The Dynamic Duo of APEX and AJAX

Oracle Database Tips by Donald Burleson

By Steve Karam

Steve Karam is the author of "Oracle and AJAX", and "Easy Oracle Jumpstart" by Rampant TechPress.

You can buy them direct from the publisher for 30%-off.

If you?re in the web application industry, you?ve probably heard of AJAX. AJAX is short for Asynchronous JavaScript and XML, and is a technology that enables the web to work in real time.

JavaScript has been around for years, and has enabled web developers to do amazing things, from populating text boxes on demand to changing an image on the fly when you hover your mouse over it. However, there has always been one thing missing from the web, and that is the ability to dynamically query a database on the fly without reloading the page you?re on.

There are many reasons for this, but the simplest explanation is that once a web page loads, it's now running on the client's computer. For instance, when you visit my blog at http://www.oraclealchemist.com, the PHP code that powers the site will query the database, build a page, and then display that page in your browser. Your browser receives the page in standard HTML and is therefore able to display it.

However, with the advent of AJAX, we now have more options. In this example, I will show you how to use AJAX in tandem with Oracle Application Express (APEX) to make a simple page using what is called 'type-ahead? querying. Don't worry if you don't get it yet; details are on the way!

Why Go Asynchronous?

Asynchronous is defined as ?Lack of temporal concurrence; absence of synchronism.? What this means to us is that multiple things can occur at the same time without being dependent upon each other. The standard web process is shown below:

Now with AJAX, we can perform the following:

Notice that we are not drawing any arrows to ?Process Inputs? or ?Modify Page.? That is because these processes are not dependent on any of the first three core processes, nor are they truly associated with them. They are simply asynchronous processes that can be called or not as necessary.

The AJAX Basics

Generally, an AJAX process is a simple JavaScript script that is called by performing some sort of action on a web page, such as selecting an item from a drop down list. This script will call another page that does nothing more than displays text. The JavaScript will take the text that it 'sees? and use it to re-process the original page.

For instance, when a drop down or <SELECT> is used on a web page, we can add an action called onChange, which is a JavaScript call activated when the select list is used. With AJAX, we can make onChange run a JavaScript that calls a page that queries a database and displays the results, allowing the JavaScript to display it as programmed somewhere on the page. Usually the place it will be displayed is a <DIV> area. For instance:

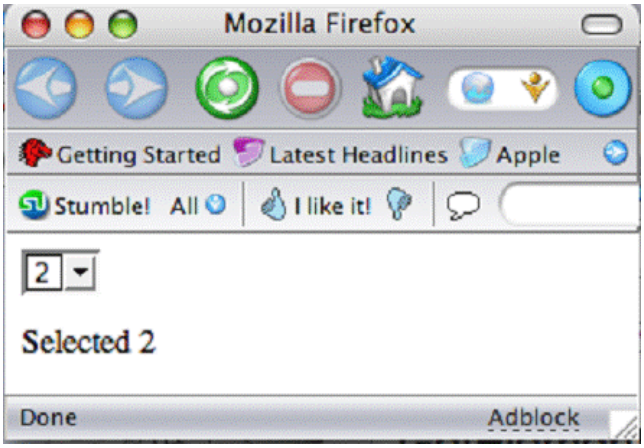

```
<DIV ID=?DisplayResults?></DIV>
```


This is an empty area on the page called DisplayResults. There is no text, images, or anything else inside of this DIV element, but JavaScript is able to put things in it. Consider the following code:

```
<html>
<head>
<script type="text/javascript">
  function returnResult(pInput) {
    document.getElementById("DisplayResults").innerHTML = pInput;
  }
</script>
</head>
<body>
<select onChange="javascript:returnResult(this.value);">
  <option value="Thank You For Selecting 1">1</option>
  <option value="Thank You For Selecting 2">2</option>
</select><br /><br />
<div id="DisplayResults"></div>
</body>
</html>
```

This is a very simple script that displays a drop down with two selections, ?1? and ?2?. When one of these is chosen, the onChange process of the <select> element calls a JavaScript function called ?returnResult,? passing in 'this.value? which can literally be translated as 'the value of this select list that you just changed.?

The JavaScript function itself is also very simple. It simply takes the input and sets it as the ?innerHTML? of the DisplayResults element. Using document.getElementById, we find the DisplayResults element, the element at the bottom of your page. We fill the innerHTML with the input, and we have a dynamic page!



Of course, this is not AJAX. We never called any external pages or queried any databases. All we have done is combined JavaScript and Dynamic HTML (DHTML) to form a simple changing page.

If you want to think of more advanced things that could be done if you were using AJAX, consider this: when selecting 1 or 2 from the dropdown, wouldn't it be much more interesting if it ran a query against a database and returned the rows where the primary key equaled your selection? If this appeals to you, read on!

Bringing AJAX to APEX

An APEX developer understands the basic flow of a page. A page is built with a combination of regions, processes, computations, items, templates, and such. At first it can seem very daunting, but as you progress you will find that it is very easy to slam out pages at a rapid rate.

However, any developer will also tap their fingers impatiently every time they select a drop down that must then call a process, which calls another page, which refreshes the screen, resulting in the need to have session safe variables and all sorts of other nuisances. Using AJAX, we can eliminate this need. Throughout the rest of the article we will be developing a small APEX page that will accomplish the following:

? Provide a text area displaying a list of USERNAMEs in the USERS table

? Provide an input box we can type in

? Narrow down the results in the text area based on characters we type in the input box

? Do so without refreshing the page or calling another page

To accomplish this, we will need to utilize three parts of APEX that make up nearly every AJAX use.

Part 1 - The Page

The page simply means the actual APEX page in which we define regions and items. For our example, we will define two simple regions and one item in our page (with page number 5)

Page Properties

Page Name: AJAX Entry

On Load:

```
onLoad="javascript:getUserList('');"
```

Region 1: Entry Area

Type: HTML Text

Region Source:

```
<script language="JavaScript1.1" type="text/javascript">

function getUserList ()

{

  var ajaxRequest = new htmldb_Get(null,&APP_ID.,'APPLICATION_PROCESS=getUsers',0);

  ajaxRequest.add('P5_USERNAME',html_GetElement('P5_USERNAME').value);

  ajaxResult = ajaxRequest.get();

  if(ajaxResult)

    {  html_GetElement('UserListDiv').innerHTML = ajaxResult  }

    else

    {  html_GetElement('UserListDiv').innerHTML = 'null'  }

  ajaxRequest = null;

}

</script>
```

Item in this Region: P5_USERNAME - Text Field

Region 2: Results

Type: HTML Text

Region Source:

```
<div id="UserListDiv"></div>
```

Explanation

In this section, we have detailed a page that you will create in APEX. We called this page AJAX Entry and gave it page number 5.

Direct your attention to ?Region 1: Entry Area? above. This is a simple HTML Text region, and we have defined a source for it, which consists of JavaScript code.

This code is a little different from the basics we went over earlier. It defines a function called getUserList. In getUserList, note the first line:

```
var ajaxRequest = new htmldb_Get(null,&APP_ID.,'APPLICATION_PROCESS=getUsers',0);

  ajaxRequest.add('P5_USERNAME',html_GetElement('P5_USERNAME').value);
```

This simple line of code packs a very powerful punch. It declares a variable called ajaxRequest, and sets it equal to a new instantiation of an object called htmldb_Get. For anyone who is unacquainted to object oriented code, this simply means we are calling something like a function, and assigning the ?object? it returns to the variable ajaxRequest.

In htmldb_Get, we pass the variables:

? null - Object ID. This is not necessary for the type of call we are making

? Flow - We set this to the application ID

? Request - We set this to APPLICATION_PROCESS=getUsers. You will set getUsers in detail soon

? Page - We pass 0 because we are not associating with a particular page in this example

The second and third lines of code are as follows:

```
ajaxRequest.add('P5_USERNAME',html_GetElement('P5_USERNAME').value);

ajaxResult = ajaxRequest.get();
```

Notice they both call functions prefixed with ?ajaxRequest.? This is the name of the variable we assigned in the previous step. ?add? and ?get? both became functions (or methods) inherited by our variable when they were instantiated in our previous call.

The first line adds a name-value pair to the object, setting a variable called P5_USERNAME equal to html_GetElement(?P5_USERNAME?).value, which is the value of the text typed into the field P5_USERNAME on the actual page.

Finally, we assign a variable called ajaxResult to the value of ajaxRequest.get(). Calling the get function will take everything it knows about the object: that it is calling an application process called getUsers, and it is passing in a variable called P5_USERNAME based off information from an input box on the webpage, and run it. The result of this call will be placed in the variable ajaxResult.

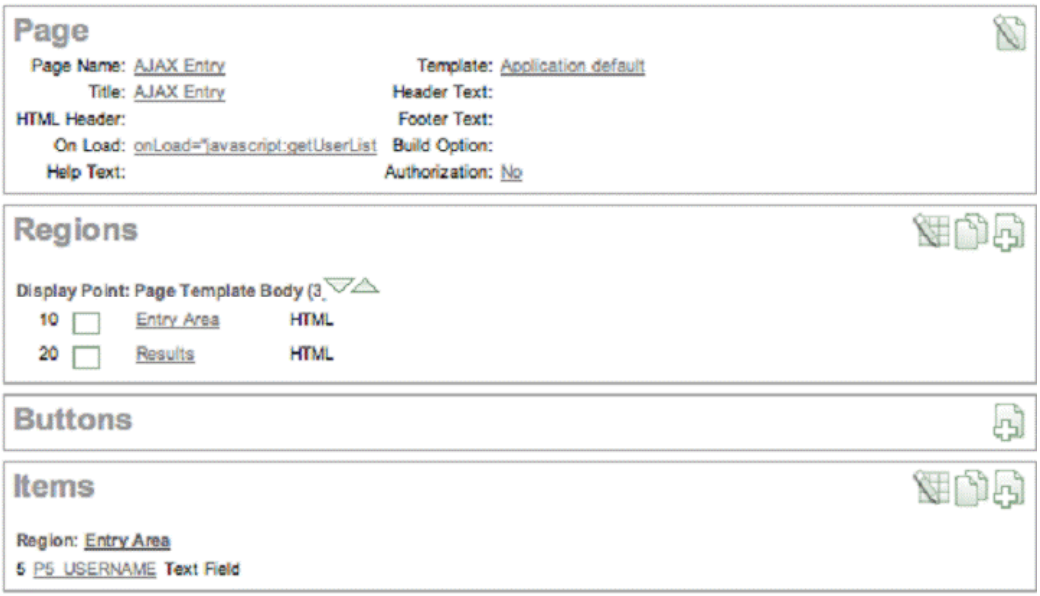
The last part of the script is:

```
if(ajaxResult)

    {  html_GetElement('UserListDiv').innerHTML = ajaxResult  }

    else

    {  html_GetElement('UserListDiv').innerHTML = 'null'  }
```

In this code, we will set the innerHTML of the DIV element UserListDiv equal to the result from the previous call to ajaxRequest.get().

And finally, we see that there is a Body On Load value in the definition of the page itself. This code simply tells the page to call the getUserList() function we have just described the moment the page is loaded, passing in a value of null.

In summary, our page looks like this:

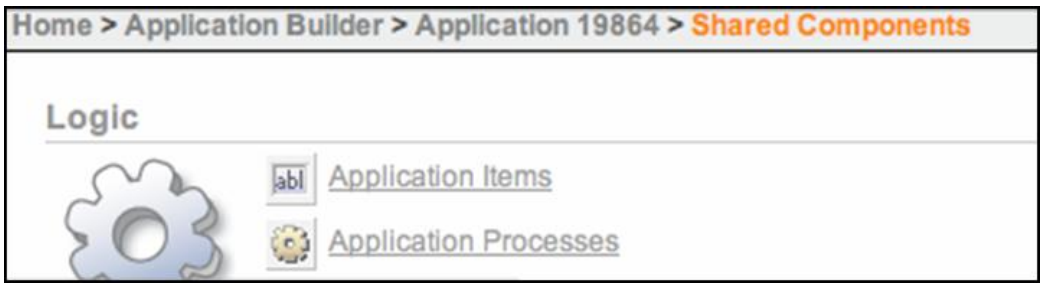


Now that we know what will be doing the calling (the page and text area) as well as HOW it will actually call, we need to define what will be called! From our code, you should be able to see it is some sort of application process called getUsers.

Part 2 - The Process

We must define a process that is not related to our page; it would not do to put the process on the page itself, because then it would not be able to run asynchronously.

Instead, we will define a Shared Component called an Application Process. We will create this process called getUsers, and set its process point to ?On Demand: Run this application process when requested by a page process.?



This means that the code herein will not be called unless specifically requested; like with our JavaScript code above!

The process is of type PL/SQL Anonymous Block, and will contain the following code:

```
declare

  lv_User_List varchar2(2000) := '';

begin

  begin

    for i in (select username from users where username like
upper(:P5_USERNAME) || '%') loop

      lv_User_List := lv_User_List || i.username || '<br />';

    end loop;

  exception when no_data_found then null;

  end;

  htp.prn(lv_User_List);

end;
```

Note that this is simply PL/SQL code. In this code, we create a for loop that finds all usernames from the users table where the username is like P5_USERNAME, concatenated with a percent sign. This means that whatever P5_USERNAME is set to (which we set in the ajaxRequest.add line in the code of Part 1) must be the starting characters of the username to pull back. Then it simply concatenates those into a variable called lv_User_List with a
 tag between each username for formatting.

Lastly, it uses the htp.prn function, which is a simple PL/SQL function that prints data to the web. In this case, it will print it back to the calling JavaScript, thus allowing us to assign it to the innerHTML of our div element.

Part 3 - The Activator

Having all these things would be pointless without interaction. This section is very simple. We will be going back to the page we created, our JavaScript being defined and the process it must call also being ready to go.

In this step, we will edit the attributes for the P5_USERNAME item we placed in the Entry Area region in step 1. We will add the following code to the ?HTML Form Element Attributes? property:

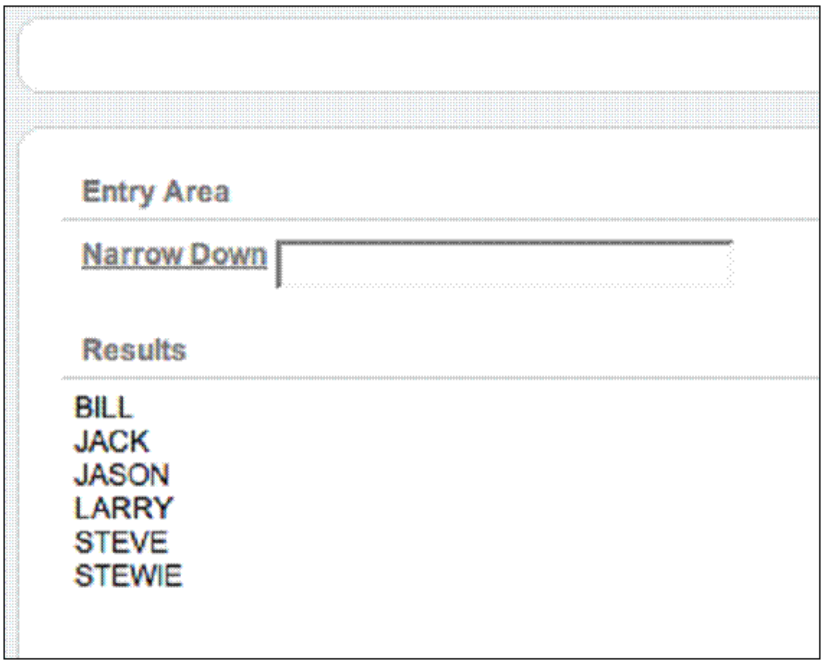
```
onKeyUp=?javascript:getUserList(this);?
```

Here, we tell the browser that every time a key is successfully pressed in the P5_USERNAME field, we wish it to call the JavaScript function that runs takes what we've typed that we created in step 1, and passes it into the application process we created in step 2, in order to fill the div element with the results.

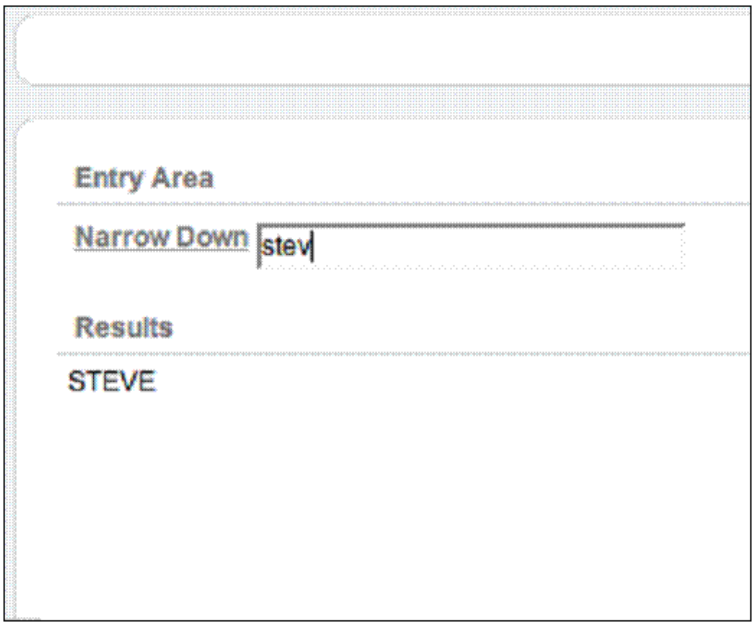
The Result

With our three parts in place, we are now ready to run the page and watch it work!

When we run the page for the first time, the On Body Load property from step one runs, and passes nothing into our application process. Therefore, our process returns all the rows.



By typing text in, but without having to press any buttons or wait for a screen refresh, we see the following:



Notice that the DIV area now only shows STEVE, because it's the only entry in the USERS table containing "stev" as typed in the text box.

Now we delete one letter, and we see:

And it shows us two records, STEVE and STEWIE, which both start with "ste".

All without refreshing or pressing a single button!

Conclusion

There truly is a lot to take in here, but if you go through the examples enough you will be able to start programming in AJAX in no time. Especially pay attention to the htmldb_Get JavaScript object that is included with APEX. Learn how to use it well, and you can do anything.

By using what you've learned here, you're only a short step from creating dynamic pages using select lists, textareas, and much more that can dynamically create reports, forms, and countless other application components. You can even set your application process to perform inserts and have it called when a button is pressed; no refresh row creation!

Steve Karam is the author of "Oracle and AJAX", by Rampant TechPress.

You can buy it direct from the publisher for 30%-off and get instant access to the code depot of Oracle AJAX scripts.

Burleson is the American Team

Note: **This Oracle documentation was created as a support and Oracle training reference for use by our DBA performance tuning consulting professionals. Feel free to ask questions on our Oracle forum.**

Verify experience! **Anyone considering using the services of an Oracle support expert should independently investigate their credentials and experience, and not rely on advertisements and self-proclaimed expertise. All legitimate Oracle experts publish their Oracle qualifications.**

Errata? **Oracle technology is changing and we strive to update our BC Oracle support information. If you find an error or have a suggestion for improving our content, we would appreciate your feedback. Just e-mail:**

DBA@remote-dba.net and include the URL for the page.

Burleson Consulting

The Oracle of Database Support

Oracle Performance Tuning

Remote DBA Services