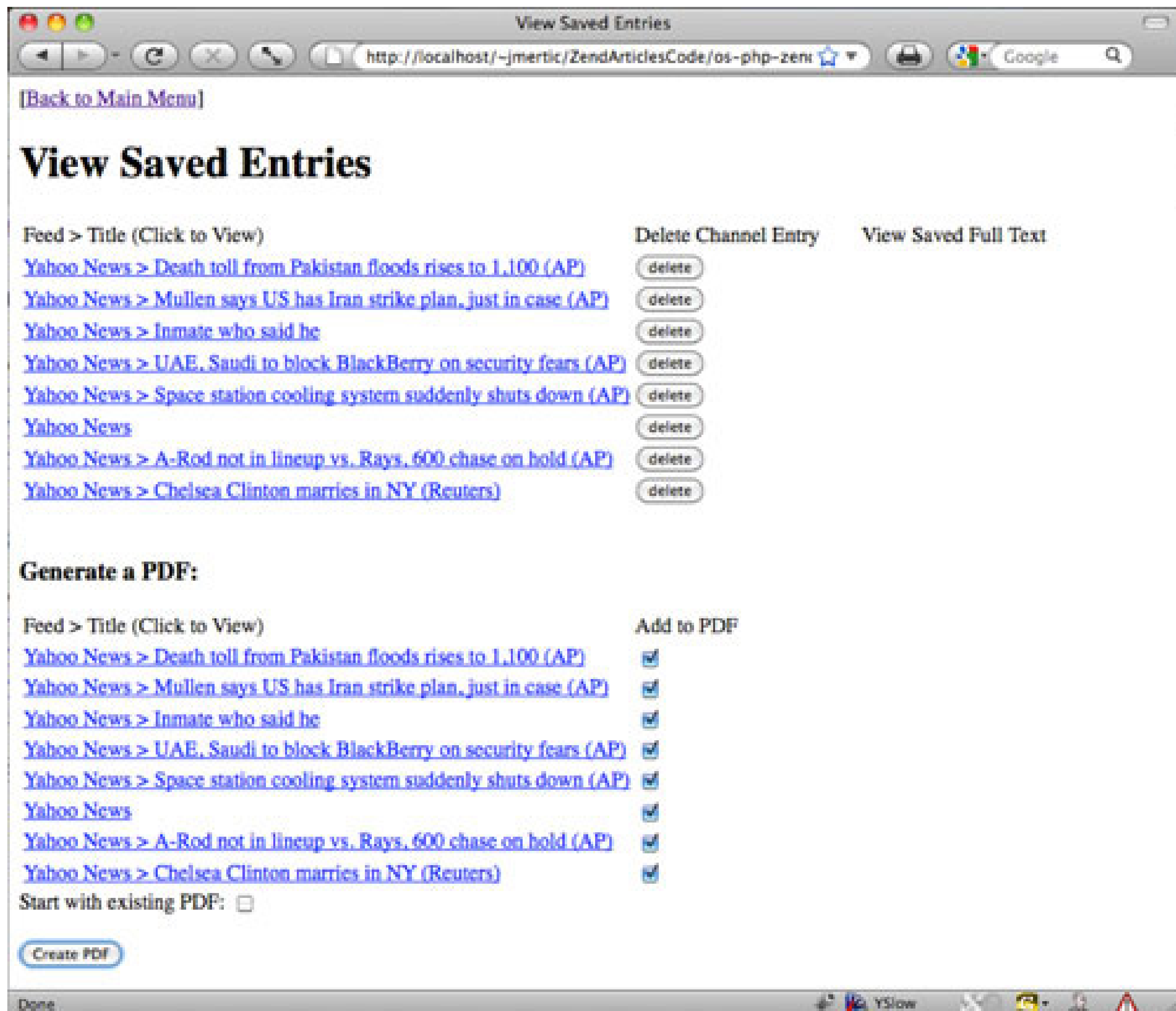

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
      <td>Add to PDF</td>
    </tr>
<?php
     foreach ($this->entries as $row) {
         $link = $row['link'];
         $channelTitle = $row['channelname'];
         $feedTitle = $row['feedname'];
         $title = "$feedTitle";
         $entrysaved = '';
         if($row['entrysaved'] == 'true')
         {
             $entrysaved = 'Full Text';
         }
         if ($row['channelname'] != '')
         {
             $title = "$title > $channelTitle";
             $type = 'rssFeed';
         } else {
             $type = 'webPage';
         }
         $id = $row['id'];
         echo "<tr><td><a href='$link'>$title</a></td>";
         echo "<td><input type='checkbox' name='$id' " .
              "checked='checked'></td></tr>\n";
     }
?>
  </table>
  <input type='Submit' value='Create PDF'>
  </form>
</body>
</html>
```

This is a generic HTML form, with checkboxes for each `id` value (see Figure 10).

Figure 10. Enabling the user to choose items

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Filtering the PDF based on user choices

Now to make this work, you have to look for the checkboxes in the `createPdfAction()` function (see Listing 23).

Listing 23. Filtering the PDF

```
...
    public function createPdfAction()
    {
...
        $sql = $select->__toString();
        $entries = $db->fetchAll($sql);

        $startPos = $topPos - 120;

        $filterPost = Zend_Registry::get('fPost');
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

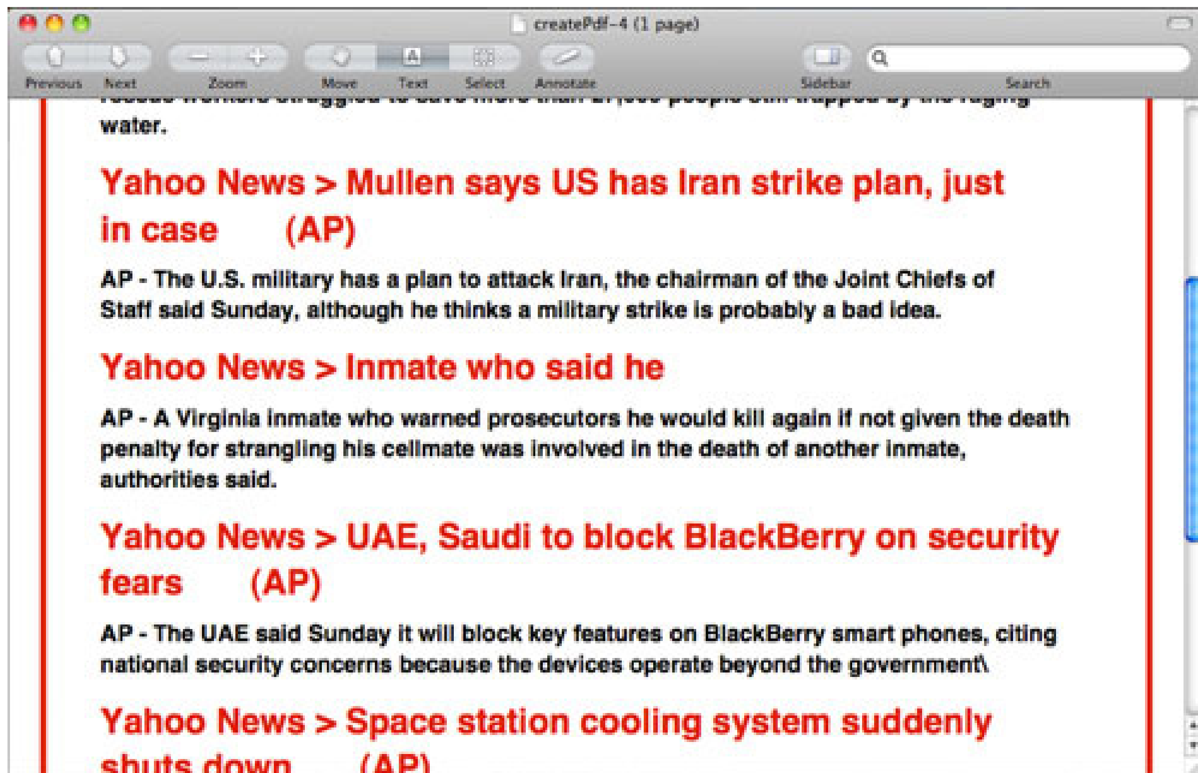
```
        foreach($entries as $row){

            if($filterPost->getRaw($row['id']) == "on"){

                $title = $row['feedname'];
                $entrydata = $row['entrydata'];
...
                    $page->drawText($line, 48, $startPos);
                    $startPos = $startPos - 16;

                }
                $startPos = $startPos - 16;
            }
        }

        array_push($pdf->pages, $page);

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

Here again, the process is straightforward. You first retrieve the filter you've used in previous parts of this series from the registry, then use it to retrieve the value for the checkbox that corresponds to the ID value of the current entry a being evaluated.

If the box was checked when the form was submitted, the value will be `on`, and the routine includes the entry in the PDF. If not, it skips the entry. The result is a PDF that includes only the entries that have been specifically requested (see Figure 11).

Figure 11. Filtering the PDF based on the user

© Copyright IBM Corporation 2006, 2011. All rights reserved.



water.

Yahoo News > Mullen says US has Iran strike plan, just in case (AP)

AP - The U.S. military has a plan to attack Iran, the chairman of the Joint Chiefs of Staff said Sunday, although he thinks a military strike is probably a bad idea.

Yahoo News > Inmate who said he

AP - A Virginia inmate who warned prosecutors he would kill again if not given the death penalty for strangling his cellmate was involved in the death of another inmate, authorities said.

Yahoo News > UAE, Saudi to block BlackBerry on security fears (AP)

AP - The UAE said Sunday it will block key features on BlackBerry smart phones, citing national security concerns because the devices operate beyond the government\

Yahoo News > Space station cooling system suddenly shuts down (AP)

Editing an existing PDF: Adding the option

Finally, you want to give the user the option to use an existing PDF, rather than creating a new one. This can be handy as a way of enabling users to create a backup of saved entries. For example, if a user collects several dozen, or even several hundred, saved entries, he can add them to a PDF file and delete them from the system. When he has more entries to add, he can simply opt to use the previous PDF file. The function adds new items at the end of the document.

First, provide the option to use the existing PDF by adding a checkbox to the viewSaveEntries.php file (see Listing 24).

Listing 24. Choosing to use an existing PDF

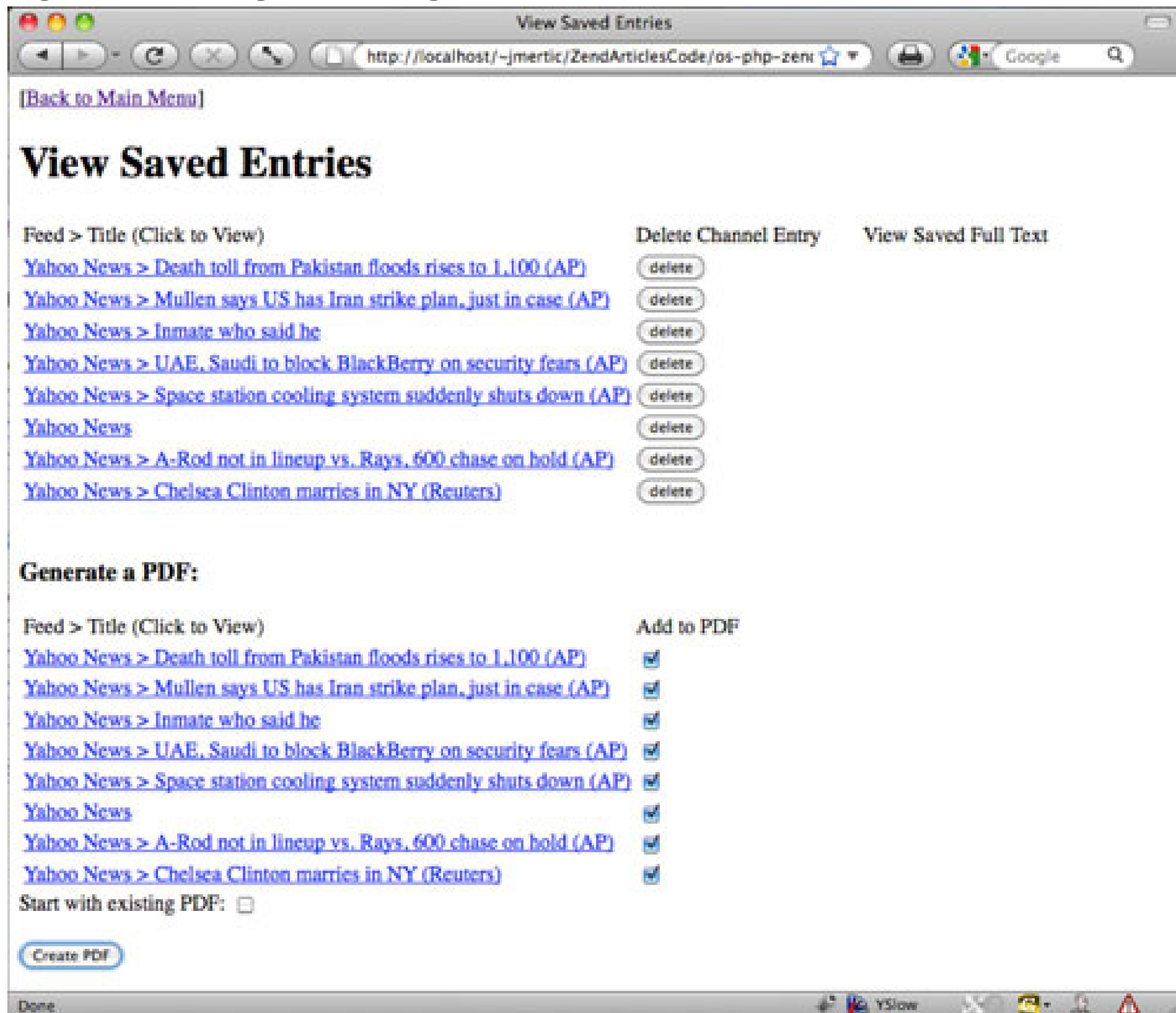
```
...
        if($row['channelname'] != ''){
            $title = "$title > $channelTitle";
            $type = 'rssFeed';
        } else {
            $type = 'webPage';
        }
        $id = $row['id'];
        echo "<tr><td><a href='$link'>$title</a></td>";
        echo "<td><input type='checkbox' name='$id'\
        checked='checked'></td></tr>\n";
    }
?>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
  </table>
  Start with existing PDF: <input type='checkbox'

 name='startExisting' /><br /><br />
  <input type='Submit' value='Create PDF'>
  </form>
</body>
</html>
```

This adds the checkbox to the end of the form (see Figure 12).

Figure 12. Editing an existing PDF



Editing an existing PDF: Adding pages

To use an existing PDF, you will need to make changes to the `createPdfAction()` function (see Listing 25).

Listing 25. Adding pages to an existing PDF document

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
...
    public function createPdfAction()
    {

        require_once 'Zend/Pdf.php';


        $input = new Zend_Filter_Input(
            array('username'=>'StringTrim'),
            array('username'=>'Alpha'),
            $_SESSION);
        $username = $input->getUnescaped('username');

        $input = new Zend_Filter_Input(
            array('*'=>'StringTrim'),
            null,
            $_GET);
        $startExisting = $input->getUnescaped('startExisting');

        $pdf = '';
        if ($startExisting == 'on')
        {
            if (file_exists($username.'.pdf'))
            {
                $pdf = Zend_Pdf::load($username.'.pdf');
            }
            else
            {
                $pdf = new Zend_Pdf();
            }
        }
        else
        {
            $pdf = new Zend_Pdf();
        }

        $page = $this->newPdfPage();
        $chompImage = Zend_Pdf_Image::imageWithPath(
                         'E:\sw\public_html\chomp.jpg');

...
                }
                $startPos = $startPos - 16;
            }
        }

        array_push($pdf->pages, $page);

        $pdf->save($username . '.pdf');

        header('Content-type: application/pdf');
        echo $pdf->render();

    }
...
```

First, note that the username check has been moved to the top of the function because if there is an existing PDF file, it will be named username.pdf. Next, if the user did choose to use an existing PDF, you check to see whether one already exists. If so, you use the `load()` function to load the document as the basis for the `$pdf` object. If not, or if the user did not choose to use existing document, you simply create a new one.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

From here, processing proceeds exactly as before, except that before you render the PDF to the browser, you save it to the server for next time.

Section 6. Summary

The Chomp online feed reader is now beginning to take shape. Users can register for accounts and subscribe to feeds, as well as read them. Now, users can export saved entries as PDF documents. To make this happen, you used the Zend Framework's `Zend_PDF` component to create a new PDF, add shapes, graphics, and text, and save it. You also looked at outputting the PDF directly to the browser and at using an existing PDF, rather than creating a new one.

In addition, you looked at ways to get around some of the problems with manually building PDFs, such as an inability to line-wrap automatically and an inability to add new pages to the document automatically.

In Part 6, learn how to send a user an e-mail when a favorite feed is updated.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Downloads

Description	Name	Size	Download method
Part 5 source code	os-php-zend5.source.zip	21KB	HTTP

Information about download methods

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Resources

Learn

- The Zend Framework website maintains the latest documentation.
- To better understand the goals of the feed-reader project, read "Introduction to Syndication, (RSS) Really Simple Syndication," by Vincent Luria.
- Thought Storms ModelViewController explains the MVC and the controversy and confusion surrounding it.
- In "Server clinic: PDF for the server," Cameron Laird explains the benefits that can be achieved from using PDF creation.
- Learn more about Zend Core for IBM, a seamless, easy-to-install, and supported PHP development and production environment that features integration with DB2 and Informix databases.
- "Generate PDF files from Java applications dynamically" provides a step-by-step guide to using iText to generate PDF documents from Java applications.
- "Publish your DB2 data to PDF using XSL" details how to do so in three steps.
- The developerWorks tutorial series "Learning PHP" takes you from the most basic PHP script to working with databases and streaming from the file system.
- Receive an RSS notification about the latest installment in the "Understanding the Zend Framework" series.
- The PHP Function Reference is another valuable resource.
- "Zend Framework Preview Release Available," in Enterprise PHP Magazine, provides an overview of the components that make up the Zend Framework.
- Implement SOAP services with the Zend Framework (Vikram Vaswani, developerWorks, May 2010): Check out this article on using SOAP web services with the Zend Framework.
- PHP.net is the central resource for PHP developers.
- Check out the "Recommended PHP reading list."
- Browse all the PHP content on developerWorks.
- Expand your PHP skills by checking out IBM developerWorks' PHP project resources.
- To listen to interesting interviews and discussions for software developers, check out developerWorks podcasts.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

- Using a database with PHP? Check out the Zend Core for IBM, a seamless, out-of-the-box, easy-to-install PHP development and production environment that supports IBM DB2 V9.
- Stay current with developerWorks' Technical events and webcasts.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Watch and learn about IBM and open source technologies and product functions with the no-cost developerWorks On demand demos.

Get products and technologies

- Download the Zend Framework from Zend Technologies.
- Download PHP V5.x from PHP.net.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.
- Participate in the developerWorks PHP Forum: Developing PHP applications with IBM Information Management products (DB2, IDS).

About the author

Nicholas Chase

Nicholas Chase, a Studio B author, has been involved in Web site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. Nick has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, and an Oracle instructor. More recently, he was the Chief Technology Officer of an interactive communications firm in Clearwater, Florida, USA, and is the author of several books on Web development, including *XML Primer Plus* (Sams). He's currently trying to buy a farm

© Copyright IBM Corporation 2006, 2011. All rights reserved.

so he and his wife can raise alpacas and chickens. He loves to hear from readers and can be reached at: nicholas@nicholaschase.com.

© Copyright IBM Corporation 2006, 2011. All rights reserved.