PRAjax – PHP Reflected Ajax

Developer Manual

Index

What is PRAjax?

PRAjax in short

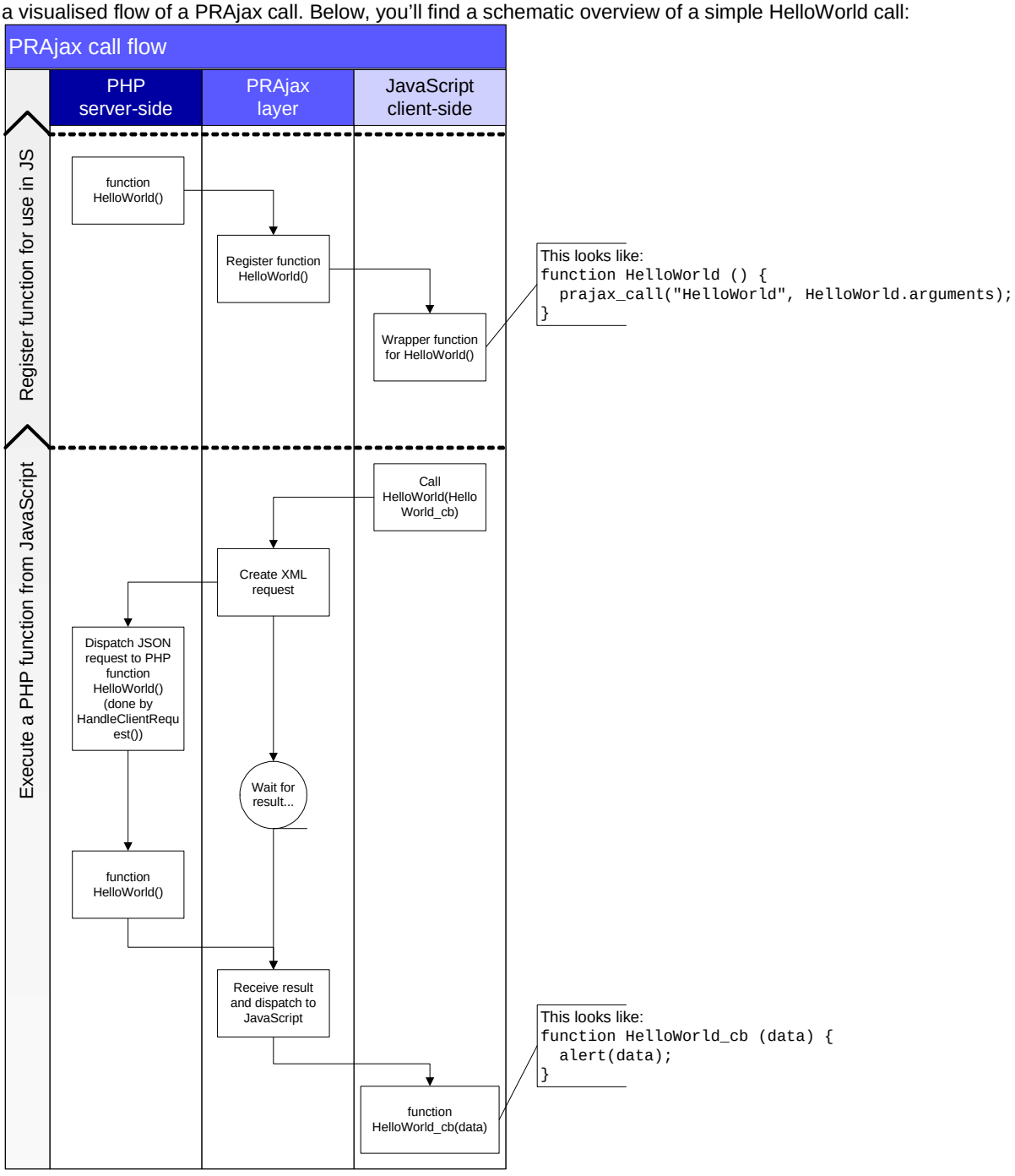
PRAjax is short for PHP Reflected Ajax. It is a helper class for reflecting PHP code into JavaScript and asynchronous methods and vice-versa.

The most significant features of PRAjax are:

- Allows easier Ajax programming with PHP
- PHP functions and objects can be reflected to JavaScript
- PHP objects can be used in JavaScript, both properties and (registered) functions
- PRAjax creates all wrappers for the developer
- Automatic conversion of return values and parameters between PHP and JavaScript
- Transparent client-side and server-side interaction

Schematic overview

In order to fully understand how PRAjax passes a call between client-side and server-side, it can be useful to see a visualised flow of a PRAjax call. Below, you'll find a schematic overview of a simple HelloWorld call:



Introduction

Requirements

To use PRAjax, the following components are required on server-side:

- HTTP web server supporting PHP
- PHP version 4.1 or higher
- Optional PEAR module Services_JSON (included with PRAjax when you do not have PEAR components installed)
- 100kB free webspace for the PRAjax libraries

Installation

Installation of PRAjax is simple. Basically, copy all files from the /bin folder of the PRAjax download to your webserver, eventually in a subfolder, and PRAjax is up and running.

Hello World

As a basic example, we'll create a simple Hello World example. It will display a button on client-side, and call a server-side function `HelloWorld`. Results of that function will be displayed in a JavaScript alert on client-side:

First of all, include PRAjax.php in your script, and instantiate it:

```
// Use bin or src?
$strSources = '../bin';

// Import PRAjax
require_once($strSources . '/PRAjax.php');

// Instantiate PRAjax
$objPRAjax = new PRAjax($strSources . '/');
```

After that, register the PHP function names yyou want to be able to call from JavaScript:

```
// Register functions
$objPRAjax->RegisterFunction('HelloWorld');
```

Optionally, you can show a wait cursor when server-side code is being executed:

```
$objPRAjax->ShowWaitCursor();
```

Very important: handle client requests, preferably after registering objects and functions for PRAjax:

```
// Handle PRAjax client request
$objPRAjax->HandleClientRequest();
```

And finally, our server-side function "HelloWorld" which we registered through PRAjax:

```
// Functions
function HelloWorld() {
  return "Hello World! " . date("Y/m/d H:i:s");
}
```

Now we can start writing our client-side code. Important thing is to include the PRAjax JavaScript code by using `<?php $objPRAjax->GetJavaScript(); ?>` in your page head tag.

Also, add your callback functions. A callback function is a function that receives the server's output and can process it. In our HelloWorld-case, we want to `alert()` a message from the server.

```
<html>
  <head>
        <title>Hello World</title>

        <?php $objPRAjax->GetJavaScript(); ?>
        <script language="JavaScript">
        <!--
        // Callback functions
        function HelloWorld_cb(data) {
              alert(data);
        }
        // -->
        </script>
  </head>

  <body>
        <form name="frmForm">
              <h1>Hello World</h1>
              <input type="button" value="Hello World"
                    onclick="HelloWorld(HelloWorld_cb);">
        </form>
  </body>
</html>
```

Make sure that when you call your server-side function, the last argument MUST be a callback function. If you do not want a callback, use an empty function (`PRAjax.Nothing`), or let PRAjax fix this automatically.

Take a look at the provided examples in the distribution. These cover most aspects and features of PRAjax development. Examples can be found in the /examples folder.

Advanced features

Add external client-side script

When you want to load an external JavaScript file on your PRAjax-enabled webpage, you might want to use one of the following functions:

- Client-side: `PRAjax.AddScript('file.js');`
- Server-side: `$objPRajax->AddScript('file.js');`

Note that the script path passed to PRAjax is relative to the path your webpage resides on.

Alert client-side on server-side

On the server-side, you can issue an `alert()` on client-side. Use the following code: `$objPRajax->Alert('This is an alert!');`

Execute script on client-side on server-side

If you want to perform a client-side script from server-side, this can be done using `$objPRajax->ExecuteScript('document.write("test");');` . This can be useful if you want to hide a DOM element or something. Adviced is not to use this too much, to keep server and client code separated.

Check if a page load is an Ajax Call

When you want to know if a page load is an Ajax call, and not a simple HTTP request, check the function , `$objPRajax->isAjaxCall()` which returns true/false.

Set character set other than default UTF-8

Some people use different character sets than UTF-8. PRAjax uses UTF-8 by default, but be sure to add the necessary header in your HTTP response. All pages using PRAjax should output `header("Content-Type: text/html; charset=utf-8");`

When you want to use a different character set, make sure you output the right header, for example `header("Content-Type: text/html; charset=ISO-8859-1");`, and that you inform PRAjax to use that character set, for example `$objPRAjax->setCharset('ISO-8859-1');`

Use a different target for specific calls

Sometimes, it can be usefull to fire a PRAjax call to another script than the originating one. For example, a news ticker updating every few seconds, can benefit from getting its data from another page. This way, the same script can be used on multiple pages showing the news ticker.

Performing a call on another script is easy, and can be done per-function on the client-side:

```
<input type="button" value="Simple Hello World"
onClick="HelloWorld(HelloWorld_cb,
'setPRAjaxTarget:example_helloworld.php;');">
```

This code calls the `HelloWorld()` function registered in example_helloworld.php.

Make sure you register the function `HelloWorld()` in your originating page AND in your target page. Browse the examples folder for a concrete example of an external call.

Use a different transport method than the one provided by PRAjax

Some people use prototype.js or other, similar JavaScript libraries providing XMLHttpRequest transfers. PRAjax uses its own transport, but can also use other transports, for example the one provided by prototype.js. PRAjax does not load its own transport to the client when you want to do that, saving a few bytes in download size for each visitor of your website.

If you want to use another transport, make sure you inform PRAjax about that on the server-side:

```
$objPRajax->setUsePRAjaxTransport(false);
```

Afterwards, you should create a wrapper for the 3rd-party transport. Here's a wrapper example for prototype.js:

```
// Custom transport class, must implement
```

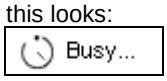
```
// PerformRequest(uri, post_data, callbackFunction,
//                                          errorFunction, openCallCount)
function CustomTransport () {}
CustomTransport.prototype.PerformRequest = function (uri, post_data,
                           callbackFunction, errorFunction, openCallCount) {
        var ajax = new Ajax.Request(uri, {method: 'post',
                         parameters: post_data, onSuccess: function (xmlObj) {
       callbackFunction(xmlObj.responseText); }, onFailure: errorFunction } );
}
```

You should also override a PRAjax method, namely `PRAjax.CreateTransfer`. This is necessary to use your wrapper function:

```
// Transport factory
PRAjax.CreateTransfer = function () {
 return new CustomTransport();
}
```

Show a busy-message when loading

When busy performing a PRAjax request, PRAjax can automatically show a message on client-side, informing that a request is being made. An example of how this looks:



An effect like that can be achieved by passing PRAjax the right information on server side:

```
$objPRAjax->ShowBusyMessage(true, '<img src="busy.gif"> Busy…',
'prajax_status');
```

The first parameter is a boolean, which is true/false if you want to show the busy message.
The second parameter is a string containing the message to show when PRAjax is performing a request. This can be plain text, but also HTML code. Be sure to escape te right characters!
The third parameter contains the DOM element id of an element in which the preceding message should be shown. This is most likely a DIV that you place somewhere on your web page.

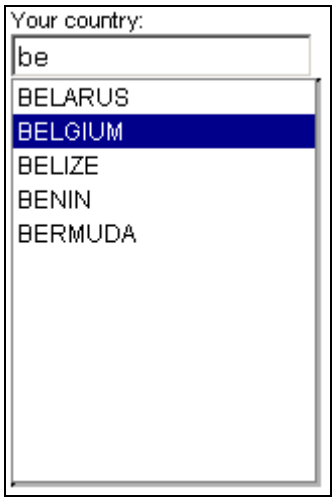
Show an hourglass pointer when loading

In addition to the busy message, you can also show an hourglass pointer when PRAjax is working. Use `$objPRAjax->ShowWaitCursor(true);` on server-side to display the hourglass pointer.

Extra's

Suggest component

Google first introduced Google suggest a while ago. This feature uses a dropdown list when typing, filling its list contents dependent on the value entered. PRAjax provides a component similar to that, using a PRAjax call for getting its data. Here is how it looks:



A suggest field can easily be added to your website. Find some example code underneath. The most important thing is that you have a server-side function which returns an array of values, in the example this is called `SuggestYear`. PRAjax will then automatically use that array as data source.

In the following code, necessary items for using the suggest component are printed in **bold**.

```
<?php
// Set the header
header("Content-Type: text/html; charset=utf-8");

// Use bin or src?
$strSources = '../bin';

// Import PRAjax
require_once($strSources . '/PRAjax.php');

// Instantiate PRAjax
$objPRAjax = new PRAjax($strSources . '/');

// Functions
function SuggestYear($pValue) {
      $aArray = array();
      for ($i = date('Y'); $i >= 1900; $i--) {
            if (substr($i, 0, strlen($pValue)) == $pValue) {
                  array_push($aArray, $i);
            }
      }
 return $aArray;
}

// Register functions
$objPRAjax->RegisterFunction('SuggestYear');

// Handle PRAjax client request
$objPRAjax->HandleClientRequest();
?>
<html>
<head>
 <title>Auto-suggest</title>
 <?php $objPRAjax->GetJavaScript(); ?>
```

```
 <script language="JavaScript"
src="../src/prajax_component_suggest.js"></script>
 <style type="text/css" media="all">
           @import "../src/prajax_component_suggest.css";
     </style>

</head>
<body>
<form name="frmForm" style="font-family: Arial, Helvetica, sans-serif; font-
size: 11px;">
Your birth year:<br>
<input type="text" name="txtBirthYear" id="txtBirthYear"
prajax_suggest="SuggestYear" style="width: 150px;"><br>
</form>
</body>
</html>
```

UpdatePanel component

Some people already know Atlas. Atlas is an ASP.NET based Ajax framework, which introduced partial page rendering without much effort. PRAjax also supports this.

The principle is easy:

- Add the attribute `prajax_updatepanelform="true"` to forms you want to submit using PRAjax
- Add the attribute `prajax_updatepanel="true"` to DIV elements you want to update after form submission

PRAjax will then automatically hook itself into your page.

When a form is submitted, PRAjax will render the page as normal, and return only updated DIV elements containing modified contents. This way, PRAjax enables much smaller downloads than an initial page load.

In the following code, necessary items for using the UpdatePanel component are printed in **bold**.

```
<?php
// Set the header
header("Content-Type: text/html; charset=utf-8");

// Use bin or src?
$strSources = '../bin';

// Import PRAjax
require_once($strSources . '/PRAjax.php');

// Instantiate PRAjax
$objPRAjax = new PRAjax($strSources . '/');

// Handle PRAjax client request
$objPRAjax->HandleClientRequest();
?>
<html>
<head>
 <title>UpdatePanel</title>
 <?php $objPRAjax->GetJavaScript(); ?>
 <script language="JavaScript"
src="../src/prajax_component_updatepanel.js"></script>

</head>
<body>
<div style="width: 100%; height: 65px; background-color: #EEEEEE; font-family:
Arial, Helvetica, Sans-Serif; font-size: 9pt;">
This example shows a partial page refresh using the UpdatePanel component. Try
filling in your name. When you submit the form, PRAjax hooks the submit event
and submits the form in the background. The whole page is then fetched and
updated on-the-fly.
<br>
<a href="example_component_updatepanel.php.txt" target="_blank">View
code...</a>
```

```
</div>
<div id="divTest1" prajax_updatepanel="true">
 <form name="frmTest1" style="font-family: Arial, Helvetica, sans-serif; font-
size: 11px;" prajax_updatepanelform="true">
 Your name: <input type="text" name="txtName" id="txtName" style="width:
150px;">
 <input type="submit" value="Submit">
 </form>
</div>

<div id="divTest2" prajax_updatepanel="true" style="font-family: Arial,
Helvetica, sans-serif; font-size: 11px;">
<?=(isset($_POST['txtName']) ? 'You entered: ' . $_POST['txtName'] : 'The
submitted name will appear here...')?>
</div>

<div id="divTest3" style="font-family: Arial, Helvetica, sans-serif; font-
size: 11px;">
<?=(isset($_POST['txtName']) ? 'You entered: ' . $_POST['txtName'] : 'The
submitted name will not appear here, because this is not an UpdatePanel...')?>
</div>

</body>
</html>
```

About

The PRAjax library is written and maintained by Maarten Balliauw. Originally, it had its own encoding and decoding mechanism for passing data between client and server. This quickly evolved to JSON, which provides a smaller size for transferring data. PRAjax has been used by myself on different sites, and provides a convenient, semi-transparent layer between the client and the server.

PRAjax contains 2 external libraries:

- Services_JSON, a PEAR library providing JSON conversions
  Authors:
  Michal Migurski <mike-json@teczno.com>
  Matt Knapp <mdknapp@gmail.com>
  Brett Stimmerman <brettstimmerman@gmail.com>
  http://pear.php.net/pepr/pepr-proposal-show.php?id=198
- SAXParser, a class providing HTML parser functionality
  Author:
  Alexey G. Piyanin <drdrzlo@mail.ru>

Visit the PRAjax website at http://prajax.sf.net.