Using Ajax with PHP and Sajax

How the Simple Ajax Toolkit can integrate your server-side PHP with JavaScript

Skill Level: Intermediate

Tyler Anderson (tyleranderson5@yahoo.com)
Freelance Writer
Stexar Corp.

18 Oct 2005

Updated 05 Jul 2006

For years, the goal of creating a truly responsive Web application was hampered by one simple fact of Web development: To change the information on part of a page, a user must reload the entire page. Not anymore. Thanks to asynchronous JavaScript and XML (Ajax), we can now request new content from the server and change just part of a page. This tutorial explains how to use Ajax with PHP and introduces the Simple Ajax Toolkit (Sajax), a tool written in PHP that lets you integrate server-side PHP with JavaScript that makes this work.

Section 1. Before you start

This tutorial is for those interested in developing rich Web applications that dynamically update content using asynchronous JavaScript and XML (Ajax) with PHP, without having to refresh entire Web pages with each user click. This tutorial assumes basic PHP concepts, including the use of `if` and `switch` statements, and functions.

About this tutorial

© Copyright IBM Corporation 2005, 2006. All rights reserved.

You will learn about Ajax, as well as issues surrounding its usage. You will also build an Ajax application in PHP that will display panels of a section of a previously written tutorial. Clicking on a panel link will reload only the content section and replace it with the content of the selected panel, saving bandwidth and time loading the page. Then you will integrate the Simple Ajax Toolkit (Sajax) into your Ajax application, which will synchronize the use of Ajax, simplifying development.

Prerequisites

The following tools are needed to follow along:

Web server
: Pick any Web server and operating system. Feel free to use Apache 2.X or the IBM HTTP Server.

PHP
: You can follow along without PHP, but if you are interested in interacting with the sample application download PHP V5.

Sajax
: You will need Sajax. This is a single-file library of PHP functions used in this tutorial.

Web browser
: You will need a Web browser that supports JavaScript. These include Mozilla, Firefox, Opera, and Microsoft Internet Explorer.

Section 2. Overview

Before diving in, let's meet Ajax, the sample PHP application, and Sajax.

Ajax

Ajax allows Web developers to create interactive Web pages without the bottleneck of having to wait for pages to load. Through Ajax, you can create applications that, with a click of a button, will replace content in one section of a Web page with totally new content. The beauty of it is that you don't have to wait for the page to load, except for the content to load for that single section. Take Google Maps, for example: You can click and move the map around without having to wait for page loads.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Issues with Ajax

There are things to watch out for when using Ajax. Like any other Web page, Ajax pages are bookmarkable, which can create problems if requests are done with `GET` vs. `POST`. Internationalization and the rising number of encoding schemes makes standardizing these encoding schemes increasingly important. You will learn about these important issues in this tutorial.

The sample PHP application

You will create an application first in Ajax, then in Sajax to show the benefits of using this tool kit. The application is a section of a previously written tutorial with section links. It will be used as an example to show you the advantages of using Ajax. Because as you click through the sections, they load asynchronously without having to wait for the rest of the page to load again. This sample application will also serve as an example to show you how to create your own Ajax applications.

Editor's note: In this tutorial, the author uses the creation of a developerWorks Web page as an illustration of Sajax's capabilities, not as a preview for any capability planned for developerWorks. If you are writing a developerWorks tutorial, please follow the author guidelines discussed in another article (see Resources).

Sajax

Say you want to create an Ajax application without having to worry about the intricate details of Ajax. Sajax is the answer. Sajax abstracts away from you, the Web developer, the high-level details of Ajax through the use of a library developed by the folks at ModernMethod. Deep down, Sajax works the same as Ajax. However, the technical details of Ajax can be ignored through the use of higher-level functions provided in the Sajax library.

Section 3. What is Ajax?

This section is a primer that will explain, with examples, the concepts of Ajax, including what happens when you click a link, and the HTML and JavaScript needed for an Ajax with PHP application. The next section will go deeper into actually creating the PHP application using the Ajax concepts you will learn in this section.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Behind the scenes

Ajax is a combination of asynchronous JavaScript and XML. It is asynchronous because you can click a link on the page, and it will load only the content corresponding to the click, leaving the header or any other desired information the same.

A JavaScript function is put to work behind the scenes when you click a link. This JavaScript function creates an object that communicates with the Web browser and tells the browser to load a specific page. You can then browse other content on the same page as normal, and when the new page gets completely loaded by the browser, the browser will display the content at a location specified by an HTML `div` tag.

CSS style code is used to create links with span tags.

The CSS style code

The CSS code is needed by the sample application so the span tags will appear as real links created using a conventional anchor (`<a href=... >`) tag and will be clicked as real links.

Listing 1. Specifying display information for span tags

```
...
<style type="text/css">
  span:visited{ text-decoration:none; color:#293d6b; }
  span:hover{ text-decoration:underline; color:#293d6b; }
  span {color:#293d6b; cursor: pointer}
</style>
```

These span tags are used in the sample application, and the color conforms to that used for links on all IBM developerWorks tutorials. The first line within the style tag specifies that when the link has been visited, the color will remain the same. Hovering over it will underline it, and the cursor will turn into a pointer, just like using regular anchor tags (`<a href... >`). Let's take a look at how to create links that refer to this CSS style code.

Creating links using the span tag

The links you will create in the Building the PHP application section will be used to communicate to the browser through JavaScript what content to go and fetch. They are not traditional links using anchor tags, but they are created using span tags. The look and feel of the span tag is determined by the CSS code in Listing 1. Here is an

© Copyright IBM Corporation 2005, 2006. All rights reserved.

example:

```
<span onclick="loadHTML('panels-ajax.php?panel_id=0',
                        'content')">Managing content</span>
```

The `onclick` handler specifies which script to run when this span tag is clicked. There are several other specifiers similar to `onclick` you can experiment with, including `onmouseover` and `ondblclick`. Notice the JavaScript function, `loadHTML`, is shown instead of a traditional `http://` link or a relative link created by listing `panels-ajax.php?` in the onclick field. You will learn about the `loadHTML`, function next.

The XMLHttpRequest object

If you are using Mozilla, Opera, or another browser in one of these genres, content will be dynamically fetched using built-in `XMLHttpRequest` objects. Microsoft's Internet Explorer browser uses a different object, which you will learn about next. They are used in essentially the same way, and providing support for both is only a matter of adding a few extra lines of code.

The `XMLHttpRequest` object is used to retrieve page content in JavaScript. You will use this code later in the sample application, along with the appendages to the `loadHTML` function that covers the `ActiveXObject`. See Listing 2 for usage.

Listing 2. Initializing and using the XMLHttpRequest object

```
...
<style>
<script type="text/javascript">
var request;
var dest;

function loadHTML(URL, destination){
    dest = destination;
    if(window.XMLHttpRequest){
        request = new XMLHttpRequest();
        request.onreadystatechange = processStateChange;
        request.open("GET", URL, true);
        request.send(null);
    }
}
</script>
...
```

The `destination` variable, passed as a parameter in Listing 2, communicates where the content loaded by the `XMLHttpRequest` object will go, denoted by a `<div id="content"></div>` tag. Then the code checks whether the `XMLHttpRequest` object exists, and if it does, it creates a new one. Next, the event handler is set to the `processStateChange` function, which is the function the

© Copyright IBM Corporation 2005, 2006. All rights reserved.

object will call on each state change. The rest of the request is setup using the `open` method that passes in the type of transfer, `GET`, and the URL that the object will load. The object is finally put to action by calling its `send` method.

The ActiveXObject

The `ActiveXObject` is used in place of the `XMLHttpRequest` object in Internet Explorer. Its function is identical to that of `XMLHttpRequest`, and even its function names are the same, as you can see in Listing 3.

Listing 3. Initializing and using the ActiveXObject

```
...
function loadHTML(URL, destination){
    dest = destination;
    if(window.XMLHttpRequest){
...
    } else if (window.ActiveXObject) {
        request = new ActiveXObject("Microsoft.XMLHTTP");
        if (request) {
            request.onreadystatechange = processStateChange;
            request.open("GET", URL, true);
            request.send();
        }
    }
}
</script>
```

In this case (if you are using Internet Explorer), a new `ActiveXObject` is instantiated of the `Microsoft.XMLHTTP type`. Next, the event handler is set, and its `open` function is called. The object's `send` function is then called, putting `ActiveXObject` to work.

The processStateChange function

The event handler, or callback function is the function described here. The purpose of callback functions is to be able to process changes in state that occur in a created object. In your case, the purpose of this function will be to process a change in state, verify that the object has reached a desired state, and read the contents of the dynamically loaded content.

The `processStateChange` function gets called by the `XMLHttpRequest` or the `ActiveXObject` objects when its state changes. When the object enters state 4, the contents of the page have been received (see Listing 4).

Listing 4. Processing state changes

© Copyright IBM Corporation 2005, 2006. All rights reserved.

```
...
var dest;

function processStateChange(){
    if (request.readyState == 4){
        contentDiv = document.getElementById(dest);
        if (request.status == 200){
            response = request.responseText;
            contentDiv.innerHTML = response;http://httpd.apache.org/download.cgi
        }
    }
}

function loadHTML(URL, destination){
...
```

When the XML HTTP object reaches state 4, the content is ready to be extracted and displayed at the desired location on the browser. The location is `contentDiv`, and it is retrieved from the document. If the request was good and received in good order, the status of the response will equal 200. The HTML response is held at `request.responseText`, and it is displayed in the browser by setting it equal to `contentDiv.innerHTML`.

If there were no errors in the transfer, then all went well and the new content will appear in the browser; otherwise, `request.status` will not equal 200. See Listing 5 for the error handling code.

Listing 5. Error handling

```
...
        if (request.status == 200){
            response = request.responseText;
            contentDiv.innerHTML = response;
        } else {
            contentDiv.innerHTML = "Error: Status "+request.status;
        }
...
```

Listing 5 will send information about the transfer error to the browser. You will use this function in your sample application as the callback function. Next, you will learn about the issues and differences between `GET` and `POST`.

GET vs. POST

`GET` and `POST` are two methods of making HTTP requests and passing variables during those requests. A developer should not arbitrarily choose which method to use because both have usage implications. `GET` requests embed the variables in the URL, meaning that they are bookmarkable. This has bad implications if the variables are meant to change a database, or buy something, etc. Imagine if you had accidentally bookmarked a page that had the URL to buy something, complete with your address, credit card number, and a $100 product, all embedded in the URL.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Revisiting such a URL would mean buying that item.

Therefore, a `GET` request should be made if the variables have no effect, meaning, you can reload all day long, and nothing should change. A variable suitable for a `GET` request would be a category ID. You can reload again and again, and that category will display again and again, having no detrimental effect.

`POST` requests, on the other hand, should be used when the variables have an effect on a resource such as a database and for security of a person's personal information. In the hypothetical case of buying a $100 product, you should use a `POST` request. If you bookmark the checkout page with no variables in the URL, nothing will happen and you won't accidentally buy something you didn't intend to buy, or buy it again when you already have it.

The implications of `GET` vs. `POST` have the same effects in Ajax. It's important to understand the differences between `GET` and `POST` requests while building this application, as well as your future applications. This will help you avoid one of the common pitfalls of Web application development.

Encoding methods

There are various ways to encode the transfer of data in HTTP, and XML only accepts a few of them. The one that maximizes interoperability is UTF-8 because it is backwards-compatible with American Standard Code for Information Interchange (ASCII). Also, there are numerous international characters used in other countries encoded in ways that are not backwards-compatible with ASCII, nor are they suitable for placing in XML files without proper encoding.

For example, putting the string "Internationalization" in your browser transforms it to I%F1t%EBrn%E2ti%F4n%E0liz%E6ti%F8n, encoded using UTF-8. UTF-8 encoding of classical ASCII characters matches the 7-bit ASCII codes for the same characters, making UTF-8 an ideal choice for an encoding method selection.

This is important to know because you deal with encoding all the time in the transfer and receipt of documents through HTTP, including Ajax. Transfers using Ajax should also use UTF-8 encoding because interoperability will improve through standardization.

Section 4. Building the PHP application

This section will cover creating a PHP application using Ajax that will display a

© Copyright IBM Corporation 2005, 2006. All rights reserved.

tutorial with panel links. This section goes deeper into the application you already started in the What is Ajax? section.

Setting up the HTML document

Let's start by creating the application PHP file. Create a file, `ajax-app.php` and start by specifying the document type as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

This specifies the document type as HTML and the XML standard you will be using. You may notice later on that if you remove this line when all is said and done, the display of the side panel will change slightly.

Begin the HTML, specify the encoding and add the CSS style code as shown in Listing 6.

Listing 6. Display the links

```
...
<html><head><title>Create a Content Management System with PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style type="text/css">
  span:visited{ text-decoration:none; color:#293d6b; }
  span:hover{ text-decoration:underline; color:#293d6b; }
  span {color:#293d6b; cursor: pointer}
</style>
...
```

The title of the page will be the section name of a previously written tutorial. The encoding is specified using a metatag, and you have set up your HTML document to be able to use span tags as links. Next, you will create these links.

Creating links in the side panel

Before you create the links, you will need to set up the the side panel. Require the header file that you can download, along with all of the sample application files (see Resources):

```
...
</style>
<?php require('content/header.html'); ?>
```


© Copyright IBM Corporation 2005, 2006. All rights reserved.

This `header.html` file contains CSS, and other JavaScript and formatting information used by developerWorks tutorials. It also sets up the page so you can start adding the links. There is an introduction to the section, and nine panels, so you will need to create 10 links, as shown in Listing 7.

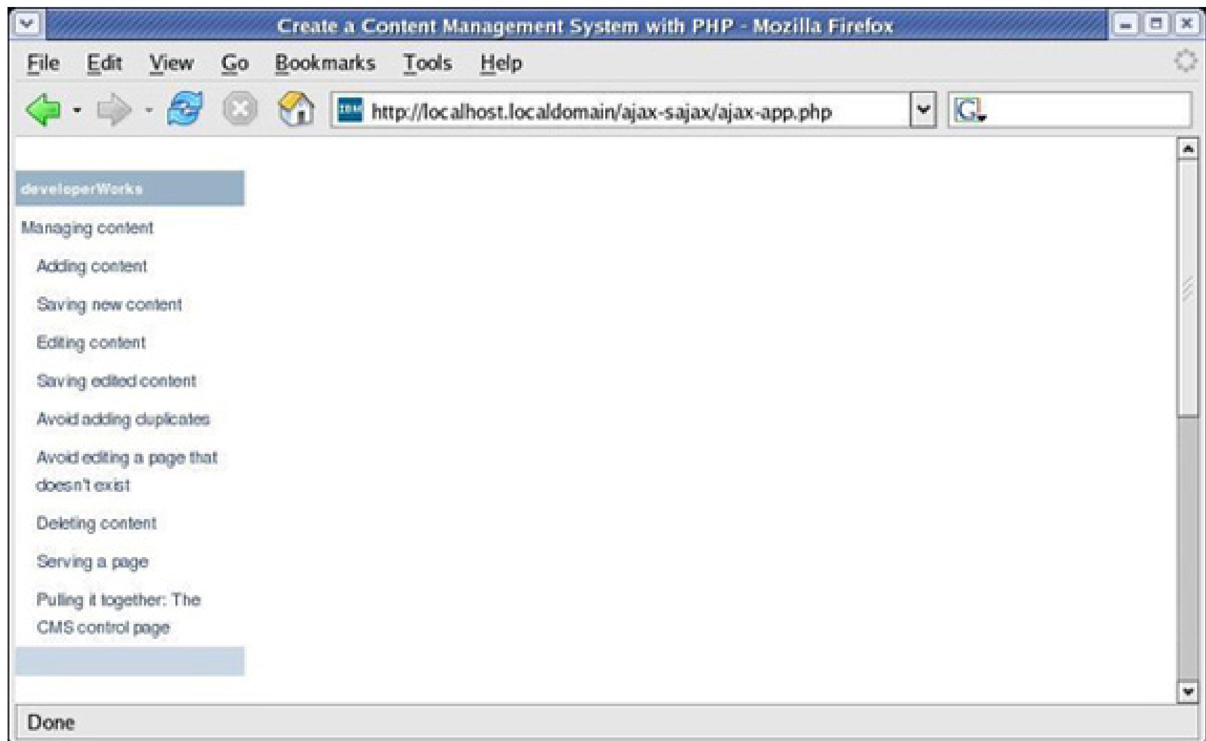
Listing 7. Create 10 links

```
...
<?php require('content/header.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=0',
                        'content')">Managing content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=1',
                        'content')">Adding content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=2',
                        'content')">Saving new content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=3',
                        'content')">Editing content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=4',
                        'content')">Saving edited content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=5',
                        'content')">Avoid adding duplicates</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=6',
                        'content')">Avoid editing a page
                                    that doesn't exist</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=7',
                        'content')">Deleting content</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=8',
                        'content')">Serving a page</span>
<?php require('content/between-link.html'); ?>

<span onclick="loadHTML('panels-ajax.php?panel_id=9',
                        'content')">Pulling it together:
                                    The CMS control page</span>
```

Every link calls `loadHTML`, passing the URL that has the panel ID, which will be used to determine what panel to load, with the second parameter specifying the destination `div` tag to place new content. Each link has some HTML code in between, and in order to abstract that away from the tutorial, the HTML was placed in `between-link.html`, which can also be downloaded (see Resources). See Figure 1 for sample browser output.

Figure 1. Displaying the side panel with links

© Copyright IBM Corporation 2005, 2006. All rights reserved.



The links are shown on the left side, just like a single section of a tutorial on developerWorks.

Initializing the content

The side panel is now up, so you now need to get the main content section ready by adding the section title:

```
...
                                    The CMS control page</span>
<?php require('content/pre-content.html'); ?>
```

This will add the section title, as well as print page links. Next, you will initialize the content section with all of the panels, one after another, within the content `div` tag, as shown in Listing 8.

Listing 8. Initializing the content

```
...
<?php require('content/pre-content.html'); ?>

<div id="content">
<?php
require('content/panel-0.html');
require('content/panel-1.html');
require('content/panel-2.html');
```


© Copyright IBM Corporation 2005, 2006. All rights reserved.

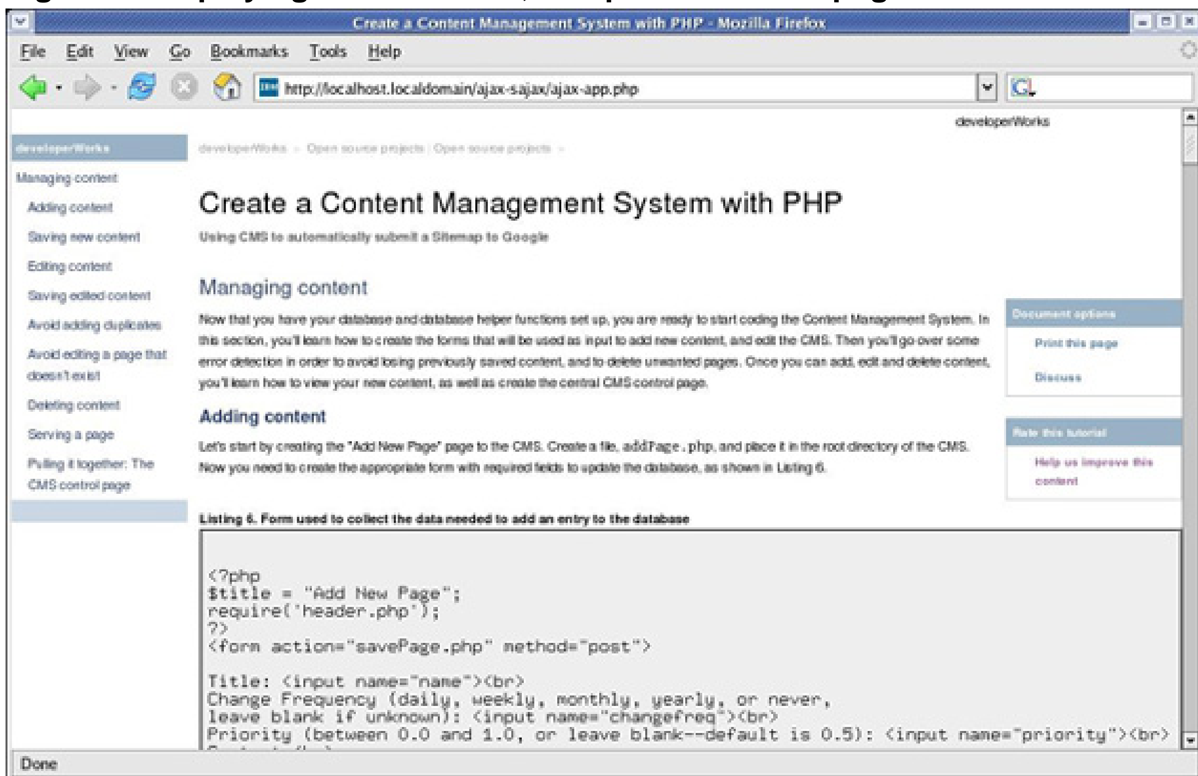
```
require('content/panel-3.html');
require('content/panel-4.html');
require('content/panel-5.html');
require('content/panel-6.html');
require('content/panel-7.html');
require('content/panel-8.html');
require('content/panel-9.html');
?>
</div>
```

All 10 panels will now be displayed one after another, in IBM's standard section format. To finish off the HTML, the next step is to require the footer file:

```
<?php require('content/footer.html'); ?>
```

This finalizes the HTML for the page. See Figure 2 for sample browser output.

Figure 2. Displaying the initialized, completed content page



The content is initialized, and the links are ready to call JavaScript instructions.

JavaScript: Adding the loadHTML function

It's now time to insert the asynchronous JavaScript and add the `loadHTML` function to the sample application, which is called by the links you created in the Creating links in the side section.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Listing 9. Set up the XML HTTP object

```
...
  span {color:#293d6b; cursor: pointer}
</style>
<script type="text/javascript">

var request;
var dest;
...
function loadHTML(URL, destination){
    dest = destination;
    if (window.XMLHttpRequest){
        request = new XMLHttpRequest();
        request.onreadystatechange = processStateChange;
        request.open("GET", URL, true);
        request.send(null);
    } else if (window.ActiveXObject) {
        request = new ActiveXObject("Microsoft.XMLHTTP");
        if (request) {
            request.onreadystatechange = processStateChange;
            request.open("GET", URL, true);
            request.send();
        }
    }
}

</script>
<?php require('content/header.html'); ?>
...
```

This code loads the URL specified in the link: panels-ajax.php. Next, you will insert the `processStateChange` function.

JavaScript: Adding the processStateChange function

Completing the JavaScript code, the `processStateChange` function is needed to place the loaded HTML content in the `div` tag you initialized in the Initializing the content section.

```
...
var dest;

function processStateChange(){
    if (request.readyState == 4){
        contentDiv = document.getElementById(dest);
        if (request.status == 200){
            response = request.responseText;
            contentDiv.innerHTML = response;
        } else {
            contentDiv.innerHTML = "Error: Status "+request.status;
        }
    }
}

function loadHTML(URL, destination){
...
```


© Copyright IBM Corporation 2005, 2006. All rights reserved.

You have completed the ajax-app.php file. Now you need to define the panels-ajax.php file pointed to by the panel links.

Returning content

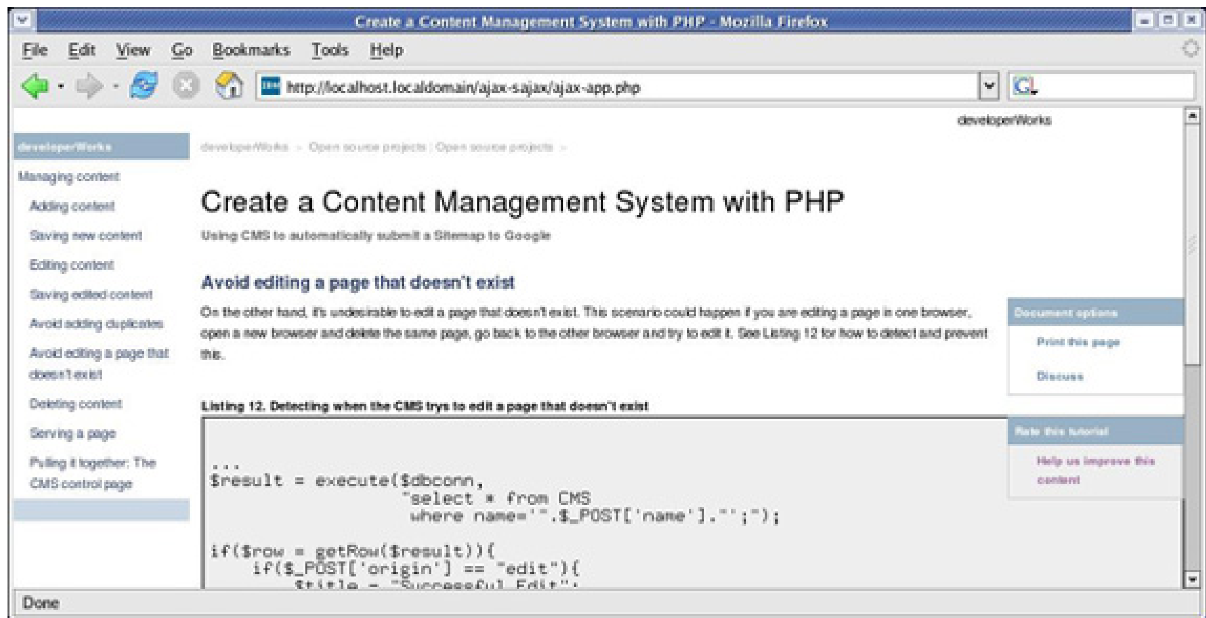
When you click on one of the links in your application, the Ajax code will try to load panels-ajax.php. Create this file, and place it in the same directory as ajax-app.php. This file will process the `panel_id` variable submitted to it using `GET`.

```
<?php

switch($_GET['panel_id']){
 case 0:
 case 1:
 case 2:
 case 3:
 case 4:
 case 5:
 case 6:
 case 7:
 case 8:
 case 9:
      require('content/panel-'.$_GET['panel_id'].'.html');
      break;
 default:
      print("No such category<br>");
}

?>
```

If the `panel_id` variable exists, the correct panel will be returned. The panel content HTML files are located in the downloadable zip file (ajax.sajax.source.zip) (see Resources), along with their accompanying figures. See Figure 3 for example of the browser output when clicking on the "Avoid editing a page that doesn't exist" panel.

Figure 3. Displaying single panel output

© Copyright IBM Corporation 2005, 2006. All rights reserved.



With the links live and Ajax at work, clicking on a link replaces the initialized content with the content specific to that panel. Next, you will add navigation links.

Adding navigation links

For the reader's convenience, add navigation links at the bottom of each panel. Clicking on the next link will cause the succeeding panel to load in the content section, replacing the current content.

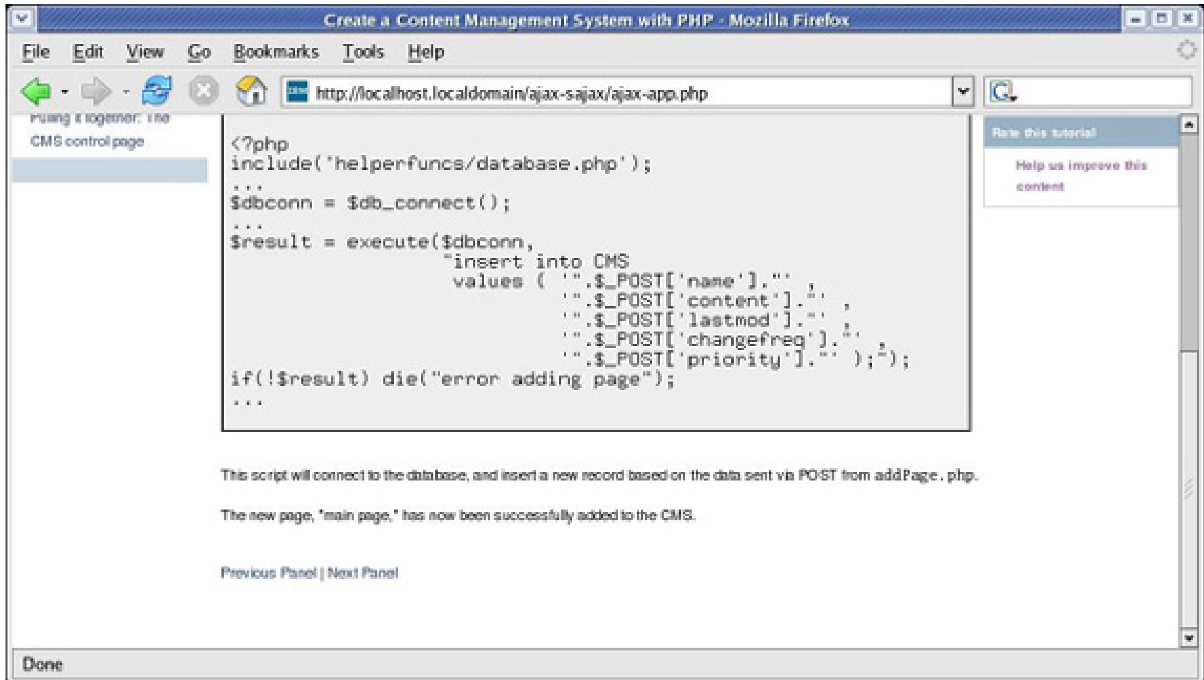
```
     require('content/panel-'.$_GET['panel_id'].'.html');
    $panel_id_next = $_GET['panel_id'] + 1;
     $panel_id_prev = $_GET['panel_id'] - 1;

     if($panel_id_prev >= 0){
       print("
<span onclick=\"loadHTML('panels-ajax.php?panel_id=".$panel_id_prev."',
                          'content')\">Previous Panel</span>
");
       if($panel_id_next <= 9)
         print(" | ");
     }

     if($panel_id_next <= 9){
       print("
<span onclick=\"loadHTML('panels-ajax.php?panel_id=".$panel_id_next."',
                          'content')\">Next Panel</span>
");
     }
     break;
 default:
```

The next link will simply be a link to the currently requested panel ID plus one, and minus one for the previous link. The previous panel link will be displayed if it exists,

Trademarks
© Copyright IBM Corporation 2005, 2006. All rights reserved.

meaning that it has a value greater than or equal to zero, and the next panel link will be displayed if it has a value less than or equal to nine. The actual links are created the same as the others you have already created, except they will change depending on the ID of the current panel. See Figure 4 for sample browser output containing the navigation links.

Figure 4. Displaying the navigation links



Clicking the links will navigate the section of the tutorial as you would expect, so when the reader reaches the bottom of a panel, clicking the next panel link will take them to the next panel. Again, without having to wait for the entire page to reload, the content of the current panel will be replaced with the next one.

This completes the application. Next, you will learn about how to integrate your application with Sajax.

Section 5. Integrating with Sajax

This section will cover what Sajax is, how it synchronizes the asynchronous JavaScript, and how your current Ajax application will be converted to a Sajax application.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

What is Sajax?

The Simple Ajax Toolkit (Sajax) is synchronous asynchronous JavaScript and XML. What makes it synchronous is that the details of the XML HTTP object used in your current Ajax application are abstracted away using the `Sajax.php` library file. This makes developing Ajax applications much easier because the chance of programming errors are reduced. Your links will also be much simpler because they will contain only function calls. Basically, Sajax is a modular way of making Ajax applications through defined and dynamic function calls, making the application development process smoother.

Synchronizing asynchronous JavaScript

There are several aspects of Sajax you will use in this section to synchronize Ajax. One of them is `sajax_init`, which initializes the Sajax library. Next, there is a `sajax_export` function you will call to notify Sajax that you have a "panels" content section, which you will create corresponding JavaScript functions for later. You can call `sajax_export` as many times as necessary for each of the dynamic content sections your application may require.

The next function you will use is the `sajax_handle_client_request`. This function initializes the Sajax data structures, preparing your application to handle client requests. You will also set the `$sajax_remote_uri`. This will be the URL where the client requests your application be sent to, similar to the embedded URL in the links of your Ajax application.

Finally, you will need to include the Sajax JavaScript functions within your JavaScript using the `sajax_show_javascript` function.

What's happening behind the scenes?

Now what's going on? Behind the scenes, Sajax acts essentially the same as Ajax. However, Sajax sets up the XML HTTP object up for you, simplifying your development job. It allows your application to easily have multiple content sections in a modular way using JavaScript functions. It's the Sajax package that makes Ajax development more productive, with the same behind-the-scenes functionality.

Initializing Sajax

Now you will begin the Sajax application. Copy your ajax-app.php file and rename the copy to sajax-app.php. Keep this file in the same directory as your ajax-app.php file. Add the following code to the beginning of the file:

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Listing 13. Initializing Sajax

```
<?
require("Sajax.php");

$sajax_remote_uri =
    "http://localhost.localdomain/ajax-sajax/panels-sajax.php";
sajax_init();
sajax_export("panels");
sajax_handle_client_request();
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
...
```

First, you will set up the remote URL. Then you need to initialize Sajax and export "panels." This will initialize Sajax to later create JavaScript functions to handle requests associated with the "panels" content.

Modifying the links

The links will now need to contain functions that will pass the panel ID to the appropriate JavaScript function.

Listing 14. Links for the panel links in the Sajax application

```
<?php require('content/header.html'); ?>

<span onclick="getPanel(0)">Managing content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(1)" >Adding content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(2)" >Saving new content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(3)" >Editing content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(4)" >Saving edited content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(5)" >Avoid adding duplicates</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(6)" >Avoid editing a page
                            that doesn't exist</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(7)" >Deleting content</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(8)" >Serving a page</span>
<?php require('content/between-link.html'); ?>

<span onclick="getPanel(9)">Pulling it together:
                            The CMS control page</span>
```


© Copyright IBM Corporation 2005, 2006. All rights reserved.

```
<?php require('content/pre-content.html'); ?>
...
```

Clicking on the links will now call a different JavaScript function, which you will add next.

Replacing the JavaScript

Before proceeding, remove the JavaScript from the Ajax application. You will need to add new functions, and these functions will have nothing to do with the XML HTTP object. Add the following, in place of the current JavaScript functions:

Listing 15. New JavaScript functions

```
...
<script type="text/javascript">
    <?php sajax_show_javascript(); ?>

    function getPanel_cb(content) {
        document.getElementById('content').innerHTML = content;
    }

    function getPanel(panel_id){
        x_panels(panel_id, getPanel_cb);
    }
</script>
...
```

The first line makes a call to `sajax_show_javascript`, which imports the JavaScript functions needed by Sajax, and the other `getPanel` function you will add. These added JavaScript functions bring the core of the Sajax application to the browser, as PHP is no longer in use once the page is first loaded by the browser.

Returning Sajax content

Now you will need to modify the panels-ajax.php file to accommodate the slight changes using Sajax. Copy and rename the panels-ajax.php file to panels-sajax.php and place it in the same directory as the others. Modify it, as shown in Listing 16.

Listing 16. Modifications for panels-sajax.php

```
<?php

if($_GET['rs'] == 'panels'){
    switch($_GET['rsargs'][0]){
    case 0:
...
    case 9:
      print("##");
```


© Copyright IBM Corporation 2005, 2006. All rights reserved.

```
        require('content/panel-'.$_GET['rsargs'][0].'.html');
        $panel_id_next = $_GET['rsargs'][0] + 1;
        $panel_id_prev = $_GET['rsargs'][0] - 1;

        if($panel_id_prev > = 0){
          print("
  <span onclick=\"getPanel(".$panel_id_prev.")\">Previous Panel</span>
  ");
  ...
          print("
  <span onclick=\"getPanel(".$panel_id_next.")\">Next Panel</span>
  ");
  ...
```

This file will check the variables submitted via `GET`. Notice that you sent "panels" to the `sajax_import` function. This should be the value of the `rs` variable in the `GET` array. If the value of `$_GET['rs']` is panels, then the `panel_id` variable is contained in `$_GET['rsargs'][0]`, which is the first parameter you sent to the `x_panels` function, auto-generated by the Sajax library.

Moving on, and before returning the appropriate panel, your code must print out any two characters, as there appears to be a bug in the Sajax library. These characters will not show up in the HTML source of the displayed Web page. Next, you will have to replace the rest of the references to `$_GET['panel_id']` with `$_GET['rsargs'][0]`. Last, you will need to modify the navigation links to look like the links you already modified in the sajax-app.php file. Swap the call to `loadHTML` with `getPanel`, passing the ID as before.

Figure 5. Sample browser output of the PHP application integrated with Sajax

© Copyright IBM Corporation 2005, 2006. All rights reserved.

The behavior and sample output of the application, shown in Figure 5, is the same as when you put it together with Ajax.

Section 6. Summary

Congratulations! You created an Ajax application in PHP and integrated it with Sajax successfully. Your application will save those that use it -- and your future asynchronous JavaScript applications -- a lot of bandwidth and time waiting for pages to load because the entire Web page will not have to load on each click, only the necessary content. This enables you to create rich interactive applications that will become more common.

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Downloads

Description	Name	Size	Download method
Source code for PHP application	os-phpajax.sajax.source.zip	117KB	HTTP

Information about download methods

© Copyright IBM Corporation 2005, 2006. All rights reserved.

Resources

Learn

- See "Ajax and scripting Web services with E4X, Part 1" for an introduction to ECMAScript for XML (E4X), a simple extension to JavaScript that simplifies XML scripting.
- See "Ajax and scripting Web services with E4X, Part 2" to learn how to use E4X to build the server side and implement simple Web services in JavaScript.
- Learn the concepts behind Ajax and the fundamental steps to creating an Ajax interface for a Java-based Web application in "Ajax for Java developers: Build dynamic Java applications."
- Read Sam Ruby's Ajax considered harmful blog to learn more about the `GET` vs. `POST` issue and the importance of encoding in UTF-8.
- Read this basic how to use CSS tutorial.
- ASCII stands for American Standard Code for Information Interchange. The table of ASCII codes is helpful.
- See this handy UTF-8 encoding reference for ASCII-compatible multibyte Unicode encoding.
- 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.

Get products and technologies

- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Visit the IBM developerWorks Ajax blog to read what your peers are saying about Ajax.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Tyler Anderson

Tyler Anderson graduated with a degree in computer science from Brigham Young University in 2004 and is currently in his last semester as a master's student in computer engineering. In the past, he worked

© Copyright IBM Corporation 2005, 2006. All rights reserved.

as a database programmer for DPMG.com, and he is currently an engineer for Stexar Corp., based in Beaverton, Ore.

© Copyright IBM Corporation 2005, 2006. All rights reserved.