

# How to Design a large AJAX Application

## *Introduction*

I'll cover the process I have developed in the course of implementing two AJAX applications as a developer for Duo Consulting <sup>1</sup>in Chicago. This approach has made it easier for me to work with the design team, produce estimates for this type of project and communicate what is involved each step of the way to the project managers for scheduling purposes.

## *Background*

I re-designed WalkJogRun <sup>2</sup>to utilize the Google maps API <sup>3</sup>over a year ago and begun experimenting with AJAX requests to handle tasks that really shouldn't trigger a complete page request. Shortly after that one of our clients, Chicago Park District<sup>4</sup>, wanted to redesign their seasonal registration application to maximize the number of people who could register for their seasonal programs and make it easier for people to use the site. After a detailed discovery phase we proposed an AJAX "browser interface" to make it simpler to drilldown through their online offerings and ultimately to locate a program and register. Inspired by the iPod menu concept, we developed and built an application<sup>5</sup> which served 3000 orders in the first 3 minutes of summer registration.

Since then I worked on a second application which sits on top of the custom CMS<sup>6</sup> we implement for clients to allow them to generate Microsoft Word proposals<sup>7</sup> to send to clients from the data that drives the website without cutting and pasting. This application was built on the prototype AJAX library but uses the same design principles.

## *Why use AJAX?*

AJAX isn't just a new buzzword or a cream cleanser from Procter & Gamble. It's a great leap forward (and a small step back) for usability. Like flash, when it is used correctly and appropriately it can dramatically enhance the user experience by making web applications respond faster and make interactions less jarring than having a page reload. AJAX also offers a performance benefit for your web server since you begin serving smaller chunks of pages instead of complete pages every time. For ColdFusion this also means only the business logic pertaining to the requested chunk needs to be recalculated instead of the logic for a whole page so your application server will get some relief.

---

<sup>1</sup> <http://www.duoconsulting.com/>

<sup>2</sup> <http://www.walkjogrun.net>

<sup>3</sup> <http://www.google.com/apis/maps/>

<sup>4</sup> <http://duoconsulting.com/experience/chicago-park-district-registration/index.cfm>

<sup>5</sup> <http://programs.chicagoparkdistrict.com/programBrowser/>

<sup>6</sup> <http://www.duocms.com>

<sup>7</sup> <http://generator.duodesign.com> : Select the proposal generator once you have logged in with the username [demo@duoconsulting.com](mailto:demo@duoconsulting.com) and password demo

## Design

Work from a mockup you can doodle on. Draw on the mockup to identify discreet sections of the applications and label them. Do panels interact or are they isolated? If there is a relationship between an interaction in one panel and the display of information, draw it with arrows and callouts. This helps to communicate the logical breakdown of the application with other developers. An example marked up mockup for the Chicago Park District program browser application might look like this:

The screenshot shows the Chicago Park District website's program registration interface. The page title is "chicago park district". The navigation menu includes "home", "events", "programs", "in the news", "parks & facilities", "departments", "affiliates", "work@cpd", and "volunteer@cpd". The main content area is titled "Program Registration Summer 2006 Session June 19 - August 27". A yellow banner states "Online registration is now open! Walk in registration opens 04/22/2006 9:00 AM". Below this is a "Find Programs" filter panel with columns for "Categories", "Program Types", "Program Groups", and "Program Names". Callouts 1, 2, 3, and 4 point to various filter options. The "Program Search Results" section shows 1265 matches, with a checkbox for "Only show programs available for online registration (highlighted in yellow)". Two program listings are shown: "Aquatic Exercise - Ages: 60 & Over" and "Aquatic Exercise - Ages: 18 & Over". Callouts 6, 7, 8, and 9 point to details like "Restrictions: None", "Base Fee: \$0.00", "Total Fee: \$0.00", and "Available slots: 1". Callout 10 points to the "Add to Cart" button. The "Wishlist" section includes a "Why create a wishlist?" explanation, "Sign in to your account" (callout 11), and "Don't have an account yet? Create one now >" (callout 12). The "Shopping Cart" section shows "Your cart is empty."

In the example I have numbered the interactions on the mockup for the sake of space but these correspond to the following interactions:

1. selectItem(panel)
2. selectItem(panel)
3. selectItem(panel)
4. selectItem(panel)

5. `getHelp(helptopic)`
6. `viewProgramDetails(activity_code)`
7. `saveToWishlist(activity_code)`
8. `checkAvailability(activity_code)`
9. `addToCart(activity_code,participant)`
10. `cancelRegistration(activity_code)`
11. `loadWishlistScreen(mode)`
12. `loadWishlistScreen(mode)`

Obviously, there are huge gaps in this model. Ideally you would take each colored section of the diagram and model each of the states (screens) and continue labeling interactions.

Identify each interaction (a function) and work out the parameters required to allow the function to work as an isolated item. Functions should be able to perform their task without going beyond global variables and the argument scope in most cases. Create a text file and group the function specifications based on the discreet areas of the application e.g. cart, wishlist, account, help. Add JavaScript comments to describe the interaction and include any pseudo code to describe the steps for a function where there are important validation tests, lookups or steps to be performed.

Are there any panels of the site which have related JavaScript functions that are repeated? Whether an undefined number of repeats or a defined number it is preferred to build a JavaScript object. This reduces code duplication, provides variables local to each instance and generally makes things easier. Prefix functions that act on a specific instance of a panel with the name of the panel e.g. `Cart:function deleteItem()`

Look at the interactions which fetch remote data (make an AJAX call). Think about each in turn and make a decision whether each request must be unique (e.g. get user cart) or whether the information doesn't change at all (e.g. get help panel). Update the comments for each function making a remote call to explain that it is remote and indicate if it is unique or not.

Next, consider the types of remote call you are making. Is the complete result sent back from the remote page as rendered HTML (rendered result) which can be inserted directly into a destination panel or will the result contain JavaScript variables, XML or a custom structured result (functional result) to be processed in a specific way? An example of a rendered result is that a help file chunk can be simply rendered in a given DIV element so it is sent back as HTML; a functional result example would be a remote address lookup function where you send an address and receive back a complex object to be inspected which then updates various fields on a form. Add a comment to each AJAX call to say whether the result is rendered or functional.

Now is probably a good time to walk through your design with another programmer and the designer to make sure you understood the designer's intentions and make sure that your approach isn't missing some obvious efficiencies. An example design document for

the subset of functionality described in the earlier Chicago Park District mockup might look something like this:

**programBrowser.txt**

```
//Search Panels
SearchPanel:selectItem(panel) {
    /* if panel is between 1 and 3 retrieve the
       contents of the next panel (getResults),
       else retrieve the program listing */
}
SearchPanel:getResults(target_panel,filter) {
    /* based on the given filter make a non-unique
       rendered AJAX call to grab the contents of
       the panel and place it in the target_panel */
}

//Help
getHelp(help_topic) {
    /* make non-unique rendered AJAX call to grab the
       help text for help_topic and place it in the
       help div */
}

//Program Search Result
viewProgramDetails(activity_code) {
    /* expand the hidden divs for the panel labeled
       with the activity_code and check availability */
}
saveToWishlist(activity_code) {
    /* make a unique functional AJAX call to return
       a result to indicate
       - is the user logged in
       - is the item already in the cart
       - is the cart full
```

```
        Pass the result to handleSaveToWishlist() */
    }
checkAvailability(activity_code) {
    /* make a unique functional AJAX call to return
    a result to indicate if there is available
    inventory and pass the result to
    handleCheckAvailability() */
}
handleCheckAvailability(result) {
    /* if none available set inventory to zero and
    change the class and text of the inventory box to
    indicate that the item is sold out, else
    update the inventory box to the returned value */
}
addToCart(activity_code,participant) {
    /* skipped for brevity... */
}
cancelRegistration(activity_code) {
    /* hide the more info div for the activity_code */
}

//Wishlist
loadWishlistScreen(mode) {
    /* make a non-unique rendered AJAX call to retrieve
    the content matching the mode passed in and update
    the wishlist div accordingly */
}
handleSaveToWishlist(result) {
    /* based on the result either
        -refresh the wishlist panel to reflect
           the new item
        -update the wishlist panel with an error message
    */
}
```

```
}

```

Again, this is by no means complete but gives you an idea of the pre-work to do at this stage.

### ***Coding the application***

The first thing you will want to do is to take each of the sections of your JavaScript design document and make each section its own .js file to be included in the head of your document. Smaller chunks make it more obvious where to look when you are trying to debug the application:

```
<script src="js/wishlist.js" type="text/javascript"></script>
```

You will need to work through each of those files next and flesh out the JavaScript. If possible, start with the panels least connected and work your way to the most connected panels.

Assuming all went according to plan during the review how exactly do you make the different types of AJAX calls mentioned above? To recap:

Non-unique rendered AJAX call	The result from this call is an HTML fragment and is inserted directly into the contents of a given target div. Non-unique means that the result will be cached at the client minimally and at the server too if you have coded your remote handling page to serve cached results.
Unique rendered AJAX call	The result from this call is an HTML fragment and is inserted directly into the contents of a given target div. The goal is to serve the latest information or information which is volatile so each request is not cached by the browser or the server.
Non-unique functional AJAX call	Results from this call is either a simple result or a complex object in either XML or a custom string to be parsed by the client. Non-unique means that the result will be cached at the client minimally and at the server too if you have coded your remote handling page to serve cached results.
Unique functional AJAX call	Results from this call is either a simple result or a complex object in either XML or a custom string to be parsed by the client. The goal is to serve the latest information or information which is volatile so each request is not cached by the browser or the server.

To begin, you must include the prototype.js libraries if you haven't already done so:

```
<script type="text/javascript" src="js/scriptaculous-js-1.6.0/lib/prototype.js"></script>
```

This currently loads the current version of the library (at this point 1.6.0).

## Rendered results

Rendered results require the user of the prototype function **AJAX.Updater** which looks like this:

```
var reportError = function(t) {
    alert('Error ' + t.status + ' -- ' + t.statusText);
}
function getHelp(helpText) {
    var url = 'index.cfm?fuseaction=cpd.gethelp';
    var params = '&helptype='+helpText;
    var ajax = new Ajax.Updater
        (
            'myDivId',
            url,
            {method: 'get',
             parameters: params,
             onFailure: reportError,
            }
        );
}
```

Breaking this down into more manageable chunks we see three parameters passed to the function. The first is the container to be updated on successful completion of the request; the second is the URL of the remote page to retrieve the rendered HTML and lastly, the options parameter which is further broken down into three elements. Method specifies that the parameters should be submitted using get as opposed to post. Parameters is a query string beginning with an ampersand e.g. “&id=1” and lastly onFailure names a function to execute should there be a transmission error with the AJAX call.

It is as simple as that to serve up a rendered chunk of HTML content! The scriptaculous wiki at <http://wiki.script.aculo.us/scriptaculous/show/Ajax.Updater> has a more in depth outline of the parameters if you need further information.

## Functional results

Functional results require the user of the prototype function **AJAX.Request** which looks like this:

```
var handlerFunc = function(t) {
    //handle result
}
var reportError = function(t) {
    alert('Error ' + t.status + ' -- ' + t.statusText);
}
function checkAvailability(activity_code) {
    var url = 'index.cfm?fuseaction=cpd.checkAvailability';
    var params = '&activity_code='+helpText;
    new Ajax.Request(
        url,
        {parameters: params,
         onSuccess: handlerFunc,
        }
    );
}
```

```
}
    onFailure:errFunc});
}
```

The major difference between this and the AJAX.Updater function is that there is no container passed to be updated and there is an additional option specified “onSuccess” which tells the function to send the results to handlerFunc(). Again, the reference wiki at scriptaculous<sup>8</sup> has a fuller description.

## Uniqueness

There is no support I could find for prototype to be able to make a request unique so I rolled my own. Both of the above scripts can be modified to add a line:

```
url = getUniqueToken(url);
```

This line references a function which, when passed a URL will append a string representing the current date and time down to the millisecond such that a user’s browser is unlikely to be able to submit the same request in that exact millisecond:

```
function getUniqueToken(url) {
    var dt = new Date();
    var dtString = ''+dt.getFullYear()+ dt.getMonth()+ _
        dt.getDate()+dt.getHours()+ dt.getMinutes()+ _
        dt.getMilliseconds();
    dataUrl = url + '&dtm='+dtString;
    return dataUrl;
}
```

This should be enough to get you started with the an AJAX application utilizing Prototype to build a functional application. For an example of a hand rolled AJAX application visit <http://programs.chicagoparkdistrict.com/programBrowser/> and for one built using prototype visit <http://generator.duodesign.com> and select the proposal generator once you have logged in with the username [demo@duoconsulting.com](mailto:demo@duoconsulting.com) and password demo.

---

<sup>8</sup> <http://wiki.script.aculo.us/scriptaculous/show/Ajax.Request>