

---

# Understanding the Zend Framework, Part 8: Adding related information and services

Skill Level: Intermediate

[Sean Kelly \(skelly@idsociety.com\)](mailto:skelly@idsociety.com)  
Web Application Developer  
ID Society

29 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. This tutorial, the eighth part of the series, adds an extra dimension to your feed reader by linking the online resources of Amazon.com, Yahoo!, Twitter and Flickr with your current application to create a robust mashup.

## Section 1. Before you start

This tutorial is for developers who want to learn about creating a simple *mashup*, a web application that uses information from different sources. It uses the Zend Framework's `Zend_Services` component and the principle of Representational State Transfer (REST) to demonstrate how to seamlessly include information to extra resources related to your feeds.

Using the examples of Amazon Web Services (AWS), Flickr Services, and Yahoo! Search Web services, and Twitter, this tutorial teaches you how to apply the PHP Zend Framework to quickly search those sites for relevant information. You will integrate this functionality into the Chomp feed reader developed in the previous parts of this series.

## About this series

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

[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 deal 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 learned about 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. [Part 5](#) explained how to use the `Zend_PDF` module to enable the user to create a customized PDF of saved articles, images, and search results. In [Part 6](#), you used the `Zend_Mail` module to alert users to new posts. In [Part 7](#), you looked at searching saved content and returning ranked results. Here in [Part 8](#), you will add an extra dimension to your feed reader by linking the online resources of Amazon.com, Yahoo!, Twitter and Flickr with your current application to create a robust mashup. And in [Part 9](#), you will add Ajax to streamline the application and improve usability.

## About this tutorial

The process of providing links to related resources, or *hyperlinking*, is at the very core of the web. In fact, it is the very reason the web exists, and it's nothing new. However, what would previously need to be done by a human — searching the web for relevant resources, bookmarking URLs, and manually entering them into an article — can be easily automated when sites like Amazon and Yahoo! open up their sites using an application programming interface (API). APIs allow developers to access small snippets of information in a form other than a web page, most commonly with some type of XML. By automatically requesting a link to a book, you can dramatically increase the richness of your application without adding much overhead. Further, the Zend Framework's built-in APIs allows you to add relevant code in a few simple lines.

Like any good feed reader, Chomp provides the user with a collection of articles that are interesting. You can probably assume that if a feed mentions, say, a book related to a given article, the user would find that book interesting, as well. This

tutorial seeks to give the user quick and easy access to this extra information by connecting directly to the Amazon.com developer's API and providing a link to the book on Amazon's site. Similarly, you can provide a link to a photo of an interesting location through Flickr, you can provide a link to Yahoo! news search results, or you could even search Twitter for the latest tweets.

This tutorial will cover:

- GET, POST, REST -- Stateless vs. sessioned network interactions and how stateless interfaces can dramatically improve usability
- A quick overview of the Amazon Web Services interface
- Using the Zend Framework's `Zend_Service_Amazon` component to access Amazon
- Yahoo! Developer Network -- using the searching websites, news, and local results
- An overview of Flickr services, tagging, and using `Zend_Service_Flickr` to search photos
- How to add Twitter results alongside the search results given when a feed search is performed
- Integrating the Yahoo! Developer Network with Chomp using the `Zend_Service_Yahoo` component
- Adding all of this information to the Chomp interface

## Prerequisites

This tutorial assumes that you have a good familiarity with PHP programming. It would be helpful if you have a general familiarity with Amazon.com, Flickr, Twitter and Yahoo!, though you do not need to know much about these sites.

## System requirements

To follow along, you will need to have several pieces of software installed. This tutorial will cover installation and configuration, but make sure to download the following pieces of software. For detailed installation instructions on installing XAMPP and the Zend Framework, see [Part 2](#).

### XAMPP

This is an easy-to-install version of Apache, MySQL, and PHP rolled into one package.

### AWS Access Key

You need an Amazon Web Services access key.

### Yahoo! Application ID

You need a Yahoo! Application ID, but first you need a Yahoo! username.

---

## Section 2. Stateless vs. sessioned interactions

In this section, you will get a brief overview of the principle of Representational State Transfer (REST) and how it applies to web services. Understanding the background and how it relates to the architecture of the Internet will help you understand Web services in general before you go ahead and add it to the Chomp application.

### REST and the stateless Web

The web is a collection of hyperlinked documents and in the end, by its very definition, nothing more. Intrinsic in the way you use the web is the assumption that every resource can be accessed using an identifier called a *Uniform Resource Locator* (URL). A URL tells your browser everything it needs to know to get a document. It is universal across competing browsers like Mozilla Firefox and Microsoft Internet Explorer®, computers and mobile devices, and networks of all kinds.

The architecture of the web, therefore, has two basic and atomic operations: `GET` and `POST`. A client can either retrieve information from a server or post new information (and retrieve any result). The core protocol of the web doesn't have any provisions for more complex communication between the client and server. This is the principle of REST: The client and server need to know nothing more than the fact that one is requesting a document, and the other is providing it. After that, both have essentially forgotten that the transaction took place. In this sense, the web is based on a stateless architecture.

### Not RESTing: Sessioned interactions

In practice, password protection, session tracking, and more complex interactions are required in the way you use the web. Systems have been built on top of the RESTful architecture of the web that nonetheless enable the client and server to keep track of each other, providing for sessioned interactions. For example, when you log into your bank's website, you're not simply requesting a page but instead

POSTing your password onto the bank's web server. Every subsequent GET request is part of a very large session between your browser and the bank. If either one loses the session information, subsequent requests will fail to provide the appropriate documents.

## Choosing sides

While sessioned interactions are useful for a single closed application, or even a large distributed one, they are not necessarily good for the web as a whole. Historically, the web has favored services that follow the principles of REST over those that do not. By keeping services open and easily accessible, developers have a number of incentives to use them to create a content-rich web applications. Companies like Google and Amazon that have opened up their stores of information to simple stateless queries have seen an explosion in the use of their services in a number of innovative applications.

## The right tool for the job: Benefits of stateless interactions

Today, there is no Swiss Army knife of web services. As a good programmer, you need to evaluate how you intend to use the service, what information you are sending, and whether the client and server need to communicate. There are a number of benefits to using a system that complies with the principles of REST:

- Resources can easily be bookmarked, returned to, sent via e-mail to friends, etc.
- Resources can be used by a much wider variety of clients and servers
- Easier caching
- Ease of programming

As far as the Chomp application goes, you're reading information provided by the databases of Flickr, Amazon, Twitter and Yahoo!, so there is little need to implement a system that keeps track of state. You are not submitting information to any of these services, nor will you need to send authentication or obtain permissions. You are simply obtaining data and relaying it to the user, so it's appropriate to use their RESTful solution.

---

## Section 3. Adding related books: Amazon

Amazon.com allows free access to its monolithic database of books, music, movies, and other retail goods through AWS. By signing up for a free account, developers can make requests to Amazon.com, displaying relevant content directly on their websites. As such, you will alter the Chomp application to display information about Amazon products related to posts returned by a search.

This section helps you get started in setting up an Amazon.com account and in utilizing the Zend Framework to make simple lookups for books at Amazon.com (though the lookups are by no means limited to one type of media). You will learn how to craft a request using the `Zend_Service_Amazon` object and to display the response with an appropriate view object. Finally, you will integrate your code into Chomp to provide relevant books for the channel and search pages.

## Obtaining your Amazon Web Services access key

One thing that remains consistent among web services APIs is the need for a key, much like a password. Since requests to the API are made from an application instead of a human using a web client, service providers use developer keys to prevent abuse and denial-of service (DoS) attacks. Developer keys are easy to obtain and require only an e-mail address.

To obtain your AWS access key:

1. Visit [Amazon Web Services](#).
2. If you already have one, sign in with your normal Amazon.com account information. Amazon.com will not charge you for using this service.
3. If you don't already have an Amazon.com account, simply enter your e-mail address, name, and a password. Your account information will be sent to you.

## Adding relevant search results

You will now use your AWS access key to enhance the feed search page you created in [Part 7](#). Recall that the search page is part of the feed controller and the `viewSearchResults` action. Open the file `controllers/FeedController.php` and make the following addition, as shown in Listing 1.

### Listing 1. Adding a simple Amazon.com lookup

```
public function viewSearchResultsAction()  
{
```

```
// ...
$view->hits = $hits;

require_once 'Zend/Service/Amazon/Query.php';

$key = 'YOUR_AMAZON_KEY';
$country = 'US';
$secret = 'YOUR_AMAZON_SECRET_KEY_HERE';
$amazonQuery = new Zend_Service_Amazon_Query($key,$country,$secret);
$amazonQuery->Category('Books');
$amazonQuery->Keywords($filterGet->getRaw('query'));

$view->amazonHits = $amazonQuery->search();

echo $view->render('viewSearchResults.php');
```

Let's examine what this does. First, the query object is created and bound to your key. Next, you set the query to only return books and set the keywords to be whatever the user entered into the search bar. Finally, you run the search and assign the results to the page view.

Requests made to `Zend_Service_Amazon_Query` are made using the Fluent Interface pattern, which means that the name of the parameter is also the name of the method. That is, `$amazonQuery->Param1('val1')->Param2('val2')` is interpreted as setting `Param1` to `val1` and `Param2` to `val2`. Notice that with the Fluent Interface pattern, the calls can be changed to a single query.

**Note:** Using this format, the configuration for `amazonQuery` can be put into the easy-to-read code shown in Listing 2.

### Listing 2. amazonQuery in easy-to-read format

```
$amazonQuery = new Zend_Service_Amazon_Query($key,$country,$secret);
$amazonQuery->Category('Books')
->Keywords($filterGet->getRaw('query'));
```

The function `Zend_Service_Amazon_Query::search()` returns a `Zend_Service_Amazon_ResultSet` object, which you can use like an array in a `foreach` loop. It contains a list of `Zend_Service_Amazon_Item` objects, which have the following properties set, among others:

- `ASIN` — A string representing a unique ID for that item in the Amazon.com database
- `DetailPageURL` — A link to the item detail page at Amazon.com
- `Title` — The title of the Item

You can use these bits of information in your view object to display the results as a list of items, providing links to the relevant resources. To do this, open `views/viewSearchResults.php` and add the code in Listing 3 to the bottom of the

page.

### Listing 3. Writing code for displaying Amazon.com results

```
...
?>
</table>

<h4>Were you looking for any of these books?</h4>
<ol>
<?php
    foreach ($this->amazonHits as $result) {
        echo "<li>";
        echo "<a href='" . $result->DetailPageURL . "' title='" .
            htmlentities($result->Title, ENT_QUOTES). " at Amazon.com'>";
        echo "<strong>" . htmlentities($result->Title) .
            "</strong></a></li>";
    }
?>
</ol>
```

Notice that even though `amazonHits` is not an array, it can still be used in a `foreach` loop. That's it. Now enter a query into the search page to display related book results.

## Changing the query to include images

To save on bandwidth, the default results from Amazon include only the most basic information: title, Amazon Standard Identification Number (ASIN), resource link. The AWS documentation states that in order to get more detailed results, you must first set the `ResponseGroup` to one of the following values:

- `Request` tells AWS to return a copy of the request.
- `ItemIds` tells AWS to return the ASIN (default).
- `OffersSummary` tells AWS to return prices for the item other than the listing price.
- `SalesRank` tells AWS to return the item's popularity in sales.
- `Images` tells AWS to return information about images of the product.
- `Small` tells AWS to return basic information about the item, such as title, creator, product group, URL, and manufacturer (default).
- `Medium` tells AWS to return everything in `Small`, as well as `OffersSummary`, `SalesRank`, `Images`, and some additional information.
- `Large` tells AWS to return every piece of information associated with the

item.

To get images to show up, you simply need to set the `ResponseGroup` parameter to `Medium`, which gives us the chain shown in [Listing 4](#).

#### Listing 4. Querying AWS for images

```
public function viewSearchResultsAction()
{
    // ...
    require_once 'Zend/Service/Amazon/Query.php';

    $key = 'YOUR_AMAZON_KEY';
    $country = 'US';
    $secret = 'YOUR_AMAZON_SECRET_KEY_HERE';
    $amazonQuery = new Zend_Service_Amazon_Query($key,$country,$secret);
    $amazonQuery->Category('Books')
        ->Keywords($filterGet->getRaw('query'))
        ->ResponseGroup('Medium');

    $view->amazonHits = $amazonQuery->search();
    // ...
}
```

This block tells AWS to return the extra information associated with the `Medium` response group, but the functionality does not change until you write the code to display the extra results. In [Listing 5](#), you write the code that tells the search page to display the image with the title and to display the sales rank of the item.

#### Listing 5. Displaying the extra results from AWS

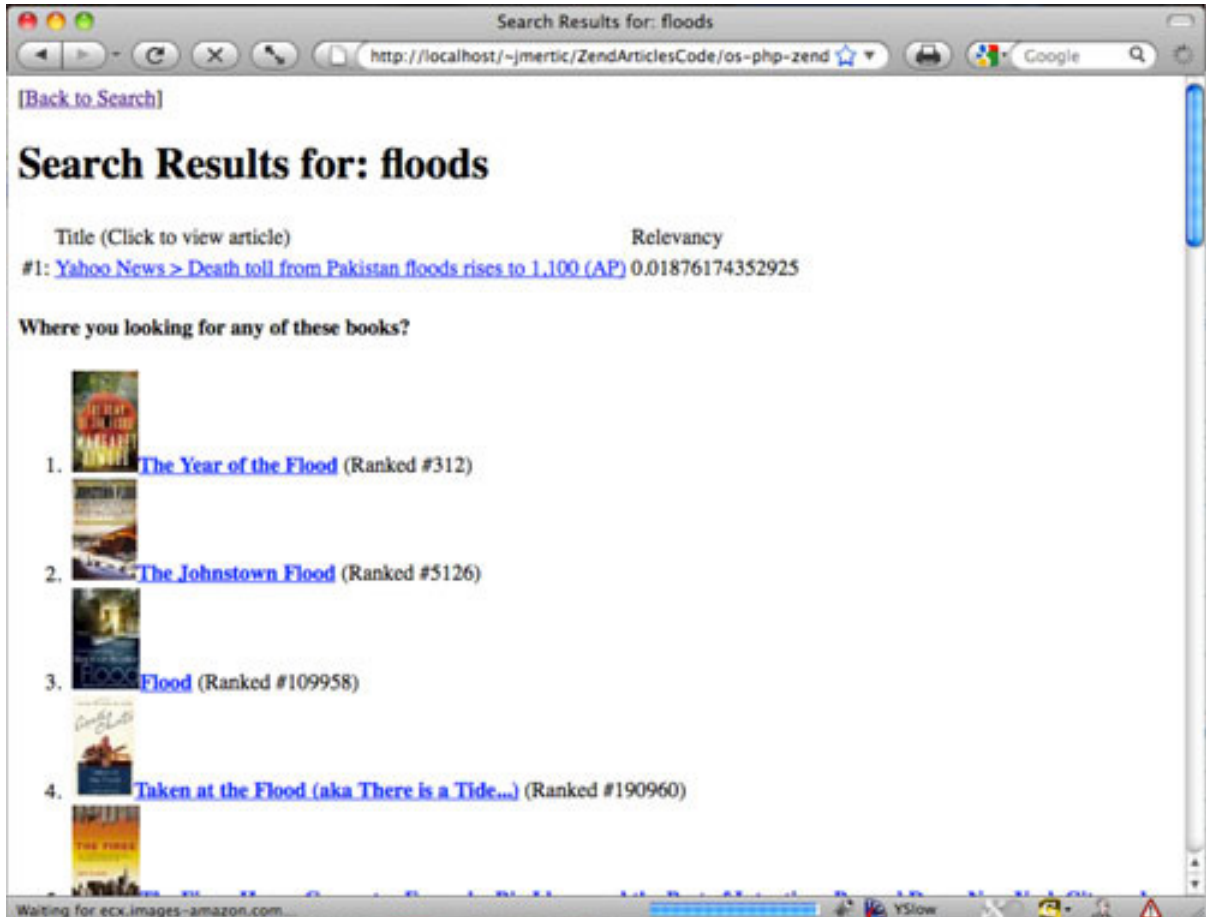
```
<h4>Were you looking for any of these books?</h4>
<ol>
<?php
    foreach ($this->amazonHits as $result) {
        echo "<li><img src='" .
            $result->SmallImage->Url->getUri() .
                "' width='" . $result->SmallImage->Width .
                "' height='" . $result->SmallImage->Height . "' /> ";
        echo "<a href='" . $result->DetailPageURL . "' title='" .
            htmlentities($result->Title, ENT_QUOTES) . " at Amazon.com'>";
        echo "<strong>" . htmlentities($result->Title) .
            "</strong></a> ";
        echo " (Ranked #" . $result->SalesRank . ")</li>";
    }
?>
</ol>
```

The `SmallImage` property of the result is actually a `Zend_Service_Amazon_Image` object, which has the following properties:

- `Url` of type `Zend::Uri` is the source of the image.
- `Width` of type `int` is the width of the image.
- `Height` of type `int` is the height of the image.

Along with `SmallImage`, you can access `MediumImage` and `LargeImage` for different-size previews of the project. All three sizes are returned with the `MediumResponseGroup`. Visiting the search results page produces the list of books shown in Figure 1. The Zend Framework Manual provides a full list of item properties accessible from `Zend_Service_Amazon_Item`, and an explanation of those properties is available in the AWS documentation (see [Resources](#)).

**Figure 1. In-depth result from Amazon.com**



As part of its developer network, Amazon offers a series of online tools for testing and debugging API calls. The Amazon Web Services Zone has a number of applications called Scratchpads that allow you to enter an API request into a form and sample the resulting XML response. For example, to test an `ItemSearch` request, you simply enter your `SubscriptionId` into the form, selected Books from the `SearchIndex` drop-down, enter a search term, and click **View Results**. The response will be XML document that exactly resembles what your PHP application gets. [Listing 6](#) shows a portion of the response you will get when you enter `PHP` into the search form. Notice that one useful piece of information you can glean from the request is the `RequestProcessingTime`.

### Listing 6. Response from AWS

```
<?xml version="1.0" encoding="UTF-8"?>
<ItemSearchResponse xmlns="http://webservices.amazon.com/...
<OperationRequest>
...
<RequestProcessingTime>0.07666015625</RequestProcessingTime>
</OperationRequest>
<Items>
<Request>
  <IsValid>True</IsValid>
  <ItemSearchRequest>
    <Keywords>php</Keywords>
    <SearchIndex>Books</SearchIndex>
  </ItemSearchRequest>
</Request>
<TotalResults>416</TotalResults>
<TotalPages>42</TotalPages>
<Item>
  <ASIN>0672326728</ASIN>
  <DetailPageURL>http://www.amazon.com/exec/obidos...
</DetailPageURL>
  <ItemAttributes>
    <Author>Luke Welling</Author>
    <Author>Laura Thomson</Author>
    <Manufacturer>Sams</Manufacturer>
    <ProductGroup>Book</ProductGroup>
    <Title>PHP and MySQL Web Development
      (3rd Edition) (Developer's Library)</Title>
  </ItemAttributes>
</Item>
...
```

Next, let's look at adding other information to the Chomp interface.

---

## Section 4. Adding related photos: Flickr

Flickr photos are commonly used to add relevant images to a site without much effort, with tags such as *colorful* and *interesting* making for popular photo streams. If you haven't yet, you should take a moment to peruse Flickr to get a sense of the depth of its collection of photos.

In this section, you will add a piece of functionality similar to what you added in the previous section. When Chomp returns a list of feed entries for a search, it will also display photographs relevant to the given search term. You will also add functionality that searches a given feed for the author's e-mail address, looks to see if Flickr has an account for that address, and provides a link to the account.

### Obtaining a developer key for Flickr

Like Amazon.com's API, Flickr requires the use of a developer key to make REST

requests. Signing up is easy:

1. Since being bought, Flickr accounts are now linked to Yahoo! accounts. Visit the Flickr sign-up page to sign up for a free account through Yahoo! (see [Resources](#)).
2. [Apply for an API key](#). Your key will be linked with your account (see [Resources](#)).

## Providing search results for Flickr

After you have obtained your API key, connecting to Flickr is as simple as making a single function call. The class that does all of the work is `Zend_Service_Flickr`. Rather than providing the complex query system `Zend_Service_Amazon` has, searching for photos is done through one basic request, shown in [Listing 7](#).

### Listing 7. Function description for searching for photos

```
/**
 * Find Flickr photos by tag.
 *
 * @param mixed $query A single tag or an array of tags.
 * @param array $options Parameters to refine your query.
 * @return Zend_Service_Flickr_ResultSet
 */
public function tagSearch($query, $options = null) {
    // ...
}
```

Notice that because the function expects `$query` to be a single tag or an array of tags, you have to take the search terms the user enters and split them. For this example, you'll simply split them along the space character. Thus, you get the simple code shown in [Listing 8](#) for querying Flickr.

### Listing 8. Requesting photos related to a search term

```
public function viewSearchResultsAction()
{
    // ...
    $view->amazonHits = $amazonQuery->search();

    // Perform a flickr.com lookup
    $key = 'YOUR_FLICKR_API_KEY';
    $flickrQuery = new Zend_Service_Flickr($key);
    $tags = explode(' ', $filterGet->getRaw('query'));
    $view->flickrHits = $flickrQuery->tagSearch($tags);

    echo $view->render('viewSearchResults.php');
}
```

As in the previous section, you add the control functionality directly to `FeedControl`'s `viewSearchResults` action. The results from the query are assigned to the view's `flickrHits` property. Again, no functionality has been added until you display the results, so you must next make another addition to `viewSearchResults` (see [Listing 9](#)).

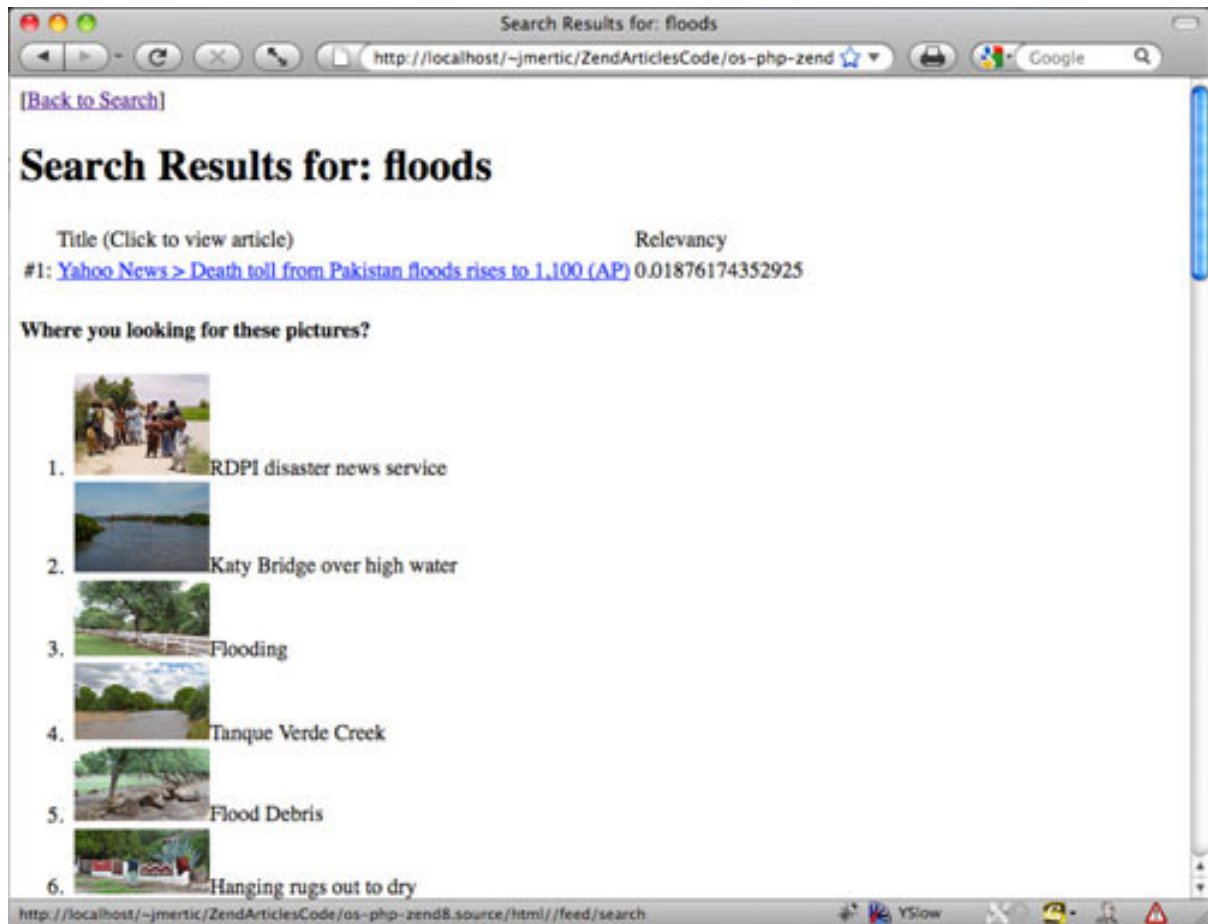
### Listing 9. Outputting Flickr photos

```
...
    echo " (Ranked #" . $result->SalesRank . ")</li>";
}
?>
</ol>

<h4>Related results from <a
href="http://flickr.com">Flickr</a></h4>
<ol>
  <?php
    foreach ($this->flickrHits as $result) {
      echo "<li><img src='" . $result->Thumbnail->uri .
'" width='" .
        $result->Thumbnail->width . "' height='" .
        $result->Thumbnail->height . "' />";
      echo $result->title;
      echo "</li>";
    }
  ?>
</ol>
```

This code outputs a basic ordered list of photos that Flickr finds as related to the search term given, which is shown in [Figure 2](#).

### Figure 2. Flickr photos added to search results



## Adding metadata from Flickr

Results from Flickr are not limited to a simple thumbnail. The `Zend_Service_Flickr_Result` objects returned contain a number of properties to provide useful information to the user about the photos, including:

- `id` (int) -- The image ID assigned by Flickr
- `owner` (int) -- The photo owner's network services ID (NSID)
- `secret` (string) -- A key used in URL construction
- `server` (string) -- The server name to use for URL construction
- `title` (string) -- The photo's title
- `license` (string) -- The license the photo is available under
- `dateupload` (string) -- The date the photo was uploaded
- `datetaken` (string) -- The date the photo was taken

- `ownername` (string) -- The screen name of the owner

Combining information from these properties, you can create a more descriptive table of photos, as well as links to more information about the photos. The code for this is shown in [Listing 10](#).

### Listing 10. Adding a full collection of metadata into the Flickr results

```

...
        echo " (Ranked # " . $result->SalesRank . ")</li>";
    }
? >
</ol>

<table>
<caption>Photos from <a
href="http://flickr.com/">Flickr</a></caption>

<tbody>
<?php

foreach ($this->flickrHits as $index=>$result) {
    // Begin column
    if ( $index % $this->columns == 0 ) {
        echo '<tr>';
    }

    $thumbnail = '';
    echo '<td><a href="' . $result->Large->clickUri .
        '" title="to Flickr">' . $thumbnail . '</a><br />';
    echo '<small>by <a href="http://www.flickr.com/photos/' .
    $this->escape($result->ownername) .
        '" title="Owner">' . htmlentities($result->ownername) .
        '</a> on ' . $result->datetaken .
    '</small></td>';

    // Close column
    if ( $index % $this->columns == $this->columns - 1 ) {
        echo '</tr>';
    }
}
?>
</tbody>
</table>

</body>
</html>
</code>

```

---

## Section 5. Adding related tweets from Twitter

Twitter is the one of the most popular social networking and microblogging services on the internet. Twitter provides an easy to use API for interacting with their web

service, which includes the ability for users to post tweets through the API, as well as search for tweets made by other users. No API key is required to use the service, and you do not need an account to perform search calls through the API.

## Adding recent tweets

For the Chomp application, you will add Twitter results alongside the search results given when a feed search is performed, similar to what you have done in the previous examples that used Amazon and Flickr. Again, you will add the functionality inside the `viewsearchresults` action.

The Zend Framework provides a very easy to use Twitter API that allows the use of native PHP method calls to access the web service, under the `Zend_Service_Twitter` component. [Listing 11](#) shows the needed code to handle the Twitter API integration.

### Listing 11. Requesting tweets related to a search term

```
public
function
viewSearchResultsAction()
{

    // ...
    $view->flickrHits
    =
    $flickrQuery->tagSearch($query);

    //
    Perform a twitter
    search
    $twitterQuery
    = new
    Zend_Service_Twitter_Search('json');
    $view->>twitterHits
    =
    $twitterQuery->search($query,
    array('lang' =>
    'en'));
    echo
    $view->render('viewSearchResults.php');
}
```

You can return results from the API in either Atom or JSON format; for this example you will choose the JSON format since it can be directly translated into a PHP object easily. Then a simple `search()` method call will return the recent tweet results.

You will also modify the view code inside `viewSearchResults.php` to add displaying the tweets to the display (see [Listing 12](#)).

### Listing 12. Adding a full collection of metadata into the Twitter results

```
...
echo "</li>";
}
?>
</ol>

<h4>Where you
looking for these
tweets?</h4>
<ol>
<?php
foreach
($this->twitterHits['results']
as $result) {
    echo
    "<li><img src=' "
    .
    $result['profile_image_url']
    .
    "'
    />&nbsp;";
    echo
    "<strong>" .
    $result['from_user']
    .
    "</strong>&nbsp;";
    echo
    $result['text'];
    echo
    "</li>";
}
?>
</ol>

</body>
</html>
```

This code will output a list of the recent tweets that match the search criteria, as you can see in [Figure 3](#).

### Figure 3. Recent tweets that match the search criteria



1. Log into your Yahoo! account
2. Visit the web services application
3. Unlike Amazon.com and Flickr, Yahoo! allows you to select your own application key, so you can select a meaningful key, such as 'mydomain.com\_Yahoo\_Key'. You may also provide an alternate e-mail address.

## Adding relevant news results and utilizing the MVC paradigm

Yahoo's enormous database of web resources will be best at providing useful results from a stream of text. Thus, you can actually take a piece of text gleaned from the RSS feed, such as the title, and pass it off to Yahoo! as your request. Since you are making a news feed reader, it only makes sense to provide news results that are relevant to the current tutorial.

Since you're going to be changing the actual display of the feed, you need to edit the function `viewChannelAction` in the `FeedController` object. As with the Amazon.com and Flickr interfaces, querying Yahoo! is as simple as initializing the `Zend_Service_Yahoo` object, passing it the query, and writing a view to display the results.

You will be creating a new view dedicated to displaying Yahoo! results. Your controller will be responsible for looping through each item in the feed and using `Zend_View` to generate the HTML representation of the response from Yahoo! Next, you will create a new array, called `yahooNews`, which has the item titles as its keys, and use the corresponding HTML as its values. The first part of this setup is shown in [Listing 13](#).

### Listing 13. Setting up your query and passing the results to a new view object

```
public function viewChannelAction()
{
    $filterGet = Zend_Registry::get('fGet');
    // ...
    $view->rssFeed = $rssFeed;

    $yahooNews = array();

    $key = 'YOUR_YAHOO_APPLICATION_KEY';
    $yahooQuery = new Zend_Service_Yahoo($key);

    foreach ($rssFeed as $item) {
        $yahoo->processItem($item);

        $yahooView = new Zend_View();
        $yahooView->setScriptPath('views');

        $yahooView->title = 'Local results from Yahoo!';
    }
}
```

```
try {
    $yahooView->results = $yahooQuery->newsSearch($item->title());
} catch (Zend_Service_Exception $e) {
    $yahooView->results = array('No results: ' . $e->getMessage());
}

$yahooNews[$item->title()] = $yahooView->render('yahoo.php');
}

$view->yahooNews = $yahooNews;

echo $view->render('viewChannel.php');
```

There are a few points to note of things you've added to this code that was not in the previous sections. The first is the addition of a `try...catch` block wrapped around the query. In the case where Yahoo! returns no results, `Zend_Service_Yahoo` causes `newsSearch` to return a `Zend_Service_Exception`. The `try...catch` block prevents an interruption in the program execution.

Another addition is the use of a second view, `yahooView`, to render the results. The idea is that by separating the Yahoo! template from the Chomp feed-reader template, you can easily reuse the template and add Yahoo! results elsewhere on the site. Modularity is the key to having a feature-rich application.

To create the view, you follow the same procedure as for rendering the page: Create a new `Zend_View` object, populate its properties, and render it using a template file, `yahoo.php`. Note that you call `setScriptPath` to tell the view that it can find the template in the views directory, along with the other templates. All that's left to do is to create the actual template, shown in [Listing 14](#).

#### Listing 14. File `views/yahoo.php` -- Used to render news results from Yahoo!

```
<?php
/**
 * This view is used to display yahoo.com results
 *
 * Parameters:
 * title - title for these results
 * results - an object of type Zend_Service_Yahoo_NewsResultSet
 */
?>
<h4><?php echo $this->title ?></h4>
<ol>
<?php
foreach ($this->results as $result) {
?>
<li>
<a href="<?php echo $result->Url ?>">
<?php echo $result->Title ?></a>
</li>
<?php
}
?>
</ol>
```

You can see some examples of the properties set in the results. The `results` object contains a list of `Zend_Service_Yahoo_NewsResult` objects, and you can loop through them using the `foreach` construction. Every web, news, and image resulting from `Zend_Service_Yahoo` is actually a subclass of the `Zend_Service_Yahoo_Result` base class, which has the following properties:

- `Title` (string) -- The title of the item
- `Url` (string) -- A URL used to link directly to the item
- `ClickURL` (string) -- A URL pointing to the item, directed through Yahoo!

According to the Yahoo! Developer Network documentation, both `Url` and `ClickUrl` link to the item, but `ClickURL` will first redirect through Yahoo.com, allowing Yahoo! to keep track of how its API is being used. Feel free to use either property.

Finally, you must include the HTML output in the Yahoo! renderer in the main page view. At the end of Listing 14, the output is assigned to the `yahooNews` property of the `view` object. All that's needed is to update your main page view (`viewChannel.php`) with the code in Listing 15.

### Listing 15. Positioning the Yahoo! results on the page

```
...
echo "<tr><td><a href=\"\$link\">\$entryTitle</a><br />
<?php
    if (isset( $this->yahooNews[ $item->title() ] ) ) {
        echo $this->yahooNews[ $item->title() ];
    }
?>
echo "</td>";
...
```

## Displaying in-depth results

Yahoo! doesn't just respond with the article title and URL. What is actually returned from the `newsSearch` method are objects of type `Zend_Service_Yahoo_NewsResult`, which have the following detailed properties:

- `Summary` (string) — A short summary of the article
- `NewsSource` (string) — The news source providing the article
- `NewsSourceUrl` (string) — The URL for the news source

- Language (string) — The language the article is in
- PublishDate (string) — The date the article was published as a UNIX® timestamp
- ModificationDate (string) — The date the article was last modified as a UNIX timestamp
- Thumbnail (Zend\_Service\_Yahoo\_Image) — Thumbnail for the article, if it exists

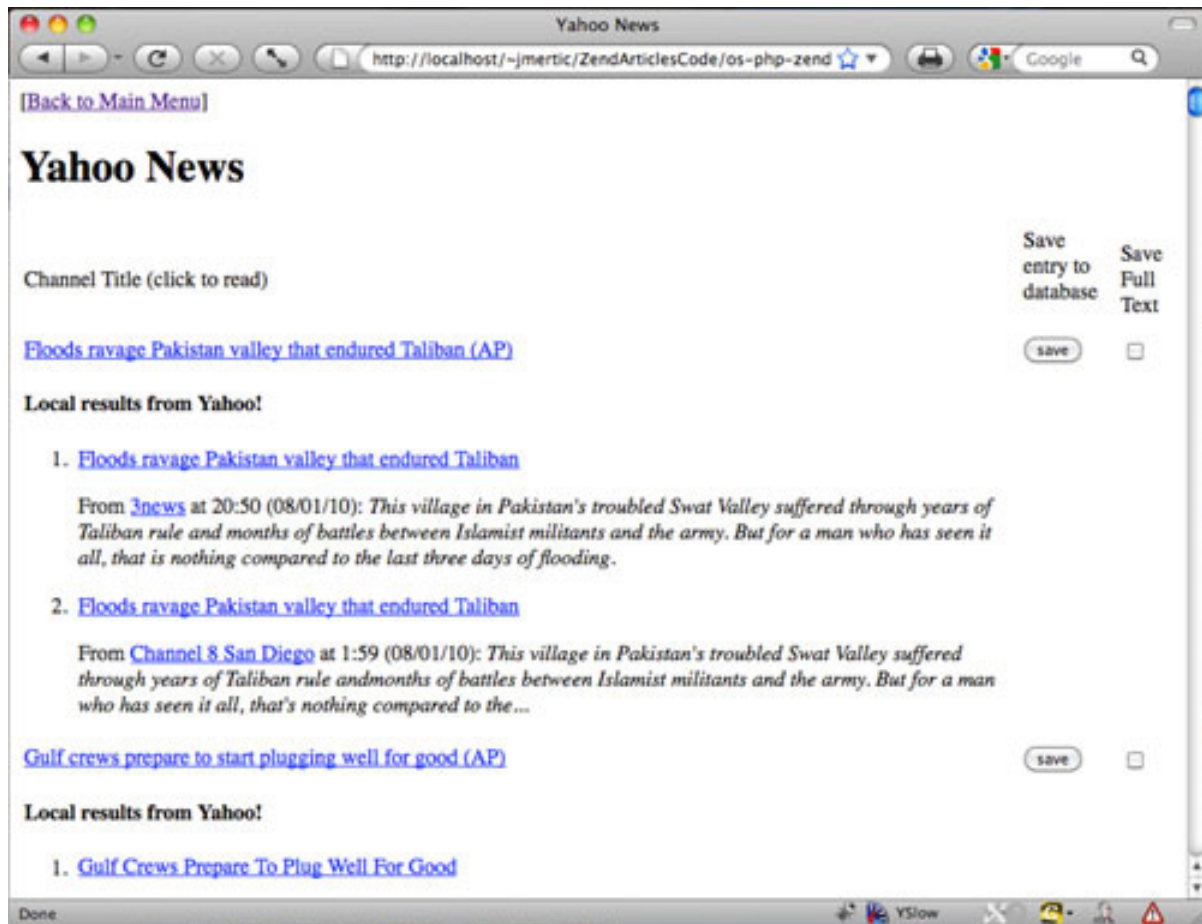
In [Listing 16](#), you update your Yahoo! view to include more information about what the source is and what time it was published.

### Listing 16. Information provided on the source and time published

```
<ol>
<?php
foreach ($this->results as $result) {
?>
<li>
<a href="<?php echo $result->Url ?>">
<?php echo $result->Title ?></a>
<p>
<?php
    echo 'From <a href="' . $result->NewsSourceUrl .
        '" title="' . $result->NewsSource . '">' .
        $result->NewsSource . '</a> ' .
        'at ' . date("G:i (m/d/y)", $result->PublishDate) .
        ': ';
?>
<em><?php echo $result->Summary ?></em>
</p>
</li>
<?php
}
?>
</ol>
```

[Figure 4](#) shows the results.

### Figure 4. News results from Yahoo! clearly formatted



## Section 7. Summary

As large web services provided by companies like Amazon, Flickr, Yahoo!, Twitter, Google and others become more commonplace, the web will continue to see enormous growth in number of innovative mashups. Books and photos are only the beginning. APIs for traffic info, sports scores, market prices, restaurant reviews, ski conditions, and much more will become easier to apply to your web application. Or perhaps you'll be inspired to start your own web services, potentially driving traffic and press to your site. But whatever your motivation, understanding that providing a RESTful service can, in many circumstances, be the best way to get a wider acceptance of your product.

You now have all the tools you need to integrate the `Zend_Services` package with your own project. As the Zend Framework matures, there will certainly be improvements and additions that will make it easier than ever to make your own

mashup, the way you've done here with the Chomp feed reader. Remember the key points:

- Identify the keywords you can pass to the service
- Integrate the service into your controller
- Construct a view that allows you to reuse the service in other places of your site

In Part 9, the final part of this series, you will use Ajax to retrieve and display the results gathered here, which will streamline the user's experience with Chomp.

## Downloads

Description	Name	Size	Download method
Part 8 source code	os-php-zend8.source.zip	13KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- This series kicks off with "[Understanding the Zend Framework, Part 1.](#)"
- The [Zend Framework](#) website maintains the [latest documentation](#).
- Read "[Mashups: The new breed of Web app](#)" for an introduction.
- To better understand the goals of the feed-reader project, read "[Introduction to Syndication, \(RSS\) Really Simple Syndication](#)," by Vincent Luria.
- You can see a good comparison of how much time Zend Framework will save you in "[Ajax RSS reader: Use Ajax to build an RSS reader](#)," in which an RSS reader is built from scratch in PHP.
- Developer Chris Laffra provides some interesting thoughts and experiments in Ajax and mashup creations in "[Considering Ajax, Part 2: Change your life with mashups.](#)"
- 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.
- A more in-depth discussion of stateless vs. sessioned architectures is discussed in the developerWorks article "[Resource-oriented vs. activity-oriented web services.](#)"
- "[Understanding Web Services specifications, Part 1](#)" provides an overview of SOAP, SOAP servers, and SOAP clients.
- [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.
- Professor Roy T. Fielding's [dissertation](#) first formalized the principle of the REST, which describes the network architecture of the web.
- Check out [Amazon's](#) database of books and retail goods, [Yahoo!'s](#) index of the web, and [Flickr's](#) quality photographs -- essential tools for any well-traveled surfer.
- If you need more information about the [ItemSearch](#) service or anything of the other web services provided by Amazon.com, read the [API Documentation](#).
- For more information about photo searches, user searches, or any other searches, check out the [Flickr API Documentation](#).
- Flickr uses a tagging system based on [del.icio.us'](#) "social bookmarking."
- The developerWorks tutorial series "[Learning PHP](#)" takes you from the most

basic PHP script to working with databases and streaming from the file system.

- The [PHP Function Reference](#) is another valuable resource.
- [Thought Storms ModelViewController](#) explains the MVC and the controversy and confusion surrounding it.
- The phpPatterns website explains [MVC from the PHP point of view](#).
- Visit IBM developerWorks' [PHP project resources](#) to learn more about PHP.
- 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.
- To listen to interesting interviews and discussions for software developers, be sure to check out [developerWorks podcasts](#).

### Get products and technologies

- Download the [Zend Framework](#) from [Zend Technologies](#).
- Download [PHP V5.x](#) from [PHP.net](#).
- Check out Yahoo!'s [Web services application](#).
- Sign up for a free [Flickr account](#).
- For the Yahoo! Application ID, you will need a [Yahoo! username](#).
- [Retrievr](#) is an experimental mashup using the Flickr API and a simple Flash drawing program to search for photos.
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.

### Discuss

- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

## About the author

Sean Kelly

Sean Kelly graduated with a degree in mathematics from Reed College. He is currently a Web application developer for ID Society, a full-service

Internet marketing agency in New York City. He is a supporter of open source content management systems, and contributes to Joomla! and the Wikipedia project.