



Easy Ajax in symfony

This PDF is brought to you by
SENSIOLABS 

License: Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Unported License
Version: symfony-ajax-1.1-en-2009-12-05

Table of Contents

Array

Easy Ajax in symfony

Overview

Symfony has Ajax helpers that make programming an elaborate interface a piece of cake. This tutorial will show you step-by-step how to create an Ajax-powered symfony application in minutes.

Introduction

Real lazy folks that can't stand reading long documentation are advised to watch the online screencast that demonstrates exactly what is written below.

Adding items to a shopping cart in common e-commerce applications isn't very close to the actual "add to cart" metaphor, since it requires clicking an "add to cart" button, watch a new page (the shopping cart), and then go back to the shop or checkout with buttons.

Ajax allows to get closer to the cart metaphor, by enabling drag-and-drop interactions and giving immediate visual feedback, without leaving the shop.

The target application of this tutorial will be a symfony ported version of the shopping cart demo published by script.aculo.us¹ in Rails². It uses the prototype³ JavaScript framework (bundled with symfony) and some script.aculo.us⁴ JavaScript that is the core of the JavaScript helpers.

Application Setup

First, create a `sfdemo` project, an `app` application and a `cart` module:

```
$ cd /home/steve
$ mkdir sfdemo
$ cd sfdemo
$ symfony init-project sfdemo
$ symfony init-app app
$ symfony init-module app cart
```

*Listing
1-1*

-
1. <http://script.aculo.us/demos/shop>
 2. <http://www.rubyonrails.com>
 3. <http://prototype.conio.net/>
 4. <http://script.aculo.us/>

Setup your web server to be able to access this new application (whether using a virtual host or an alias, as described in the web server setup⁵ chapter of the documentation). For this example, let's assume that this module is accessible via a localhost:

Listing 1-2 `http://localhost/cart/`

Congratulations, it says.

Your app must have access to the symfony JavaScript libraries. If your app doesn't work, check you can access these libraries within your browser (test `http://localhost/sf/prototype/js/prototype.js` for example). If not, you have 3 different ways to fix this problem:

- configure Apache with the following Alias:

Listing 1-3 `Alias /sf /$data_dir/symfony/web/sf`

- create a sf symbolic link in your web directory:

Listing 1-4 `$ cd /home/steve/sfdemo/web`
`$ ln -sf /$data_dir/symfony/web/sf sf`

- copy the JavaScript files in web/js directory:

Listing 1-5 `$ cd /home/steve/sfdemo/web`
`$ mkdir -p sf/js`
`$ cp /$data_dir/symfony/web/sf/prototype/js/*.js sf/js/`

The Main Page

First, you need to create a list of items to be purchased. To keep the project simple, the element list is accessed via a simple `getProducts()` method of the `cart` actions class. The shopping cart is a simple parameter of the `sfUser` object, set with the `Attribute` parameter holder⁶. Modify the `sfdemo/apps/app/modules/cart/actions/actions.class.php` to:

Listing 1-6

```
class cartActions extends sfActions
{
    public function executeIndex()
    {
        $this->getUser()->setAttribute('cart', array());
        $this->products = $this->getProducts();
    }

    private function getProducts()
    {
        return array('iPod black', 'iMac', 'iMac RC', 'iPod');
    }
}
```

The main page of the cart module will contain a list of items, and a zone to drag items to. This zone is the shopping cart. So open the template `sfdemo/apps/app/modules/cart/templates/indexSuccess.php` and write in:

5. http://www.symfony-project.org/cookbook/1_1/web_server

6. http://www.symfony-project.org/book/1_1/02-Exploring-Symfony-s-Code#Parameter%20Holders

```

<h1>symfony Apple store demo</h1>

<div id="shopping_cart">

  <h2>Products:</h2>

  <div id="product_list">
    <?php foreach ($products as $id => $title): ?>
      <?php echo image_tag('product'.$id, array(
        'id' => 'product_'.$id,
        'class' => 'products'
      )) ?>
    <?php endforeach; ?>
  </div>

  <h2>Cart:</h2>

  <div id="cart" class="cart">
  </div>

</div>

```

Listing
1-7

You can see that products are shown as images. Use the images available in this archive, and put them in the `sfdemo/web/images/` directory. In addition, part of the styling was done for you, so it is recommended that you upload this stylesheet to the `sfdemo/web/css/` directory and add a `view.yml` in the `sfdemo/apps/app/modules/cart/config/` directory with the following content:

```

all:
  stylesheets: [cart]

```

Listing
1-8

Now watch the chop and cart backdrop by requesting again:

```
http://localhost/cart/
```

Listing
1-9

Focus on the Cart

The cart content will change as you drag items to it. This means that the content of the cart in the template must be in an independent file. Use the `include_partial()` helper for that. The items in the shopping cart will be stored in divs with `float:left` style, so a clearing div is necessary after the container. So change the end of the `indexSuccess.php` template to:

```

<h2>Cart:</h2>

<div id="cart" class="cart">
  <div id="items">
    <?php include_partial('cart') ?>
  </div>
  <div style="clear: both"></div>
</div>

</div>

```

Listing
1-10

The `include_partial()` helper will include a `_cart.php` file, and look for this file in the `sfdemo/apps/app/modules/cart/templates/` directory. Create it with the following content:

```

Listing 1-11 <?php foreach ($sf_user->getAttribute('cart') as $product_id =>
$quantity): ?>
<div>
  <?php for ($i = 1; $i <= $quantity; $i++): ?>
    <?php echo image_tag('product'.$product_id, array(
      'class' => 'cart-items',
      'id' => 'item_'.$product_id.'_'.$i,
      'style' => 'position:relative'
    )) ?>
  <?php endfor; ?>
  (<?php echo $quantity ?> <?php echo $products[$product_id] ?>)
</div>
<?php endforeach; ?>

<?php if (!$sf_user->getAttribute('cart')): ?>
  nothing yet in your shopping cart.
<?php endif; ?>

```

If the cart contains items, they appear as images, as many times as they are added; the quantity is displayed after each series.

Now watch again the shopping cart at:

```

Listing 1-12 http://localhost/cart/

```

Well, there is not much change, it is still very empty... It's time to make things AJAX.

Add JavaScript Behaviors

Edit the `indexSuccess.php` template to require the JavaScript helper:

```

Listing 1-13 <?php use_helper('Javascript') ?>

```

Make the images draggable by adding the following call to the `draggable_element` JavaScript helper:

```

Listing 1-14 <?php foreach ($products as $id => $title): ?>
  <?php echo image_tag('product'.$id, array(
    'id' => 'product_'.$id,
    'class' => 'products'
  )) ?>
  <?php echo draggable_element('product_'.$id, array('revert' => true)) ?>
<?php endforeach; ?>

```

This adds a 'draggable' behavior to each of the images of the list of products. The `revert` option will make images go back to their origin position when released (unless received by a receiving element).

Now, define the cart as a receiving element. You just need to define which part of the template will have to be updated when the event occurs, which action will be called for its content, and which type of draggable elements can be dragged into it. Use the `drop_receiving_elements` JavaScript helper for that:

```

Listing 1-15 <?php echo drop_receiving_element('cart', array(
  'update' => 'items',
  'url' => 'cart/add',

```

```
'accept'      => 'products',
) ?>
```

Now try again, and move the products to the cart: it works. When a draggable item is dragged to the receiving element, an XMLHttpRequest is sent to the add action, and the result is displayed in the items div. The thing is, the add action of the cart module is not defined yet...

Define the Updating Action

Edit the `sfdemo/apps/app/modules/cart/actions/actions.class.php` to add an add action:

```
public function executeAdd()
{
    $tmp = split('_', $this->getRequestParameter('id', ''));
    $product_id = $tmp[1];

    $cart = $this->getUser()->getAttribute('cart');
    if (!isset($cart[$product_id]))
    {
        $cart[$product_id] = 1;
    }
    else
    {
        ++$cart[$product_id];
    }
    $this->getUser()->setAttribute('cart', $cart);
    $this->products = $this->getProducts();
}
```

*Listing
1-16*

This action looks for the parameter sent by the JavaScript (the id of the dragged item) and adds it to the cart.

The result of the add action will be the `addSuccess.php` template. It is a simple inclusion of the `_cart.php` partial, but this time it is necessary to pass the products as a parameter :

```
<?php include_partial('cart', array('products' => $products)) ?>
```

*Listing
1-17*

This template must not use the global layout, so edit the `view.yml` in the `sfdemo/apps/app/modules/cart/config/` directory, and write in:

```
addSuccess:
  has_layout:  off

all:
  has_layout:  on
  stylesheets: [cart]
```

*Listing
1-18*

Try it on: you can now add items to the cart by dragging them.

Focus on Usability

You could stop now, but this shopping cart has a big default: while the cart is updated, the interface doesn't change and the user might be disoriented. This is a general issue of

asynchronous requests: an indicator zone has to be added to show that the request is being processed. In addition, nothing tells the user when the dragged item is considered accepted by the cart, so the hover style of the cart div also has to be defined.

To do that, edit the `indexSuccess.php` template and write in:

```
Listing 1-19 <div style="height:20px">
            <p id="indicator" style="display:none">
              <?php echo image_tag('indicator.gif') ?> updating cart...
            </p>
          </div>
```

Save the 'indicator.gif' image file⁷ to your `sfdemo/web/images/` directory.

Now, modify the `drop_receiving_element()` JavaScript helper call in the same template to show this new indicator while requests are processed and declare the hover style:

```
Listing 1-20 <?php echo drop_receiving_element('cart', array(
            'update'      => 'items',
            'url'         => 'cart/add',
            'accept'     => 'products',
            'script'     => 'true',
            'hoverclass' => 'cart-active',
            'loading'    => "Element.show('indicator')",
            'complete'  => "Element.hide('indicator')"
```

Conclusion

The complete source of the demo can be downloaded⁸ and is available online⁹. You will notice a few minor differences with the code described in this tutorial (including a trash box), but the core behaviors are the same.

Until the full documentation of the JavaScript helpers is released, you can find more information about them in the `script.aculo.us` documentation¹⁰.

7. <http://www.symfony-project.org/downloads/demo/cart/indicator.gif>

8. <http://www.symfony-project.org/downloads/demo/cart/project.tgz>

9. <http://www.symfony-project.org/demo/cart.html>

10. <http://wiki.script.aculo.us/scriptaculous/list?category=Controls>