AJAX Asynchronous JavaScript and XML

By Fletcher Sandbeck

Introduction

AJAX is an acronym for Asynchronous JavaScript and XML. It refers to a technique of building dynamic Web sites by downloading data as XML fragments through a background process written in HTML. However, AJAX is also shorthand for a new generation of Web sites which allow the contents of the page to be manipulated without reloads.

The idea of building Web sites which are more dynamic and more responsive to users has been around a long time. One prior incarnation was dubbed DHTML or dynamic HTML. Another current buzzword for this type of Web site is Web 2.0.

This paper discusses a collection of techniques which are commonly collected under the AJAX moniker. It shows how Lasso works as the back-end for an AJAX solution and how the LJAX (Lasso JavaScript And XML) framework can be used to make programming a dynamic Web site easier. Many of the techniques are derived from the script.aculo.us JavaScript libraries and the Prototype JavaScript framework.

The examples presented in this paper are collected into an AJAX Examples Pack which is included on the Lasso Summit CD. The folder "ExamplesPack" should be dragged into your Web server folder and then accessed through a URL like:

<http://localhost/ExamplesPack/AJAX/index.lasso>

Web Site Interactions

Traditionally a visitor's interaction with a Web site has been defined in terms of either following a series of links or filling out and submitting forms. These techniques have been sufficient to create Web sites for newspapers, shopping carts, message boards, online banking, and many more applications.

Plug-in technologies including Flash and Java make it possible to embed more dynamic applications within Web sites. Flash-based Web sites are often heavily graphics based with animations. Flash allows for interactivity to the point where even video games can be implemented within a Web browser. Java can similarly be used to create interactivity within a downloaded applet whose UI is displayed within a browser.

JavaScript presents a technique of creating dynamic Web sites without using plug-in technologies. JavaScript can be used to auto-fill forms, to check form values before they are submitted, to submit forms immediately after a value has been entered rather than waiting for a submit button to be pressed, and more.

The goal of many AJAX Web sites is to present a user interface which is as rich and responsive as that of any traditional desktop application, but to present that interface entirely in a Web browser. AJAX sites implement drag and drop of page elements, direct editing of text on the page, and pages that update automatically without reloading.

The conceit of AJAX is that this can be accomplished within a Web browser without any plug-ins like Flash and without downloading Java Applets.

Foundation Technologies

AJAX relies on four key Web browser standards: JavaScript, XHTML, XMLHttpRequest, and the DOM.

JavaScript - This ubiquitous client-side scripting language ties all the other technologies together. JavaScript is used to capture the site visitor's mouse clicks and key strokes, fetch data from the server using XMLHttpRequest, parse the resulting XHTML, and rewrite the page's DOM to reflect the new data.

XHTML - XHTML is an elaboration of the HTML standard which makes it follow the same rigorous parsing rules that define XML. XHTML is important to AJAX since it makes it possible for JavaScript to easily parse and manipulate downloaded data without worrying about all the quirks and inconsistencies of traditional HTML.

XMLHttpRequest - This JavaScript function, originally introduced by Microsoft and since embraced by all other major browsers, allows XHTML content to be downloaded asynchronously from whatever Web site is currently being visited in the browser. This secondary data channel is the key to allowing a site to send and receive data without unnecessary browser page reloads.

DOM - The Document Object Model is the Web browser's internal representation of the current page being shown to the site visitor. The current DOM can be manipulated through JavaScript allowing the page to be modified without reloading. DOM manipulations can be as simple as minor text or form element value changes or as major as CSS overhauls or complete page replacements.

A common AJAX procedure is to use JavaScript to call XMLHttpRequest when the user clicks on a link or submits a form. XMLHttpRequest fetches an XHTML fragment and manipulates the DOM of the current page in order to show the new content without reloading page.

Flow Chart

A traditional Web site using links or forms has the following basic flow chart. A Web page is downloaded and displayed to the visitor, they click on a link or submit a form and that action is uploaded to the Web server, a new Web page is generated in return.

```
-> Download HTML
-> Display Web Page
-> Visitor Clicks on Link or Submit Button
-> Upload Data
-> Generate New HTML Page
-> Download HTML
…
```

An AJAX Web site has a tighter loop. By only updating those portions of the Web site which have actually changed, the Web site seems more interactive. When the visitor clicks a link or submits a form the action is uploaded to the Web server, but only a fragment of the page is downloaded and then merged into the visible page in the browser.

```
-> Download HTML
-> Display Web Page
-> Visitor Clicks on Link or Submit Button
-> Asynchronous JavaScript
  -> Upload Data
  -> Generate New XHTML Fragment
  -> Download XHTML Fragment
  -> Update Page DOM Using Fragment
-> Visitor Clicks on Link or Submit Button
…
```

LJAX

A set of tools have been created in Lasso and JavaScript which work together to make AJAX techniques easy. These tools are collectively called LJAX and consist of the following. Full documentation of these tools is included at the end of this paper.

- Lasso.IncludeTarget(target,options) - This JavaScript function encapsulates the process of sending an action to the Web server, downloading an XHTML fragment, and merging that fragment with the current page's DOM.

- [LJAX_Target] … [/LJAX_Target] - This Lasso tag is used to mark different portions of a site so they are served for LJAX requests, for normal requests, or only for certain LJAX targets.

- LJAX.Lasso - This page is created by the site author and is responsible for serving appropriate XHTML fragments based on the what targets and other parameters are passed to it.

In order to use these tools a Lasso site must validate as XHTML. The site can first be written using traditional forms and links. The elements of the site that need to be dynamic are identified. The [LJAX_Target] … [/LJAX_Target] tags are used to block out portions of the page which are LJAX enabled. The LJAX.Lasso page is created to serve XHTML fragments. Finally, the links and forms which should trigger dynamic updates are modified to use Lasso.IncludeTarget().

AJAX Example

The AJAX example included with this paper shows how these tools can be used to create a dynamic Web site. The example is served as a "one file" solution where the "index.lasso" page handles all requests and uses different include files to generate page contents. The "index.lasso" page includes the HTML template wrapper. The "template.lasso" file from the examples is responsible for generating the page contents by checking the "code" action parameter and serving the appropriate file from the "examples" folder.

```
index.lasso
  examples/template.lasso
    examples/dragdrop.lasso
    examples/dynamic.lasso
    examples/reveal.lasso
    …
```

The "ljax.lasso" page from the example functions similarly to the "index.lasso" page. It also includes the "template.lasso" file to serve page contents. However, this contents is wrapped in <ljax> … </ljax> tags and served as an XHTML fragment rather than as an HTML page for

the Web browser. The "ljax.lasso" file also intercepts several targets and serves specially crafted XHTML fragments.

Within the "template.lasso" file the [LJAX_Target] … [/LJAX_Target] tag is used to signify that the HTML s which define the template should only be served if the current target is "page_frame" or if there is no target. This means that these template elements will not be served for other targets.

```
[ljax_target: 'page_frame', -notarget]
  …
[/ljax_target]
```

Several of the individual examples also use [LJAX_Target] … [/LJAX_Target]. The "dragdrop.lasso" file has a section which is served if the target is "nu_content" or "page_frame" or if there is no target.

```
[ljax_target: (array: 'nu_content', 'page_frame'), -notarget]
  …
[/ljax_target]
```

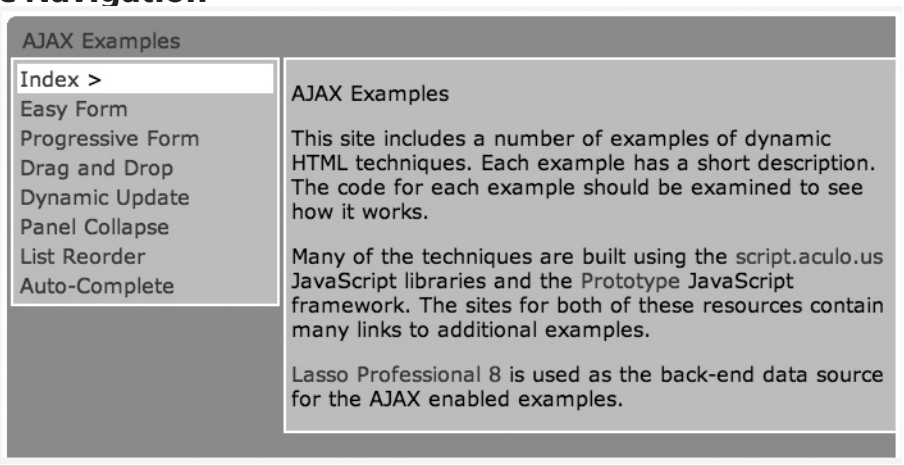
These [LJAX_Target] … [/LJAX_Target] tags create a situation where if the target is "nu_content" then only a small portion of the "dragdrop.lasso" file is served. If the target is "page_frame" then the complete template and page contents is served. If there is no target then the complete HTML page is served. Loading these URLs in your browser lets you see how the contents changes for the different -Target values.

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=page_frame>

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=nu_content>

Each of the examples demonstrates a different LJAX principle or script.aculo.us technique. In order to understand each example you should first try the user interface, then look at the appropriate page from the "examples" folder and any associated JavaScript functions in the "ajax.js" file. Loading the "ljax.lasso" file directly can also help to understand how XHTML fragments are being used to update the page.

Example Navigation



The menu on the left side of the AJAX example is itself a dynamic element. Each time a link in the menu is selected the page content is refreshed dynamically rather than having the entire page reload.

```
<a href="index.lasso?page=dragdrop" onclick="Lasso.includeTarget('page_frame', {args:
this, afterFunc: nu_decorate}); return false;">Drag and Drop</a>
```

The links in the menu each have an href that would reload the page normally. If the browser doesn't support JavaScript (or something goes wrong while executing the JavaScript handler) then the page will simply reload with the new contents like any traditional Lasso Web site.

The onclick handler allows a JavaScript function to be called when the user clicks on the link. The Lasso.includeTarget() function is called with a target of "page_frame". It uses the parameters from the current anchor tag "this" as its args option. And, it asks that nu_decorate() function be called after the page is refreshed dynamically. Finally, false is returned to prevent the browser from handling the click (there is no need since we handled it dynamically).

When the link is clicked by a site visitor the XMLHttpRequest method is used to fetch the following URL asynchronously (in the background).

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=page_frame>

The contents is an XHTML fragment which includes new contents for the HTML div tag with an ID of "page_frame". The Lasso.includeTarget() tag swaps in the new contents and the user sees the new page contents without the entire page refreshing.

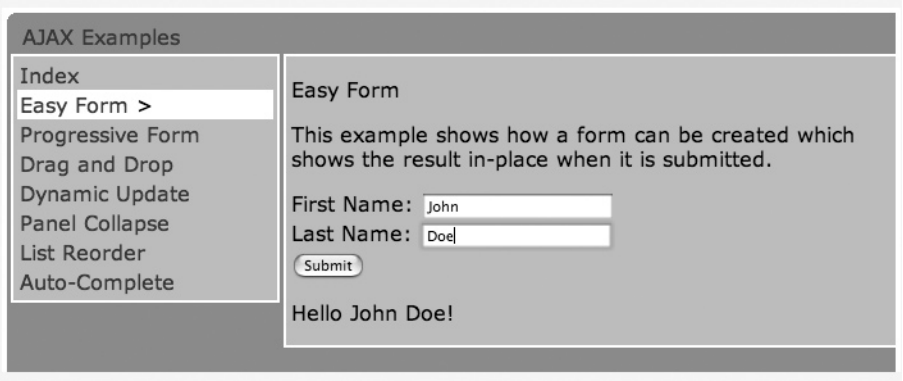
```
<ljax>
  <div class="group_list" id="page_frame">
    …
  </div>
</ljax>
```

Finally, the nu_decorate() function is called to initialize some elements that the drag and drop example needs (see below for details).

It is interesting to turn JavaScript off and try the site. This returns the site to its non-dynamic behavior where each page reloads. The performance of the site is similar in both cases (since the

site is quite simple), but with the AJAX version the URL of the site never changes and the browser user interface doesn't "flash" as much.

Simple Form



The simple form demonstrates how JavaScript can be used to intercept a form submission and change it into an LJAX dynamic page refresh instead. The results of the form are shown below the form dynamically without refreshing the page. Submitting the form multiple times simply updates the results.

The form from the page is shown below. The form is typical except for the onsubmit handler in the <form> tag and the submit <input> tag. The handlers call the JavaScript function Lasso.includeTarget() with a target of form_display. The args for the handler in the <form> tag references "this" to pass the values of all the form elements to the called page. The args for the handler in the <input> tag references "this.form" to accomplish the same thing. Each handler returns false in order to prevent the browser from submitting the form usuing the usual method.

```
<form action="index.lasso" method="post"
    onsubmit="Lasso.includeTarget('form_display',{args:this}); return false;">
  <input type="hidden" name="page" value="easyform" />
  <p>First Name:
    <input type="text" name="form_first" value="[var: 'form_first']" />
  <br />Last Name:
    <input type="text" name="form_last" value="[var: 'form_last']" />
  <br /><input type="submit" name="form_action" value="Submit"
    onsubmit="Lasso.includeTarget('form_display',{args:this.form}); return false;" /></
p>
</form>
```

The Lasso.includeTarget() tag calls the solution's ljax.lasso file with a target of form_display. The [LJAX_Target] … [/LJAX_Target] tags ensure that only the portion of the page which has changed is served. The LassoScript at the top always runs since it is not contained in [LJAX_ Target] … [/LJAX_Target] tags. The introduction text and form are only served if the target is page_from or if there is no target. That is if the page is being loaded by the user clicking on a menu option or by visiting the URL of the page directly.

```
<?LassoScript
  var: 'form_first' = (action_param: 'form_first');
  var: 'form_last' = (action_param: 'form_last');
?>
[ljax_target: (array: 'page_frame'), -notarget]
  <p>Easy Form</p>
  …
  <form action="index.lasso" method="post"
```

```
    onsubmit="Lasso.includeTarget('form_display',{args:this}); return false;">
    …
  </form>
[/ljax_target]
```

The results of the form are wrapped in [LJAX_Target] … [/LJAX_Target] tags which display the results if the target is page_frame or form_display or if there is no target. In particular, when the target is form_display this portion of the page is displayed, but the top of the page is not. The contains results if either the first or last name have a value or is empty if both the first name and last name are empty. The first time the page is loaded this is served empty, but on subsequent loads the contains the "Hello" message. The xmlns attribute of the is required only if the is being served as part of an XHTML fragment.

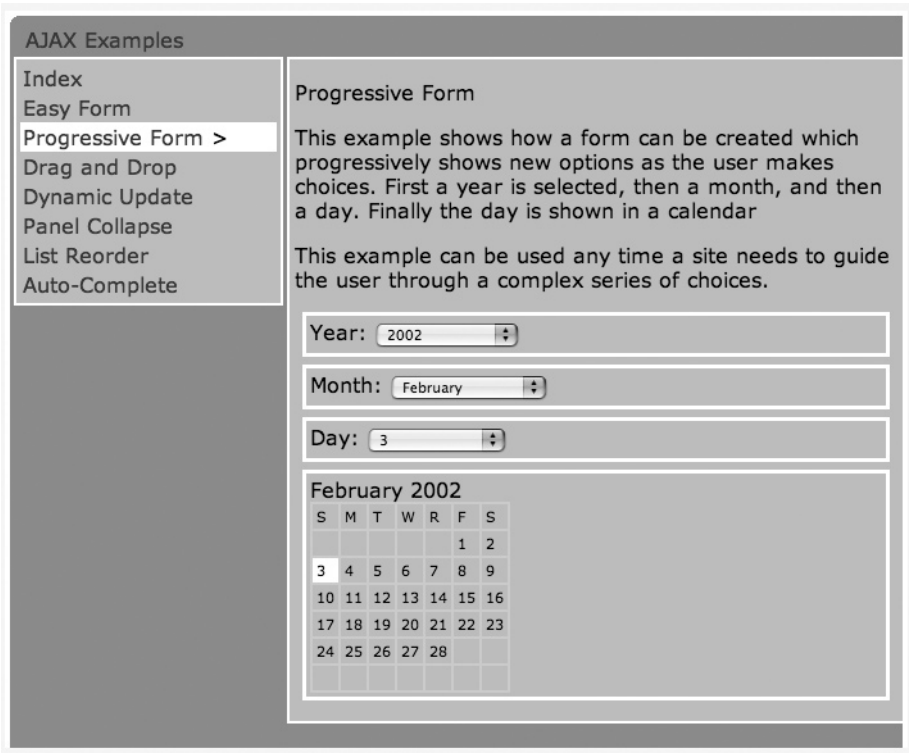
```
[ljax_target: (array: 'page_frame', 'form_display'), -notarget]
  [if: $form_first != '' || $form_last != '']
    <div id="form_display"[if: (var: 'ljax') == true]
        xmlns="http://www.w3.org/1999/xhtml"[/if]>
      <p>Hello [var: 'form_first'] [var: 'form_last']!</p>
    </div>
  [else]
    <div id="form_display"[if: (var: 'ljax') == true]
        xmlns="http://www.w3.org/1999/xhtml"[/if]></div>
  [/if]
[/ljax_target]
```

If the form is submitted with the values "John" and "Doe" then the ljax.lasso page ends up serving code like that shown below. The Lasso.includeTarget() tag is responsible for finding an element with the same ID of form_display and replacing its contents with the contents of the from this fragment.

```
<ljax>
  <div id="form_display" xmlns="http://www.w3.org/1999/xhtml">
    <p>Hello John Doe!</p>
  </div>
</ljax>
```

This example shows the most basic form of LJAX, a form is submitted and a portion of the page is dynamically refreshed. This example can be used as the basis for a wide range of solutions which need to provide user feedback, but don't need the entire page to be refreshed each time the user submits additional data. For example, Lasso could actually be adding records to a database each time the form is submitted and the feedback could be a confirmation that the record was added.

Progressive Form



The progressive form demonstrates how a guided form can be created which shows additional options as the visitor makes choices. The visitor is asked to choose a year, then a month, and finally a day. A calendar of the selected month is shown below with the selected day highlighted. As the visitor progresses only the parts of the page which need to be updated are.

This example uses the same basic idea as the easy form shown above. When the year is chosen an onchange handler calls Lasso.includeTarget() to refresh the dynamic portion of the page. The dynamic portion encompasses the month, day, and calendar displays. The afterFunc option is used to perform a task after the dynamic contents of the page has been updated. In this case the with ID cal_month is made visible by manipulating its CSS display attribute.

```
<select name="cal_year"
    onchange="Lasso.includeTarget('cal_display',{args:this.form,
    afterFunc:function(request){
      document.getElementById('cal_month').style.display = 'block';
    }});">
  <option value=""[if: $cal_year == 0] selected="selected"[/if]>
    Select a Year...
  </option>
  [loop: -from=2000, -to=2010]
    <option value="[loop_count]"
        [if: $cal_year == loop_count] selected="selected"[/if]>
      [loop_count]
    </option>
  [/loop]
</select>
```

The for cal_month has its CSS display attribute initially set to none. This prevents the from being shown on the page. After a year is selected this attribute is set to block (the default for s) in order to have the be visible on the page again.

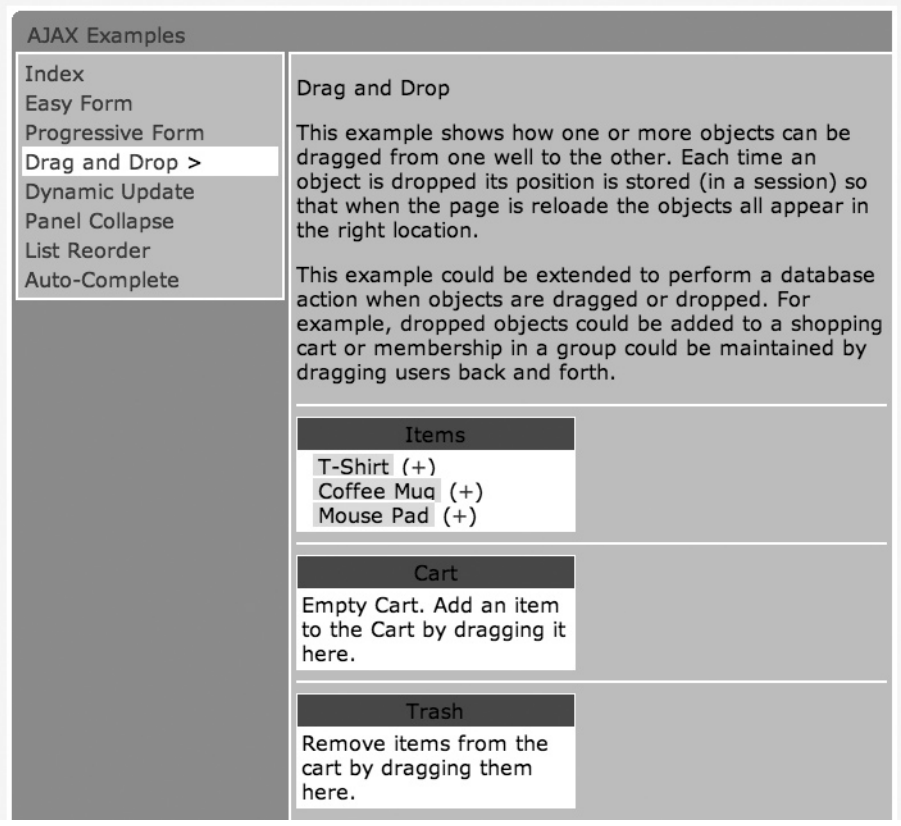
```
<div id="cal_month" style="[if: $cal_year == ‘’]display: none; [/if]border-color:
white;“> … </div>
```

After a month is chosen the day is shown using a similar technique and after a day is chosen the calendar itself is displayed. There is no provision in this example for rolling the inputs back up so the user can de-select a day and de-select a month. Instead, the day and calendar are simply hidden if the month is set to an invalid value.

This technique can be used on sites which need to present the user with a series of questions or hierarchical choices. As the user makes choices the remaining elements of the page are refreshed giving either the next question or the next level of choices.

Note - The browser will not interpret the inline CSS style attributes of dynamically loaded elements in all browser. When this is refreshed it will not have a display attribute of none so it should be shown on the page, but most browsers will remember that the was hidden and won’t make it visible again. Instead, it is necessary to use afterFunc to change the display attribute of the explicitly.

Drag and Drop



One of the most impressive examples from the script.aculo.us library is the drag and drop functionality. It is possible to make any HTML div a draggable object and any other div a drop target. Some Web sites are now using this technique for shopping carts so that users can simply drag items into their cart rather than having to click on links.

The drag and drop example starts as a simple shopping cart interface consisting of two s. The first has a list of items and each item has a + button which calls a URL with an action and item to add an item to the cart. The second div lists items in the cart and has +/- buttons for changing the quantity of items in the cart and an x button for removing items from the cart.

index.lasso?page=dragdrop&action=add&item=nu_one

In order to make elements draggable the function nu_decorate() from the "ajax.js" file is called at the bottom of the page (this function must be called after the s that it decorates have already been defined). This function calls several functions from the script.aculo.us library to create draggable items and drop targets.

Draggables are created by simply referencing their ID in the Draggables() creator function. Each item from the store is marked as draggable and the nu_cart_draggables() function marks each item which is contained in the cart as draggable as well.

```
new Draggable('nu_item_one', {revert:true});
new Draggable('nu_item_two', {revert:true});
new Draggable('nu_item_three', {revert:true});
function nu_cart_draggables()
  {
    if (document.getElementById('nu_cart_one'))
      new Draggable('nu_cart_one', {revert:true});
    if (document.getElementById('nu_cart_two'))
      new Draggable('nu_cart_two', {revert:true});
    if (document.getElementById('nu_cart_three'))
      new Draggable('nu_cart_three', {revert:true});
  }
```

Droppables require that the ID of the drop target be identified as well as the class of draggable item that the drop target should accept and the function that should be called after each item is dropped on the cart. The function in this case is Lasso.includeTarget() with the target of "nu_content", args generated by the + link for the item, and the function nu_cart_draggables() called after the dynamic refresh.

```
  Droppables.add('nu_cart',
    {
      accept:'item',
      onDrop:function(element){
        Lasso.includeTarget('nu_content',
          {args: document.getElementById(element.id + '_add'),
          afterFunc: nu_cart_draggables});
      }
    });
```

A similar call is used to make the trash a drop target for draggable items in the class "cart" which uses args generated bythe x link for the cart item.

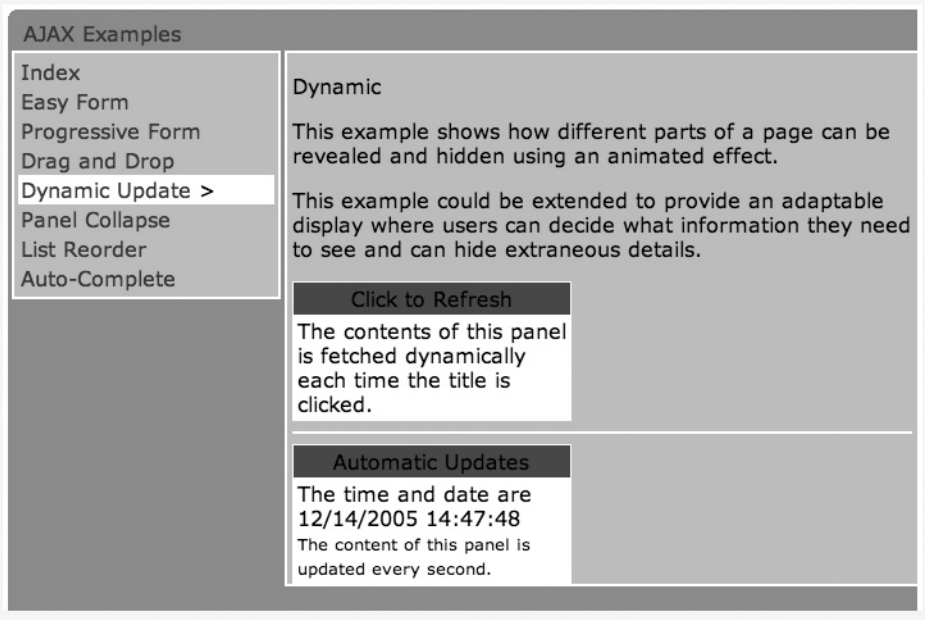
```
  Droppables.add('nu_trash',
    {
      accept:'cart',
      onDrop:function(element) {
        Lasso.includeTarget('nu_content',
          {args: document.getElementById(element.id + '_rem'),
          afterFunc: nu_cart_draggables});
      }
    });
```

Amazingly, that is all the JavaScript required. Now, dragging an item from the “Items” div to the “Cart” div works the same as clicking the + link next to an item. If an item is already in the cart then its quantity is incremented. Items which are dragged to the trash are simply removed from the cart.

The cart can be easily customized by changing the look of the three div’s. The items can be enhanced with product pictures. The cart could show the quanitity of items visually. The trash can be made to look like an actual trash can.

For backward compatibility the +/-/x buttons can be used to perform the same basic operations in a browser that does not support JavaScript.

Dynamic Update



This example shows two methods of dynamically refreshing the contents on a page. Each example shows the current date/time within a small panel. The upper panel is refreshed when the title is clicked. The lower panel is refreshed automatically every second.

The upper panel has the following basic structure. An onclick handler is added to the title div so that it functions like a clickable link. Lasso.includeTarget() is used to refresh the lower div. Note that the “args” option is generated simply as target=theta_content.

```
<div style="background: blue;">
  <div id="theta_title" onclick="Lasso.includeTarget('theta_content',
    {args: 'target=theta_target'}); return false;">Click to Refresh</div>
  <div id="theta_content" style="background: white;">
    The contents of this panel is fetched dynamically
    each time the title is clicked.
  </div>
</div>
```

Within the “ljax.lasso” file this action parameter “target” is checked for the value “theta_target” and a special XHTML fragment is created with the following content. This demonstrates how the

"ljax.lasso" file can quickly serve just the content required for a dynamic update without loading the rest of the site.

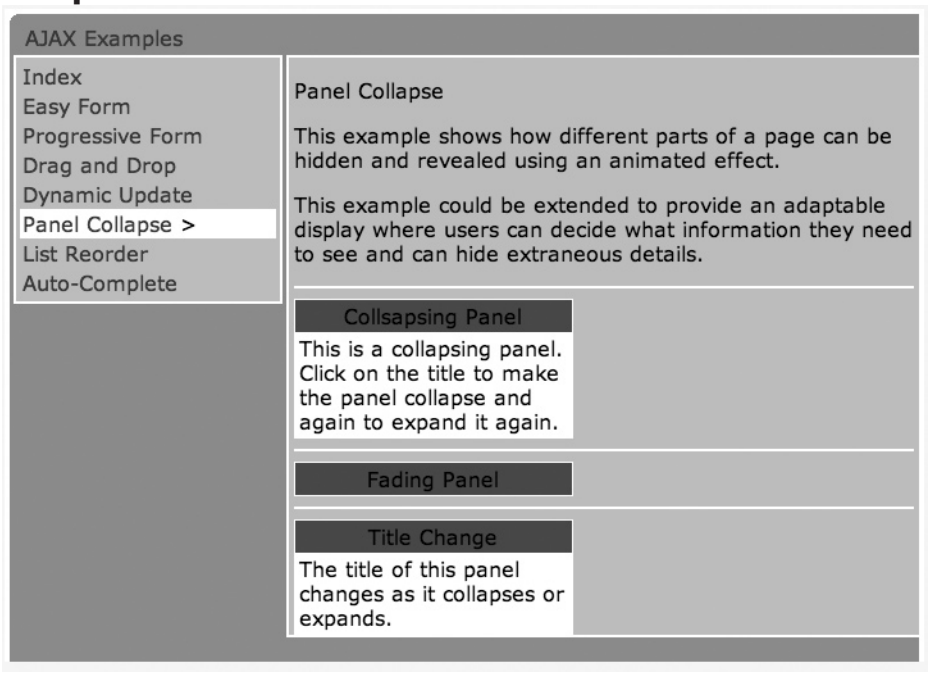
```
'<div id="theta_content" xmlns="http://www.w3.org/1999/xhtml">';
  'The time and date are';
  '<br />' + date->(format: '%D %T');
'</div>';
```

The lower target uses a JavaScript setTimeout() function to schedule an update to run every second. The function eta_refresh() checks if an element with ID "eta_content" is contained on the page. If it is the contents is updated using Lasso.inclueTarget() and the function is rescheduled to run a second later (1000 milliseconds). Otherwise, the function simply exits.

```
function eta_refresh()
{
  if (document.getElementById('eta_content'))
  {
    Lasso.includeTarget('eta_content', {args: 'target=eta_target'});
    window.setTimeout("eta_refresh()",1000);
  }
}
```

These techniques could be used to create a section of a site which is updated each time it is clicked to provide a random "fortune", to provide the answer to a question, or to allow the user to check on the progress of a background activity. The timed version could be used to poll a site for a dynamic status message periodically or to display the ongoing progress of a background activity.

Panel Collapse



This example is a script.aculo.us technique. It shows how to create panels on the page which can be opened and closed by clicking on the title. The panels are animated as they open or close providing good visual feedback for the site visitor.

The first panel appears to roll up into the title of the panel. The panel is defined as follows. The onclick hander in the title calls a function beta_blind() which is defined in the "ajax.js" script file.

```
<div style="background: blue">
  <div onclick="return beta_blind();">Collsapsing Panel</div>
  <div id="beta_content" style="background: white">
    This is a collapsing panel. Click on the title to make the panel collapse
    and again to expand it again.
  </div>
</div>
```

The beta_blind() function is defined as follows. It finds the div with ID "beta_content" and then uses a pre-built script.aculo.us effect to hide the div using an animated effect. Once the div is hidden its height is "0px" and calling the function again will show the div again.

```
function beta_blind()
{
  var beta_content = document.getElementById('beta_content');
  if (beta_content.style.height != "0px")
  {
    new Effect.BlindUp(beta_content);
  }
  else
  {
    new Effect.BlindDown(beta_content);
  }
  return false;
}
```

The second panel both rolls up and fades to blue. This is accomplished in the gamma_blind() function by combining two pre-built script.aculo.us effects.

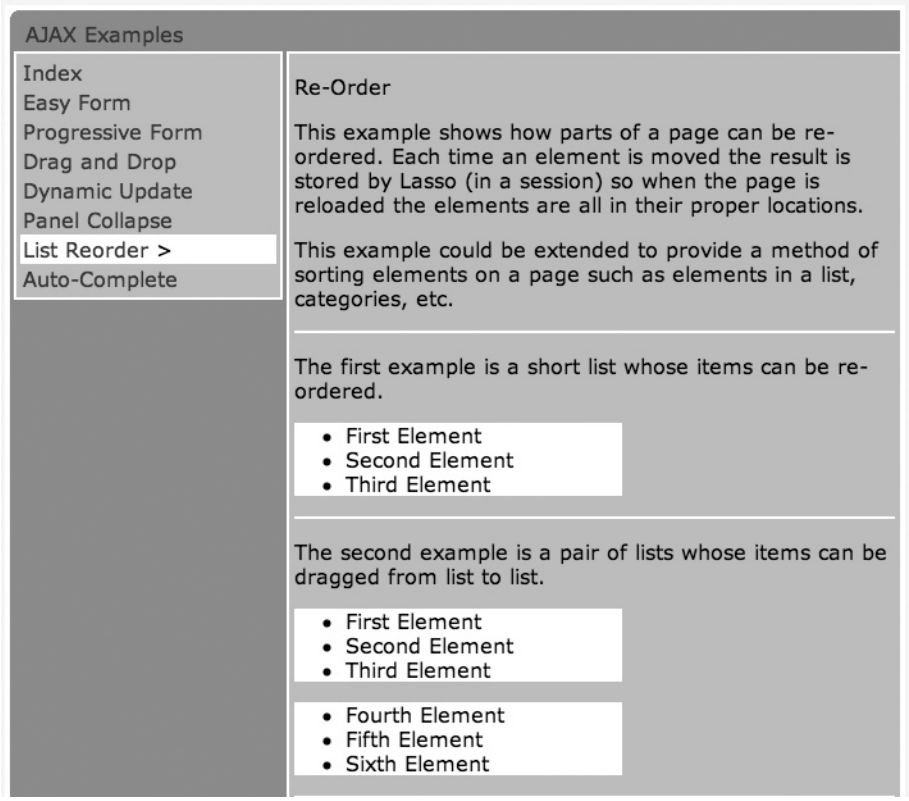
```
new Effect.BlindUp(gamma_content);
new Effect.Fade(gamma_content);
```

The third panel also changes the title of the panel when it is expanded or contracted. This is accomplished in the delta_blind() function by changing the innerHTML property of the "delta_title" div.

```
var delta_title = document.getElementById('delta_title');
if (delta_title)
  delta_title.innerHTML = 'Collapsed';
```

These effects can be used to create a page which has revealable content. The title of each panel could be a summary and the details can be shown only if the panel is expanded.

List Reorder



This example is a script.aculo.us technique. It shows how to create re-orderable lists. The site visitor can re-order the list in the top panel and can drag items from one list to the other in the next two panels. The third part is a sentence which is re-orderable.

The panels are all made re-orderable by the function list_decorate() in the "ajax.js" page. The top panel is made sortable by the following code. The containment parameter specifies that only elements from within "rho_list" can be dragged within the list. The onChange handler logs the new value of the list (and could be used to initiate a database operation, etc.). The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('rho_list',
 {containment:['rho_list'],
   onChange:function(element){
     Lasso.includeTarget('', {args: 'target=none&log=rho:' +
       list_order('rho_list')});}});
```

The next two panels are made sortable by the following code. The containment parameter her specifies both lists. This allows items to be dragged from one list to the other. The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('sigma_list',
  {
    containment:['sigma_list','tau_list'],
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=sigma:' +
        list_order('sigma_list')});
    }
  });
Sortable.create('tau_list',
```

```
  {
    containment:['sigma_list','tau_list'],
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=tau:' +
        list_order('tau_list')});
    }
  });
```

The third part contains a sentence whose words are re-orderable. The new sentence is logged each time it is changed. The or-orderable element in this case are span tags. The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('upsilon_list',
  {
    containment:['upsilon_list'],
    tag:'span',
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=' +
document.getElementById('upsilon_list').innerHTML.replace(/<\/?span.?>/gi,'')});
    }
  });
```

This can be used to have the user prioritize elements on a site or to move users between groups. The Lasso.includeTarget() call could perform a database or session operation to store the new value of the list rather than simply logging the new order.

Auto-Complete

This example is a script.aculo.us technique. It shows how to create an auto-completing text input. The text input is named "zeta_input" and is immediately followed by a "zeta_complete" div which is initially empty.

```
<input type="text" id="zeta_input" name="test" value="" />
<div class="complete" id="zeta_complete"></div>
```

Auto-complete is enabled by calling the zeta_decorate() function in the "ajax.js" file. This function uses the script.aculo.us Autocompleter object with a list of explicit terms to be used in the auto-complete div when it is shown.

```
new Autocompleter.Local('zeta_input', 'zeta_complete',
  ['one','two','three','four','five','six','seven','eight','nine','ten'],
  {});
```

This example could be extended with a list of words generated from site-specific data. The script.aculo.us library also offers several different Autocompleter classes which can be used in conjunction with live data fetched through AJAX methods.

Details

This section contains more detailed documentation of some of the functions used by the AJAX examples.

Lasso.includeTarget(target,options)

The function requires one parameter which is a target (or array of targets) to fetch. Additional options can be specified in a second parameter (described below). The function uses XMLHttpRequest to call a file LJAX.lasso on the current site with a series of -Target=target parameters.

The LJAX.Lasso file is expected to return a valid XHTML fragment surrounded by <ljax> … </ljax> tags. This fragment is parsed and any sub-elements which have an ID are used as replacements for elements with the same ID from the current Web page.

The most common option is "args" which allows any anchor tag or form to be specified. The URL or form parameters of the specified element are passed to the LJAX.Lasso file. Alternately, a string of parameters can be specified. This option allows Lasso.IncludeTarget to essentially simulate a link or form submit.

The option "afterFunc" allows a JavaScript function to be named which will be called immediately after any matching XHTML elements have been replaced. The option "argsoverride" allows specific values to be sent in place of the actual parameters from the link or form specified as "args".

Finally, the option "func" allows a custom function to be used in place of the default behavior of replacing elements with matching IDs. A second optional "param" is passed to the function when it is called.

The Lasso.IncludeTarget() function is typically called in an onclick handler for an anchor tag or in an onsubmit handler for a form tag. In either case the anchor or form can be passed as "this" for the "args" option. The function should be followed by "return: false;" in order to avoid the natural behavior of the tags (reloading the page).

```
<a href= [response_filepath]?name=value"
    onclick="Lasso.IncludeTarget('target',{args: this}); return false;">
  …
</a>
<form action="[response_filepath]
    onsubmit="Lasso.IncludeTarget('target',{args: this}); return false;">
  <input type="hidden" name="name" value="value" />
</form>
```

[LJAX_Target] … [/LJAX_Target]

These tags mark how an area of a Web page should be served. Areas can be marked to serve when no LJAX target has been specified, when any LJAX target has been specified, or when a specific LJAX target has been specified.

[LJAX_Target: 'target'] … [/LJAX_Target] will serve the contents only when the specified target is being asked for through an LJAX request. The contents will not be served for normal HTML

requests. An array of targets can also be specified as in [LJAX_Target: (Array: 'target1', 'target2')] … [/LJAX_Target].

[LJAX_Target: -NoTarget] … [/LJAX_Target] will serve the contents only when an LJAX target has not been requested. This is most useful for HTML-only parts of the Web page including the head elements, title, stylesheet, etc. -NoTarget can be combined with a specific target to serve the contents when either that target or no target has been requested.

[LJAX_Target: -AnyTarget] … [/LJAX_Target] will serve the contents when any LJAX target has been requested. This is most useful for LJAX-only parts of the page.

The most common form of the tag is [LJAX_Target: (Array: 'target1','target2'), -NoTarget] … [/LJAX_Target]. This form serves a portion of the page when any of the targets within the array or no target is specified. This marks an area of the page so it will be served as part of the initial HTML Web page and also as part of an LJAX request for an XHTML fragment.

Note that nested [LJAX_Target] … [/LJAX_Target] tags can be specified, but every outer tag must share at least one target with the inner tags. The following example works fine since "target1" is specified both in the outer tag and the inner tag.

```
[LJAX_Target: (Array: 'target1','target2')]
  [LJAX_Target: 'target1']
    …
  [/LJAX_Target]
[/LJAX_Target]
```

However, in the following example the inner contents will never be served at all since a call to "target3" will cause the contents of the outer tag to be skipped and the code to check whether the inner tag should be served will never be seen by Lasso. If "target1" or "target2" is requested then the contents of the inner tag won't be served anyway.

```
[LJAX_Target: (Array: 'target1','target2')]
  [LJAX_Target: 'target3']
    …
  [/LJAX_Target]
[/LJAX_Target]
```

LJAX.Lasso

The LJAX.lasso page must be created for each site which wants to make use of the Lasso.includeTarget() JavaScript function. The minimum implementation of an LJAX.Lasso must ensure that it be served as an XHTML fragment with appropriate XML header and content-type. The outermost XML tag in the file should be a simple <ljax> … </ljax> tag. And, the file should set the variable "ljax" to be true.

```
<?xml version="1.0" encoding="UTF-8"?>
[content_type('text/xml; charset=UTF-8')]
[var('ljax'=true)]
<ljax>
  …
</ljax>
```

The contents of the <ljax> … </ljax> tags should be one or more XHTML fragments tagged with appropriate IDs. The Lasso.includeTarget() function will scan for all IDs in the file and replace like elements in the current HTML page with their contents.

One of the easiest ways to create an LJAX.lasso file (and the method the AJAX example uses for the most part) is to use the same site includes in the LJAX.lasso file as in the actual site, but to mark up the site with [LJAX_Target] … [/LJAX_Target] tags.

However, it is also possible to craft all of the XHTML within the LJAX.lasso file specially. For some applications this may result in a speed boost, but at the cost of maintaining separate code bases which perform much the same tasks.

Browser Support and Backward Compatibility

The LJAX method and script.aculo.us methods (and included Prototype library) have all been tested in the most recent versions of Safari 2 for Mac OS X, FireFox for Mac OS X and Windows, and Internet Explorer 6 for Windows. The methods may be supported in additional standards compliant browsers as well.

Many of the methods are backward compatible with older browsers and there are notes throghout this paper and the provided code which show how to provide the best backward compatibility possible. In particular, the drag and drop and menu navigation examples are fully backward compatible.

If the XMLHttpRequest method isn't supported by a browser then the Lasso.includeTarget() tag will fail. If the browser doesn't support JavaScript (or JavaScript is turned off) then the callback in the link, div, or form will simply never be called and the mouse click will be recorded as a normal Web browser action.

Examples which are not backward compatible include the re-orderable lists, the expanding and collapsing panels, and the text auto-complete. In general, these features should be incorporated into a site in such a way that their loss in older browsers does not cause the site to be completely unusable.

Additional Resources

These are links to the script.aculo.us Framework and Prototype JavaScript Library. Many of the examples in this paper are built using these tools.

```
script.aculo.us Framework - <http://script.aculo.us/>
Prototype JavaScript Library - <http://prototype.conio.net/>
```

These are links to the Web standards for XHTML, CSS, and a JavaScript resource.

```
XHTML - <http://www.w3.org/MarkUp/>
CSS - <http://www.w3.org/Style/CSS/>
JavaScript - <http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/>
```

Finally, these are some useful Web links which you probably already know.

Lasso Professional - <http://www.omnipilot.com/lasso>
Lasso Examples - <http://www.omnipilot.com/addons>
Lasso Reference - <http://reference.omnipilot.com>
Lasso Tips of the Week - <http://www.omnipilot.com/totw>
OmniPilot Software - <http://www.omnipilot.com>