Rico AjaxEngine Tutorial

The Rico JavaScript library provides a single JavaScript object, AjaxEngine, for adding Ajax to any HTML page.

What is Ajax?

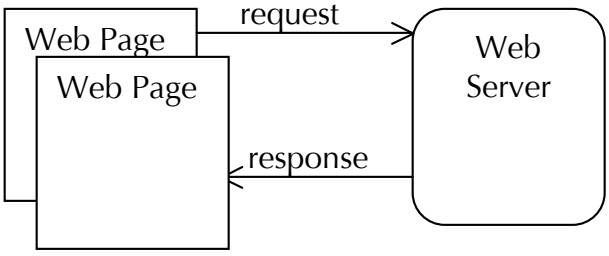
Widkipedia has the following definition for Ajax:

Traditional web applications allow users to fill out forms, and when these forms are submitted, a request is sent to a web server. The web server acts upon whatever was sent by the form, and then responds back by sending a new web page. A lot of bandwidth is wasted since much of the HTML from the first page is present in the second page. Because a request to the web server has to be transmitted on every interaction with the application, the application's response time is dependant on the response time of the web server. This leads to user interfaces that are much slower than their native counterparts.

AJAX applications, on the other hand, can send requests to the web server to retrieve only the data that is needed, usually using SOAP or some other XML-based web services dialect, and using JavaScript in the client to process the web server response. The result is more responsive applications, since the amount of data interchanged between the web browser and web server is vastly reduced. Web server processing time is also saved, since a lot of this is done on the computer from which the request came.

Comparing Ajax to Normal HTTP Mechanism

As stated above, the traditional way users interact with a page is to click a link (usually translates to an HTTP GET request) or click a submit button on a form (usually translates to an HTTP POST request). In the first case, a new page is requested and the browser refreshes with new content. In the second case, either a new page or the same page with modified values is returned.



In Ajax the request is made using the JavaScript function XmlHttpRequest. The request asks for a block of XML data (rather than a whole page to refresh) that the JavaScript code can handle in whatever way it sees fit.

For example, here are some ways the XML data could be interpreted:

- XML data that the JavaScript code parses to extract data. This data in turn is used to fill in field values or tables in a display.

- XML data that the JavaScript runs an XSLT process on to convert into HTML code to be inserted on the page somewhere
- XML data that holds JavaScript that the JavaScript code evaluates to issue commands
- XML data that contains HTML code that can be placed inside another HTML element on the page

This list is not exhaustive. The point is – Ajax allows you to make a server side call outside of the normal page refresh cycle. The data that is returned can be defined in any manner that XML allows and the way it is interpreted is open to the application's determined usage of this data.

Rico AjaxEngine

Rico provides a JavaScript object named *AjaxEngine* that simplifies using Ajax to update the contents within a page.

The engine provides the following features:

- A standard XML definition for the basic Ajax response
- A way to specify that a response is targeted for a specific HTML element
- Automatic update of the targeted HTML element's innerHTML with the contents of the response
- A way to specify that a response is targeted for a specific JavaScript Ajax response handler
- Standard mechanism for an Ajax response handler to receive response messages
- Support for multiple Ajax responses as the result of one Ajax request (multiple updates to elements and/or JavaScript objects).
- Simple API for registering the Ajax Request Handler and the Ajax Response Handler

Overview of AjaxEngine

Using the AjaxEngine to create dynamic content is very straightforward. There are three steps.

1. Register the Ajax request handler. This tells the AjaxEngine that a specific web service or server call is mapped to a request handler name.
2. Register the Ajax response handler. This is either specifies an HTML element as the target of the data being returned (in which case the contents of the response is HTML code) or it specifies a specific JavaScript object that will manage the response
3. Invoke the Ajax request when the appropriate event occurs in the interface.

Before starting, make sure that you have included both prototype.js and rico.js in your HTML page. You should include these two library files in your <head></head> section like this:

```
<script src="scripts/prototype.js"></script>
<script src="scripts/rico.js"></script>
```

Rolodex Demo
Lets use the Rolodex demo from the openrico web site (http://openrico.org/demos.page?demo=ricoAjaxInnerHTML.html).

Recall that by selecting persons from the list on the left, a rolodex card is displayed on the right hand side.

For this demo we built a server-side process to handle requests for information about a person. When the person is selected in the list we use Rico's AjaxEngine to call this request handler and then we process the response containing personal information and display it as a rolodex card.

Step 1: Register a Request Handler

The first thing we must do is register this Request Handler with the Ajax engine. We do this by associating the URL for a request handler with a given name. This name will serve as the identifier for the Ajax request handler.

```
ajaxEngine.registerRequest( 'getPersonInfo', 'getPersonInfo.do');
```

Notice the Ajax Engine is refereced with the identifier *ajaxEngine*. The included *rico.js* file creates an instance of the Ajax Engine with this name making it available to your page scripts to reference.

A request handler can be written in a number of different server languages or mechanisms. Some of these include web services using SOAP or HTTP Rest interfaces or with Java Server Pages (JSP), Jakarta Struts Actions, PHP code, ASP code, Ruby code – and the list goes on.

In our current set of examples we use JSP/Jakarta Struts to handle requests and generate the Ajax response. This means that we have written some Java code that handles the Ajax request for data. For example, in the *Inner Html/Rolodex* demo we have register a request handler named *getPersonInfo* and map it to the URL *getPersonInfo.do*. an URL named 'getPersonInfo.do'. This is a struts action that will return the HTML for the rolodex when passed the last and first name of the person.

Jakarta Struts is an open source framework that provides mechanisms for handling requests, responses, page forwarding and other mechanisms that are required by web applications.

*The **.do** is simple a convention that means this is an Action (an event handler.)*

Try the Request Handler

Try this in your browser:
http://openrico.org/rico/getPersonInfo.do?firstName=Pat&lastName=Barnes

This is what you should see as the response:

```
<ajax-response>
      <response type="element" id="personInfo">
            <div class="person"><span class="personName">Mr. Pat
            Barnes</span><span class="personAddress">1743 1st Avenue
            Boston, Boston 71204-2345</span><span
            class="personOccupation">Executive Vice
            President</span><span class="personPhoneNumber">Phone #:
            (972) 555-0293</span><span class="personMobile">Mobile #:
            (972) 555-0295</span><span class="personNotes">Notes:
            Spouse playes tennis at the country club with
            John.</span></div>
      </response>
</ajax-response>
```

This is a typical Rico ajax-response. It contains a block of well-formed XML code.

Understanding the ajax-response in Rico

Notice several important items about the Ajax response

First the response is wrapped in the tags <ajax-response></ajax-response>. Every Rico Ajax response must have this element as the root of the XML returned.

Second notice the response contained within the ajax-response. The response tags (<response></response>.) wrap the response content An ajax-response in Rico is not limited to a single response. It can contain multiple responses. Each response is marked by the <response></response> set of tags.

> *Every Rico Ajax response must have the <ajax-response> element as its root and at contain at least one <response> element.*

Third, notice the value of the *type* attribute. It is set to *element*. Also notice the value of the *id* attribute is set to *personInfo*. Together this says to take the contents of the response and copy it directly to the inner HTML of the element with *id=personInfo* – in this case a DIV on our page with the *id* set to *personInfo*. Later, we will discuss the other type, *object*.

And last, notice that contained in the XML is a set of well-formed XHTML code. The code in this example is the necessary HTML to create a rolodex card (and coupled with the correct CSS style class) looks like this:

Step 2: Register a Response Handler

So we have a response. And we would like it to create a rolodex card as in our example. How do we get the Ajax response HTML inserted into the contents of a specific DIV on our HTML page?

The key is in the response.

Recall that our response specified the

- *type="element"*
- *id="personInfo"*

The *element* type of response indicates that we will be updating an Html *element* directly whose *id* matches the response *id*. In this case if we specify a DIV (or

whatever HTML element that can contain innerHTML) with this id, then it will be automatically updated with the content of the response.

Here is what our HTML roughly looks like. The DIV with id=*personInfo* will have its content replaced with the HTML for the rolodex card.

```
<div id="rolodexTabContent" class="accordionTabContentBox">
    <table cellspacing="5" cellpadding="5"><tr>
        <td valign="top">
            <select id="listBox" onchange="getPersonInfo(this)">
                [snip]
            </select>
        </td>
        <td>
            <div style="display:inline" id="personInfo">
                Select a name to the left to see the AJAX rolodex entry
                for the selected person.
            </div>
        </td>
    </tr></table>
</div>
```

The div that is in **bold** is the response handler. Two things are required for Rico to recognize this element as the response handler:

1. Give the element an *id* that matches the *id* of the response in the ajax-response
2. Register the element id with Rico
   ajaxEngine.registerAjaxElement('personInfo');

When the response returns the AjaxEngine will copy the response HTML into the innerHTML of the DIV (personInfo). The result in this case would be equivalent to the following HTML code:

```
<div style="display:inline" id="personInfo">
   <div class="person"><span class="personName">Mr. Pat
   Barnes</span><span class="personAddress">1743 1st Avenue Boston,
   Boston 71204-2345</span><span class="personOccupation">Executive
   Vice President</span><span class="personPhoneNumber">Phone #:
   (972) 555-0293</span><span class="personMobile">Mobile #: (972)
   555-0295</span><span class="personNotes">Notes: Spouse playes
   tennis at the country club with John.</span></div>
</div>
```

Step 3: Initiating a Request

This leads us to the third and final step, making the Ajax request. At this stage we have both the request handler and the response handler registered.

What we need now is something to trigger the request. In the HTML code above, notice we provided an event handler on the select. We registered a JavaScript method to be fired on the *onchange* event in the <select> list. Every time a person is selected in the list an Ajax request is initiated and the response gets populated in the DIV on the right.

Here is how we set up our event on the HTML Select element.

```
<select id="listBox" onchange="getPersonInfo(this)">
```

The *getPerson* function makes the Ajax request. Here is the method:

```
function getPersonInfo(selectBox) {
      var nameToLookup = selectBox.value.split(",");
      var firstName = nameToLookup[1].substring(1);
      var lastName  = nameToLookup[0];

      ajaxEngine.sendRequest( 'getPersonInfo',
                              "firstName=" + firstName,
                              "lastName=" + lastName );
}
```

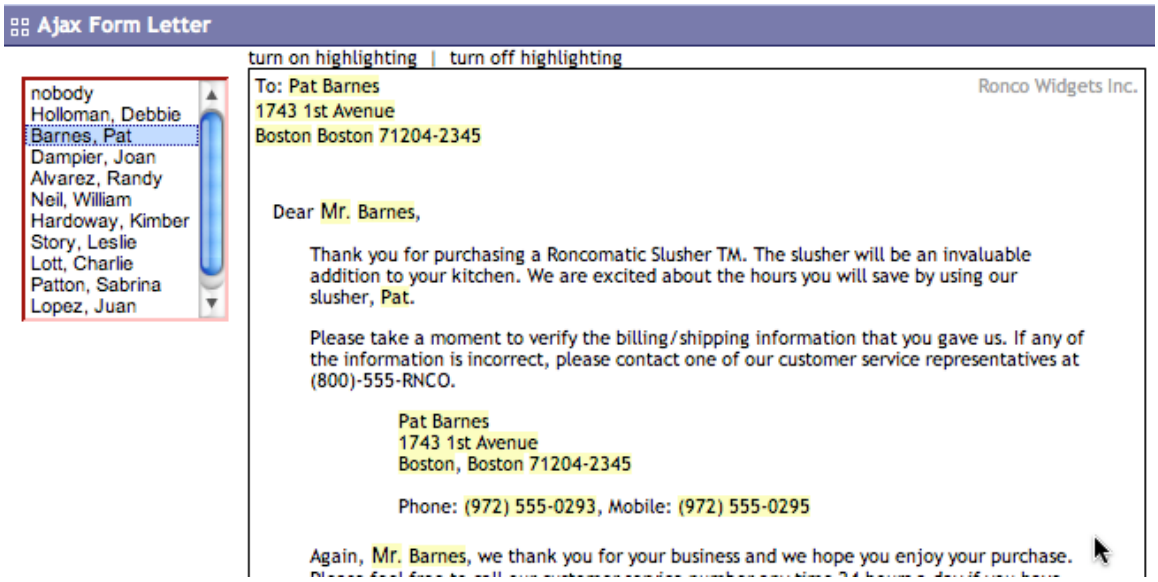
This function's main responsibility is to call the *sendRequest* method on the Ajax Engine passing the request parameters expected by our request handler (remember our Struts Action, getPersonInfo.do?) Notice that it uses the *name* for the request handler not the actual URL.

That's it!! Let's review how to get Ajax running in your page.

1. Register your Ajax request handler
2. Register your Ajax response handler
3. Invoke sendRequest in the appropriate event.

Let's look at the second type of response type—object. The Form Letter Updater demo (http://openrico.org/demos.page?demo=ricoAjaxComplex.html.)

In this demo, selecting a person on the left causes the personal information to be substituted into the form letter (a mail merge.)



Variant: The Object Response Type

Recall that when a response is provided in the ajax-response XML, its type was set to *element*. It meant that the contents were to be targeted into an HTML element with an *id* equal to the response *id*.

There is another response type named *object*. Instead of automatically inserting HTML inside an HTML element, it calls a JavaScript object registered under the name matching the *id* given here. The Ajax engine will call the object's method named *ajaxUpdate*. This method must be present to handle requests.

First, try this URL
http://openrico.org/rico/getPersonInfoXML.do?firstName=Pat&lastName=Barnes

You should see something like this:

```
<ajax-response>
<response type="object" id="formLetterUpdater">
      <person fullName="Pat Barnes" title="Mr." firstName="Pat"
      lastName="Barnes" streetAddress="1743 1st Avenue" city="Boston"
      state="Boston" zipcode="71204-2345" occupation="Executive Vice
      President" phoneNumber="(972) 555-0293" mobileNumber="(972)
      555-0295" personNotes="Spouse playes tennis at the country club
      with John." />
</response>
</ajax-response>
```

Notice in our case the content of the response is XML code describing a single person. Obviously the content is completely application specific. We are retrieving fields of information about a person as XML attributes.

In order to handle this XML format, we must write a JavaScript object that has an *ajaxUpdate* method. When the response is returned, the AjaxEngine will note that it is of object type and locate the JavaScript object that has been registered under a name matching the response *id*.

It is the duty of the *ajaxUpdate* method to parse this XML response and do with it as necessary in the application. Here is the *ajaxUpdate* method signature:

```
ajaxUpdate(ajaxResponse)
```

When a request is made, the XML is processed by the JavaScript object and the appropriate person data is inserted into the form letter.

Here is what the FormLetterUpdater object does with the ajax response:

```
   ajaxUpdate: function(ajaxResponse) {
      this.setPerson(ajaxResponse.childNodes[0]);
   },

   setPerson: function(aPerson) {
      this.lastPersonSelected = aPerson;
      for ( var i = 0 ;i < FormLetterUpdater.attributes.length ;i++ )
      {
         var attr = FormLetterUpdater.attributes[i];
         this.substitute( "span", attr, this.emphasizedHTML(
                         aPerson.getAttribute(attr) ) );
      }
   },

   substitute: function( tagName, tagClass, value ) {
      var elements = document.getElementsByTagAndClassName(
                        tagName, tagClass);
      for ( var i = 0 ; i < elements.length ; i++ )
         elements[i].innerHTML = value;
    },
```

Notice the *ajaxUpdate* expects one child as its node—a person. This is passed to the *setPerson* method. The FormLetterUpdater:setPerson method iterates over the attributes found on the person object and locates spans with the same id and substitutes the attribute values into the innerHTML of each span.

Since you can define your response any way you want (as long as the response is valid XML markup) you can process the response data anyway you see fit.

Here is a snippet of the HTML code that makes up the form letter. The HTML contains a number of spans that have the id matching the response attribute id. Our ajaxUpdate method uses this to perform a simple substitution:

```
<p>
      Please take a moment to verify the billing/shipping
      information that you gave us.
      If any of the information is incorrect, please contact one of
      our customer service
      representatives at (800)-555-RNCO.
</p>
<div style="width:250px;margin-left:50px;margin-bottom:15px;">
<span class="fullName">[fullName]</span><br/>
<span class="streetAddress">[streetAddress]</span><br/>
<span style="text-align:right" class="city">[addresscity]</span>,
<span class="state">[addressState]</span>
<span class="zipcode">[addressZipcode]</span><br/><br/>
```

Registering the Ajax Object
Just like we registered an element (AjaxEngine.registerElement) if you are going to have a request handler that is a JavaScript object you will need to register the object with the Ajax Engine.

```
ajaxEngine.registerAjaxObject('formLetterUpdater',
                              new FormLetterUpdater());
```

Differences
So what is the difference between the two types of responses?

The response type *element* is for simple updating of an HTML element's content with some HTML.

The response type *object* allows for custom or more advanced manipulation of the response data to update the page content.

Multiple Responses

Don't forget that you can also embed multiple responses in the same ajax-response. You can also mix element and object responses in the same ajax-response.

Summary

In summary, the Rico AjaxEngine simplifies updating page content with Ajax. It provides a very straightforward way to change the HTML content of any HTML element. But it also provides an open ended way to respond to more complex data sets returned via a JavaScript object.