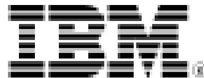


developerWorks.

Ajax overhaul, Part 4: Retrofit existing sites with jQuery and Ajax forms

Streamline a multistep process into a single-screen interface with open-source tools

Skill Level: Intermediate

Brian J. Dillard (bdillard@pathf.com)
VP, Ajax Development
Pathfinder Development

29 Jul 2008

Ajax techniques have changed the face of large, commercial Web applications, but many smaller Web sites don't have the resources to rebuild their entire user interface overnight. New features should justify their costs by solving real-world interface problems and improving user experience. With this series, you've been learning to modernize your UI incrementally using open source, client-side libraries. In this installment, learn to transform a multistep checkout process from a series of sequential forms into a single-screen interface using Ajax techniques. You do so using the principle of progressive enhancement, ensuring that your site remains accessible to all sorts of user-agents.

About this article

This article walks you through the steps of retrofitting a Web 1.0 shopping site with Ajax techniques. You can download the before and after source code for an example application and see both versions in action on the author's Web server. In addition to Ajax techniques and best practices, you learn how Ajax can improve your user experience through the principle of progressive enhancement.

This article assumes you have a solid grasp of HTML and CSS and at least a basic understanding of JavaScript and Ajax programming techniques. The example application is built using only client-side code; the techniques demonstrated can be

© Copyright IBM Corporation 1994, 2008. All rights reserved.

adapted to any server-side application framework. As with all Ajax applications, you must run the example code from a Web server rather than from files on your desktop. Alternatively, you can just follow along in the source code and see the example site in action on the author's Web server.

Looking back at Parts 1-2

Part 1 and Part 2 of this series introduced the example application, Customize Me Now, and began the process of retrofitting it from a Web 1.0 version to an Ajax-powered Web 2.0 version. Using the jQuery JavaScript framework and other open source libraries, you streamlined the user flow of Customize Me Now by replacing popups, off-site links, and navigational side streets with modal dialogs, tooltips, and lightboxes. In Part 3, you made even more improvements. You wrapped long chunks of content in Ajax/DHTML tabs and replaced click-and-wait photo pages with snappy image carousels.

Objectives for Part 4

In this installment, you learn to streamline complex processes by turning multipage forms into Ajax tabs. Your use case is the checkout path of your example shopping site. Without Ajax, multipage forms can seem long, painful, and off-putting to prospective customers. After an Ajax overhaul, even a complex checkout process can seem humane and approachable — as long as you're careful about how you structure the user interface. E-commerce sites aren't the only places that can benefit from these techniques. The same principles apply anywhere users must fill out a series of interrelated forms to complete a multistep process.

To understand the concepts in this installment, refer to Customize Me Now 1.2 (see Download, a slightly revised version of your original, pre-Ajax example site. As you make changes to 1.2, you will create Customize Me Now 2.2, which incorporates all of your changes from the entire series.

E-commerce checkout: The Web 1.0 version

Even customers who love online shopping often loathe checking out. The problems are myriad:

- It's often unclear how many steps are involved in the process.
- It's also unclear how long each step will take.
- Depending on how users answer certain questions, they may get taken on a round-trip detour to unrelated parts of the application. Choosing

© Copyright IBM Corporation 1994, 2008. All rights reserved.

shipping options, applying discount codes, or even logging in can all make a seemingly straightforward process take longer than expected.

- Unless site developers are careful about how they code the secure portions of the process, cryptic security warnings can raise users' hackles.
- Poorly worded error messaging and inconsistent error signposts can make it hard for users to understand when an error has occurred.
- Poorly coded validation routines can result in an endless loop of frustration. The culprits range from credit card numbers that get blanked out and have to be re-entered after a server hit to check boxes that don't stay checked like they should.

E-commerce checkout: The Web 2.0 version

Improving user experience should be one of your top objectives when investing time and development dollars in retrofitting an existing Web site with Ajax. Ajax can't fix all of the things that consumers hate about checking out, but it can help in at least three areas:

- Clarity about the number of steps necessary to complete a process
- Speed when transitioning between steps
- Simplicity and consistency when users must take a detour to log in, enter a discount code, or otherwise deviate from the "happy path"

The example application: Customize Me Now 1.2

If you take a look at Customize Me Now 1.2 in your Web browser, you see that checkout is, as on many sites, relatively confusing. The breadcrumb navigation provided shows checkout as a five-step process:

1. Personal Info
2. Shipping Details
3. Billing Details
4. Order Review
5. Confirmation

However, the reality is more complicated. As a user, before you even dive into Step

© Copyright IBM Corporation 1994, 2008. All rights reserved.

1, you must choose whether to check out as a guest or log in. When you choose the former, you're taken directly to Step 1, where you must enter your name and contact information. When you choose the latter, you must go off to a login screen, enter your username and password, and come back to Step 1 to review your previously entered contact information.

After you get past Step 1, the process is fairly linear. But when you reach Step 4, Order Review, you're presented with another potential detour, this time to the Apply Discount page.

During the entire process, including any detours, you have exactly one tool to help you understand where you are in the process: the aforementioned breadcrumb navigation. The only breadcrumb items that are clickable at any given time are the ones you have previously completed. There's no way to jump forward in the process, only backward. You can see this breadcrumb trail in action in Figure 1:

Figure 1. Customize Me Now 1.2 breadcrumb trail

This breadcrumb navigation is useful to a point, but because it's styled as a pipe-delimited list, many users may not even notice it. It also doesn't accurately reflect the detours users can take. Login is actually a sub-step of Step 1, and discount codes are a sub-step of Step 4. Yet each of these detours requires two server round-trips just to get the users back to where they were. Each server hit lengthens the process and disorients users as the new page loads.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

As with all previous versions of Customize Me Now, the navigation in the header and footer would probably not exist in the real world. It's there only to help you jump quickly to various pages within the example application as you read this article. Additional differences exist between the Customize Me Now 1.2 example code and a real-world e-commerce application. There is no Secure Sockets Layer (SSL) encryption. There's no validation on either side of the Hypertext Transfer Protocol (HTTP) connection. Finally, each step of the process is static. For instance, even if users check the box to use their shipping address as their billing address, the billing address isn't prefilled. A live e-commerce site would enable this functionality with server-side code. For the purposes of this article, the client-side code at least suggests the additional complications that would occur in the real world.

Retrofitting your checkout path: First steps

Now you're ready to begin building Customize Me Now 2.2. By turning your checkout path into a single-screen interface, you cut down on the latency between server round-trips and provide a less disorienting transition between steps. By the time you're done, trips to the login and discount screens will seem like an intuitive part of the process rather than confusing detours.

Plugging in the JavaScript libraries

Your tools for this overhaul include jQuery, the popular Ajax library you used in previous installments, and two familiar plug-ins:

- **jQuery UI Tabs**, used in Part 3, turns an unordered list into a tab interface and fills each tab with content from an existing Document Object Model (DOM) element or an Ajax call. Here, you use it to turn seven separate HTML files into a single-screen interface. When you employed this plug-in last time, you altered the included background image to improve the appearance of its rounded corners. You should use that same modified image this time around.
- **jQuery Form**, used in Part 1 and Part 2, provides methods to submit forms via Ajax and manipulate the results in a variety of ways. Here, you use it to control the flow between the steps of your single-screen checkout process.

All of the heavy lifting for your overhaul occurs on one page: checkout.html. To get this file ready, download the JavaScript and CSS files for jQuery and its plug-ins and reference them in the head of your HTML file. The result should look like Listing 1:

Listing 1. JavaScript library links

```
<!--jquery assets-->
<script type="text/javascript"
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
        src="../js/jquery-1.2.3.min.js"></script>
<script type="text/javascript"
        src="../js/jquery.form.js"></script>

<!--jquery.ui.tabs assets-->
<script type="text/javascript"
        src="../ui.tabs/ui.tabs.pack.js"></script>
<link rel="stylesheet"
        href="../ui.tabs/ui.tabs.css" type="text/css"
        media="print, projection, screen">
```

Creating the HTML fragments

As in Part 3 of this series, you also need to create some HTML-fragment versions of existing HTML files. This step is necessary because Ajax responses usually need to omit headers, footers, and other extraneous elements that are ordinarily served with the page's main content. Full-page and fragment versions of HTML files can usually be served up from the same server-side templating engine. Here, you simulate that effect by copying several files over to new filenames, leaving the original files in place. When you're done, you have the following new files:

- login-fragment.html
- checkout1-fragment.html
- checkout2-fragment.html
- checkout3-fragment.html
- checkout4-fragment.html
- checkout4a-fragment.html

Next, open each of those files and strip out everything but the main form element and its child elements. When you're done, the entire contents of login-fragment.html looks like Listing 2, with each subsequent file following the same pattern:

Listing 2. HTML fragment file format

```
<form method="GET" action="checkout1.html"
        class="checkout" id="lform">

        <h2>Step 1: Personal Info</h2>

        <h3>Login</h3>

        <div>
                <label>
                        Email Address
                        <input type="text" name="email"
                                id="email"/>
                </label>
                <label>
                        Password
                        <input type="password" name="password"
                                id="password"/>
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
                </label>
        </div>

        <div>
                <input class="button" type="submit"
                        name="submit" id="submit"
                        value="submit" />
        </div>

</form>
```

Retrofitting your checkout path: The heavy lifting

Now that you have all the files you need, it's time to begin the real work of this Ajax overhaul. Most of the code changes take place within checkout.html, which, as you've noticed, did not get copied over to a fragment version. That's because the existing version of checkout.html serves as both the hub of your new tab interface and the starting point for the existing interface, which users still see when JavaScript functionality is not available.

Turning separate pages into tabs with jQuery UI Tabs

Within checkout.html, you need to create some `div` elements to hold your tabbed content. Each `div` takes a `class` attribute of `"tabContent"` so you can apply styles to it, while each gets a unique `id` attribute so you can get an object reference to it in your JavaScript code. The first `div` element is wrapped around the existing content of checkout.html. Three additional, empty `div` elements are added below as placeholders for content that will be fetched later via Ajax. When you're done, your HTML code looks like Listing 3:

Listing 3. HTML for tab content wrappers

```
<div id="personalInfo" class="tabContent">

        <h2>Step 1: Personal Info</h2>

        <div class="buttons">
                <a class="button" href="checkout1.html"
                        id="checkoutAsGuest">
                        check out as guest
                </a>
                <a class="button" href="login.html"
                        id="login">log in</a>
        </div>

        <div class="fakeForm">
                <p>[long, boring playback of order details]</p>
        </div>

</div>

<div id="shippingDetails" class="tabContent"></div>
<div id="billingDetails" class="tabContent"></div>
<div id="orderReview" class="tabContent"></div>
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

You may have noticed that you created `div`s for only four of the five steps in your checkout process. Never fear. Step 5, Confirmation, is a special case. You want to show it as one of the five steps so that users maintain a sense of where they are in the process, but Step 5 actually occurs after the checkout process is complete. Therefore, it opens in a new page rather than in your tab interface, so there's no need to create a placeholder for it.

You also need to make sure the container `div`s you just created get styled properly. You add a border and some padding to make them look good underneath your tabs. You also suppress the `h2` elements inside them because the labels on the tabs render them redundant. Luckily, you already created style rules to cover both of these needs in Part 3 of this series. You just need to make sure those rules get added to the bottom of customizemenow.css. The result looks like Listing 4:

Listing 4. CSS to style tab content

```
#CMN .tabContent {
        padding: 14px;
        border: 1px solid #97a5b0;
}
#CMN .tabContent h2{
        display: none;
}
```

Now that you've created and styled the `div` wrappers for your tabbed content, you need to create the tabs themselves. As you may remember from Part 3, jQuery UI Tabs constructs its tab interface from an unordered list. You already have a `ul` element that displays your breadcrumb navigation; it looks like Listing 5:

Listing 5. HTML for breadcrumb navigation

```
<div class="breadcrumb nav">
        <ul>
                <li class="current">1. Personal Info</li>
                <li>2. Shipping Details</li>
                <li>3. Billing Details</li>
                <li>4. Order Review</li>
                <li class="last">5. Confirmation</li>
        </ul>
</div>
```

However, several differences exist between this markup and the HTML structures jQuery UI Tabs expects. Therefore, you should create an alternate `ul` element for your tabs and then use CSS and JavaScript code to toggle the visibility of the two lists. The default mode is to show the breadcrumb navigation and hide the tabs; this ensures that users without JavaScript capabilities see the same Web 1.0 interface they've always seen. You can use JavaScript code to hide the breadcrumb list and show the tab list in its place for the JavaScript-enabled users who see your Ajax checkout interface.

The `ul` element for the tabs looks like Listing 6:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 6. HTML for tabs

```
<ul class="navTabs">
        <li><a id="tab0" href="#personalInfo">
                <span>1. Personal Info</span></a>
        </li>
        <li><a id="tab1" href="#shippingDetails">
                <span>2. Shipping Details</span></a>
        </li>
        <li><a id="tab2" href="#billingDetails">
                <span>3. Billing Details</span>
        </a></li>
        <li><a id="tab3" href="#orderReview">
                <span>4. Order Review</span>
        </a></li>
        <li class="last"><a id="tab4" href="#">
                <span>5. Confirmation</span>
        </a></li>
</ul>
```

Note that the URLs for the link elements inside your tabs correspond with the names of the content `div`s you created earlier. The Confirmation tab, which needs no associated content, needs only a hash symbol for its dummy URL.

The CSS to hide your new `ul` element goes in a `noscript` block in the body of checkout.html. As you may remember, you used a similar strategy for your tabs in Part 3 of this series. Now, as then, you add a couple of additional `noscript` styles to assist users without JavaScript functionality enabled. Basically, you reverse the style rules you created in Listing 4. When you're done with your `noscript` style block, it looks like Listing 7:

Listing 7. Noscript CSS

```
<noscript>
        <style type="text/css">
                /*don't show tabs when JS disabled*/
                #CMN .navTabs {
                        display: none;
                }
                /*without the tab box, disable border+padding*/
                #CMN .tabContent {
                        padding: 0;
                        border: 0;
                }
                /*without the tab labels, stop hiding h2s*/
                #CMN .tabContent h2 {
                        display: block;
                }
        </style>
</noscript>
```

Next comes the fun part. It's time for you to wire everything together with some custom JavaScript code. You bind all of your code to jQuery's custom `document.ready` event, which fires when the DOM becomes available to the browser's JavaScript engine. This ensures that all the DOM elements you need to manipulate are ready and waiting.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

To create this intricate event handler, you simply add a script block to the bottom of the `head` element of checkout.html like the one in Listing 8:

Listing 8. JavaScript code to create tabs

```
<script type="text/javascript">
//when the document is ready
$(document).ready(function() {

        /* hide the breadcrumbs ul and show the tab ul */
        $('div.breadcrumb').hide();
        $('ul.navTabs').show();

        /*turn the newly visible ul into tabs*/
        var tabSet = $('ul.navTabs').eq(0).tabs(
                {
                        /*apply a nice visual effect to tab activation*/
                        fx: { height: 'toggle', opacity: 'toggle' },
                        /*disable all but the first tab by default*/
                        disabled: [1,2,3,4]
                }
        ).bind('select.ui-tabs', function(event, ui) {
                        /*
                        ensure that each time a new tab is activated
                        all subsequent tabs are disabled. This will
                        prevent users from jumping around in the process
                        */
                        var currentTab = parseInt(ui.tab.id.substring(3));
                        var tabSetLength = 5;//a necessary hack
                        for (var i = 0; i < tabSetLength; i++) {
                                if (i > currentTab) {
                                        tabSet.tabs("disable", i);
                                }
                        }
        });

        /*
        bind the checkout and login links to new click
        handlers and cancel their original behavior.
        now, these links will open the fragment versions
        of their original targets inside the first tab.
        */
        $("#checkoutAsGuest,#login").click(function() {
                $('#personalInfo')
                        .load(
                                $(this)
                                .attr("href")
                                .replace(".html", "-fragment.html")
                        )
                ;
                return false;
        });

});
</script>
```

As the comments in Listing 8 make clear, you're using the power of jQuery to transform your original markup into its new tabbed format. You're hiding elements of the page that are no longer needed, revealing elements that are, and telling jQuery UI Tabs to transform the appearance and behavior of those elements. The options bundle that you're passing to the `tabs` method employs the same visual effect you used in Part 3, but an additional parameter allows you to disable all but the first tab.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

This allows you to make your checkout path a one-way process in which users cannot jump ahead until they've completed each step in turn. Later on, you'll write code to enable the disabled tabs one at a time when they're needed.

jQuery UI Tabs also provide some custom event hooks within the life cycle of the tab interface. Like built-in DOM events such as `onclick` and `onmouseover`, custom events can have callback handlers applied to them. The only difference is that the events themselves are defined by jQuery and its plug-ins rather than the browser. You can use one of these custom event handlers to further lock down which tabs are enabled or disabled at any given time.

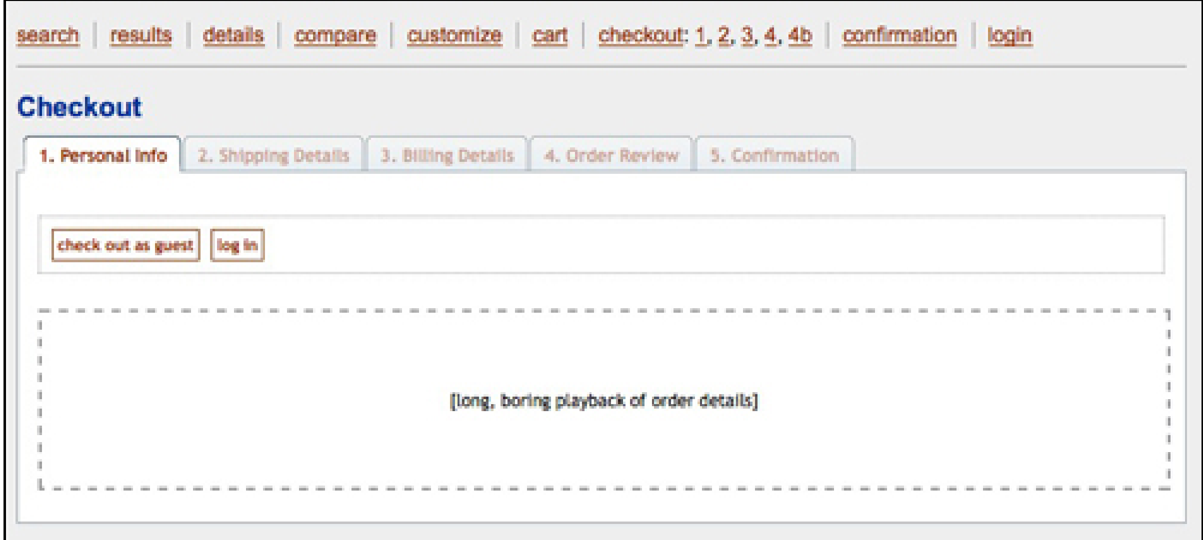
Each time a new tab is activated, an event called `select.ui-tabs` is thrown. (The name for this event has changed in subsequent versions of jQuery UI Tabs, so check the documentation if you encounter problems.) By calling jQuery's `bind` method on your newly created tab set, you can attach a handler to this custom event. This handler cycles through the entire set of tabs and disables each tab that is later in the process than the currently selected one. Unfortunately, the `select.ui-tabs` event handler does not have access to the entire tab set, only the individual tab that has been selected. You therefore have to hard-code your references to the tab collection and the number of tabs in the set.

Now that you've wired the behavior of the other tabs, you need to tell the content in the first tab what to do. This involves altering the behavior of the links labeled **log in** and **check out as guest** so that they load their targets via Ajax inside the current tab. The original links point to complete HTML pages, but you just need the HTML fragments you created earlier. You therefore use jQuery to grab object references to the two links and override their default behavior with `click` event handlers. You grab each link's `href` attribute, alter it to point to a fragment file, and fetch the resulting URL via Ajax. Finally, you render the response inside the Personal Info tab.

Voila! You've now built Step 1 of your five-step checkout path. Visit the checkout page in your browser, and you see a page like Figure 2:

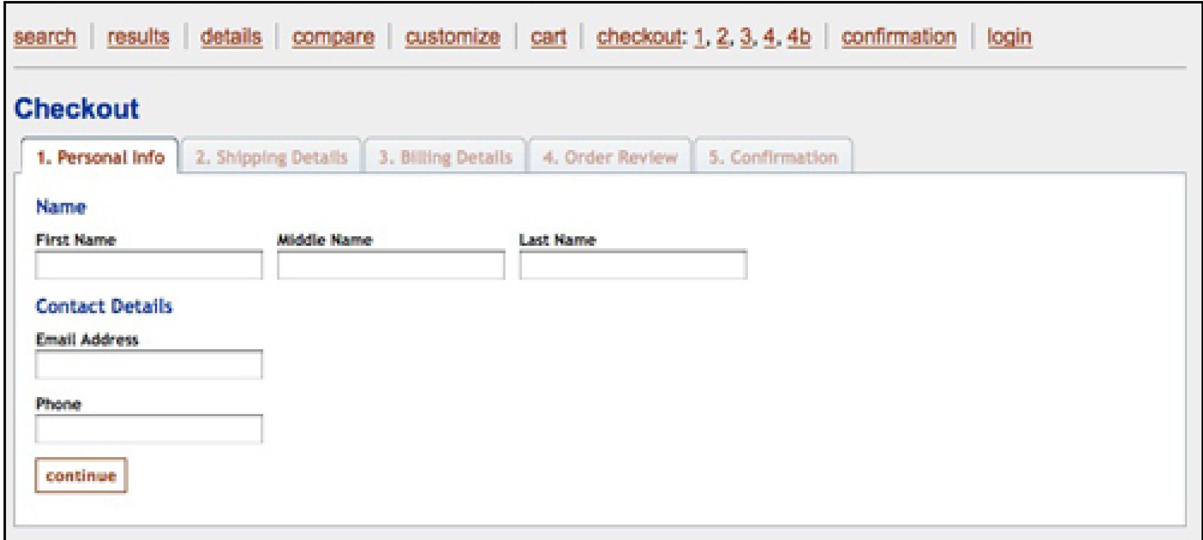
Figure 2. Customize Me Now 2.2 checkout Step 1: Log in or check out as guest

© Copyright IBM Corporation 1994, 2008. All rights reserved.



If you choose the **check out as guest** link, the contents of the Personal Info tab refresh with the actual personal information form, as shown in Figure 3:

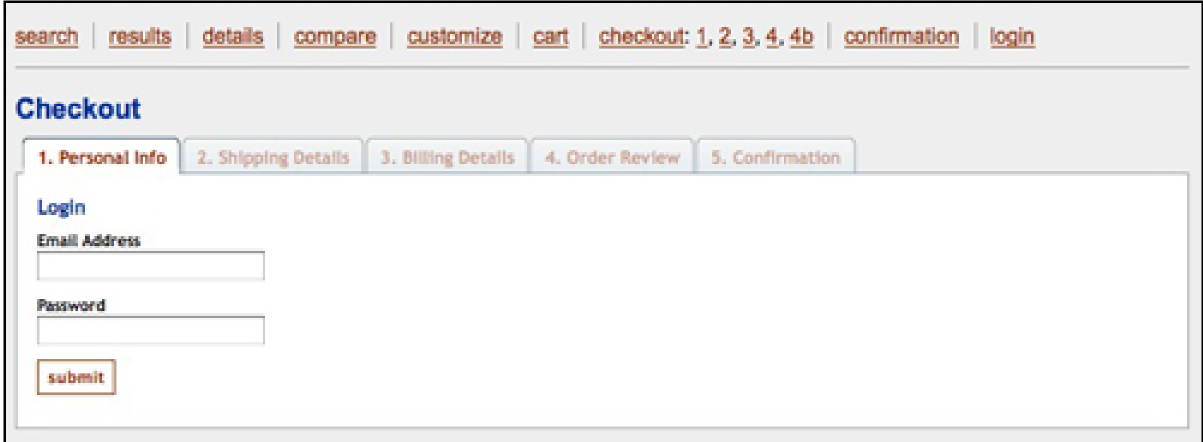
Figure 3. Customize Me Now 2.2 checkout Step 1: Personal Info form



If you click the **log in** link instead, the contents of the Personal Info tab refresh with the login form, as shown in Figure 4:

Figure 4. Customize Me Now 2.2 checkout Step 1: Login form

© Copyright IBM Corporation 1994, 2008. All rights reserved.



If you try to get past either the login or the personal information forms at this point in your overhaul, your tab interface will quickly break. Instead of continuing to load content inside tabs, you get full-page refreshes of the original, non-fragment versions of your pages. That's because you have not yet told your individual HTML fragment files how to function inside your tab infrastructure. To complete this final task, you need your other plug-in: jQuery Form.

Controlling the step-by-step flow with jQuery Form

Redirecting the **log in** and **check out as guest** links into your Ajax tabs was easy. The rest of your checkout pages contain HTML forms, whose default behavior is a little more difficult to override. As you saw in Part 1, however, jQuery Form provides the necessary power. The trick to completing your Ajax overhaul is using this plug-in to transform stodgy forms into nifty Ajax widgets.

To accomplish this, add a script block to the bottom of each HTML fragment file to rewire that fragment's form submission. Because fragments are loaded via Ajax and their scripts are executed within the context of checkout.html, there's no need for the fragments to link to any external script files. They're already included in checkout.html.

First, you tackle the login form. You previously enabled users to reach that form via Ajax. Now you just need to make it submit via Ajax, too, by attaching a handler to the form's `submit` event. Because you're going to load several forms inside the same HTML shell, you must give each form a unique HTML `id` attribute so your scripts can tell them apart. If you look at the HTML of your fragment files, you see this has already been taken care of. The form in login-fragment.html has an `id` of `lform`, the form in checkout1-fragment.html has an `id` of `pform`, and so on.

(To prevent collisions, you also need to give each individual form field a unique ID. Again, this has already been taken care of in the example code.)

Your submit handler for login-fragment.html looks like Listing 9:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 9. JavaScript submit handler for login page

```
<script type="text/javascript">
// bind the form and provide a callback function
$('#lform').submit(function() {

        //submit the form via ajax
        $(this).ajaxSubmit({
                target:     '#personalInfo',
                url:        'checkout1-fragment.html',
                success:    function() {
                        var tabSet = $('ul.navTabs');
                        tabSet.tabs("enable", 0);
                        tabSet.tabs("select", 0);
                }
        });

        //don't actually submit the form normally
        return false;
});
</script>
```

This handler grabs an object reference to the associated form and uses the `ajaxSubmit` method to reroute it via Ajax. By passing an options bundle to `ajaxSubmit`, you're telling it the following:

- Where to submit the form's values (checkout1-fragment.html, the file containing the personal information form)
- Where to load the response (the `div` element for the first tab, which carries an `id` attribute of `personalInfo`)
- What to do when the Ajax call succeeds (enable and select the first tab, identified by its index of 0)

Finally, by returning `false`, you're cancelling the normal, non-Ajax submission of the form.

To see your progress, begin the checkout process anew in your browser. Click the **log in** link, submit the login form, and you should see the personal information form load in the first tab. You now have the ability to get to this form either directly from the start of the checkout process or after a detour to the login page. (In the real world, of course, the personal information form would probably be prefilled for you if you logged in.)

Now, though, you must create a way to get off of your first tab altogether and advance to the second and subsequent tabs. This time, you need to add a form submit handler to checkout1-fragment.html, the file that contains the personal information form. This handler is similar to the login one. But instead of loading the results into the first tab, enabling it, and selecting it, you perform all of those actions on the second tab. Your options bundle therefore points to the `div` with the `id` of `shippingDetails`, while the 0-based index of the corresponding tab is 1. When you're done, your handler looks like Listing 10:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 10. JavaScript submit handler for Personal Info form

```
<script type="text/javascript">
// bind the form and provide a callback function
$('#pform').submit(function() {

        //submit the form via ajax
        $(this).ajaxSubmit({
                target:     '#shippingDetails',
                url:        'checkout2-fragment.html',
                success:    function() {
                        var tabSet = $('ul.navTabs');
                        tabSet.tabs("enable", 1);
                        tabSet.tabs("select", 1);
                }
        });

        //don't actually submit the form normally
        return false;
});
</script>
```

You will continue this pattern for most of your subsequent fragment files, with one exception: The file for Step 4, Order Review (checkout4-fragment.html), doesn't need an Ajax form. As discussed earlier, Step 5 is the Confirmation page, which breaks out of the tab interface and loads as an entirely new page. Therefore, the normal form action is correct for Step 4. However, Order Review does offer a detour to enter a discount code. This fragment, therefore, needs a `click` handler to reload the current tab with the discount form from checkout4b-fragment.html. The results look like Listing 11:

Listing 11. Click handlers for Order Review

```
<script type="text/javascript">
$('#enterDiscount').click(function() {
        /*
        grab the url of the current link and load
        the fragment version it in an existing tab
        */
        $('#orderReview')
                .load(
                        $(this).attr("href")
                                .replace(".html", "-fragment.html")
                        )
                ;
        return false;
});
</script>
```

Reviewing your retrofit

That's a wrap. The changes you've made have transformed Customize Me Now 1.2 into version 2.2.

Following the happy path

© Copyright IBM Corporation 1994, 2008. All rights reserved.

To see your tabbed checkout path in action, visit the site in your browser and navigate through the steps without taking detours to log in or apply a discount. You should see a single-screen interface in which each step of the process is carried out inside its own tab and a window-shade visual effect marks the transitions between steps. When you get to the penultimate step and click **purchase**, you should be taken to the Confirmation page, which looks just as it did in Customize Me Now 1.2.

Following a more complex path

Now run through the checkout path a couple more times and take detours through the login and discount processes. Your linear, tabbed process remains unchanged, except that some steps (1 and 4) are compound steps that include multiple sub-steps. The individual sub-steps each load without visual transition. When you do progress to a subsequent tab, you see the familiar window-shade effect.

Now, get really creative. Complete the process through Step 4, and then click on the tab for Step 2 to simulate a user who wishes to redo an earlier step. You should see the second tab reactivate (with any previously entered values in the form fields intact) while the third and fourth tabs become disabled. After you've moved backwards in the process, the only way to progress again is to resubmit the forms. The activation and deactivation code you added in Listing 8 works perfectly.

This limitation might seem arbitrary in your example application, but in the real world it would be important. A user's answers in Step 2 might affect the questions asked in Step 4, so if the earlier answers change, the user will have to redo later steps. In these types of real-world scenarios, you would have to add additional code to your tab interface so that each time a tab is revisited, its content gets reloaded from a fresh Ajax call.

Testing your progressive enhancement

Finally, disable JavaScript functionality using your browser preferences or plug-ins. Reload the checkout page and confirm that browsers without JavaScript functionality enabled can still check out, albeit using the old, pre-Ajax interface. By progressively enhancing the old interface instead of rewiring it to depend on JavaScript functionality, you've helped ensure that a wide variety of user agents can access your application for years to come.

What you've accomplished

That was a lot of work for such a simple goal as rendering a series of forms as a single tabbed page. What did it get you? The answer, as always, comes down to user experience.

There was nothing awful about the multipage interface of Customize Me Now 1.1,

© Copyright IBM Corporation 1994, 2008. All rights reserved.

but the process of loading consecutive forms slowed down your users and disoriented them between each step. You did provide old-fashioned breadcrumb navigation. But, visually, the breadcrumb trail provided little unity to the experience of checking out. Because they looked so much like global navigation or textual content, the breadcrumb links were easier to miss than your visually distinctive tab interface. And that tab interface provides visual cohesion to your process, turning a series of disconnected pages into a single interface. The window-shade effect that occurs between steps strengthens the sense of cohesion while also subtly reminding users of their progress.

You've also subtly improved the experience for users who take detours to log in or apply a discount. In the old interface, they could get confused about where they were in the process. By placing the compound actions of an individual step within a single tab that never disappears from the page, your new checkout path keeps users oriented. Because there's no window-shade effect between the sub-steps within a single tab, users can tell that they're not yet finished with the current step.

What's left for next time

Now that you're done building your tabs, you can continue to overhaul the problem areas in your interface. The following are just some of the additional enhancements you could make:

- **Deal with Ajax failures:** Your submit handlers only account for successful Ajax calls. A real, production-quality site would have to anticipate and recover from failed Ajax calls due to server errors or network latency.
- **Provide form validation:** Input validation is one of the biggest headaches for both developers and users. You've sidestepped this thorny issue entirely so far. When done correctly, Ajax form validation can vastly improve your user experience and make your code more efficient and maintainable.
- **Keep the shopping cart visible:** Your new Ajax interface shows users their shopping carts only at the beginning and end of the process. You could add a compact view of the cart as a sidebar next to your tabs and update it with each completed step in the process. This continual feedback about the state of their orders would be of immense value to users.
- **Incorporate Back button management:** Despite your improvements, you've created one new problem for users. You've made the Back button useless within the checkout path. This is easily remedied, however, with the use of an Ajax history management library, which would allow you to

© Copyright IBM Corporation 1994, 2008. All rights reserved.

add an entry to the browser's history stack each time a new form is loaded into your tabs. That way, when users click the Back button, they will be taken back to the previous tab rather than being yanked out of the checkout process to whatever page they were on previously. To accomplish this, you can turn to a stand-alone library such as Really Simple History, which works with jQuery, Prototype, and many other Ajax frameworks. (Full disclosure: The author of this article is the code steward for Really Simple History.)

As you can see, the possibilities are endless. But the changes you've made in this installment provide a strong Ajax backbone for you to continue retrofitting your checkout path.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Downloads

Description	Name	Size	Download method
Source code for the original demo app	wa-aj-overhaul4OneClientTwo.zip	632KB	HTTP
Source code for the retrofitted demo app	wa-aj-overhaul4TwoClientTwo.zip	923KB	HTTP

Information about download methods

More downloads

- Demo: Customize Me Now 1.2[1]
- Demo: Customize Me Now 2.2[2]
- Demo: Customize Me Now: All Versions[3]

Notes

1. See the original demo application in action on the author's company Web server.
2. See the retrofitted demo application in action on the author's company Web server.
3. See all versions of the demo application, including the versions from previous installments, in action on the author's company Web server.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Resources

Learn

- Get to know the jQuery application program interface (API) at the jQuery documentation Web site.
- Participate in the jQuery community and access tutorials and forums at the Learning jQuery Web site.
- Continue your Ajax education with additional articles such as "Simplify Ajax development with jQuery" (Jesse Skinner, developerWorks, April 2007).
- For additional jQuery help, check out the book *jQuery in Action* (Manning Publication Co., February 2008).
- Check out Brian Dillard's blog, Agile Ajax, for more on jQuery and other UI topics.
- For an exhaustive overview of best practices for the security of Ajax applications, see Billy Hoffman and Bryan Sullivan's recent *Ajax Security* (Addison Wesley Professional, December 2007).
- Visit Functioning Form, the blog of renowned Web developer Luke Wroblewski, for ongoing commentary about best practices in Web form design.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download jQuery, learn all about its philosophy, and find additional plug-ins at the main jQuery Web site. The current version as of this writing is 1.2.3.
- jQuery UI Tabs, a jQuery plug-in, allows you to wrap inline or Ajax content in a tabbed interface. This plug-in is part of the jQuery UI collection of customizable widgets and user-interface components.
- jQuery Form, a jQuery plug-in, allows you to transform HTML forms into powerful Ajax components using a variety of intuitive convenience methods.
- Explore Really Simple History, an Ajax history library that restores the **Back** button and bookmark behavior users expect from Web applications.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Brian J. Dillard
In his 12 years as a Web developer, Brian J. Dillard has built rich user interfaces for companies as diverse as Orbitz Worldwide, Reflect True Custom Beauty, Archipelago LLC, and United Airlines. Now serving as VP of Ajax Development at Pathfinder Development in Chicago, Brian builds rich Internet applications for a variety of clients, participates in open source projects, and contributes to the Agile Ajax blog. He is the project lead on Really Simple History, a popular Ajax history and bookmarking library.

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

© Copyright IBM Corporation 1994, 2008. All rights reserved.