Lotus Developer Domain The Technical Resource for Lotus software LDD Today

Level: Intermediate
Works with: Domino Designer
Updated: 04-Nov-2002

by
Becky
Gibson

Earlier releases of Notes and Domino introduced the concept of collapsible, categorized views. In the client, you can expand and collapse each categorized section with a click of a mouse button. On the Web, however, this feature requires a refresh of the page to reload each category as either expanded or collapsed. The situation improved somewhat when we added DHTML support for Microsoft Internet Explorer (IE) users. However, Netscape browsers still require a page reload when a categorized section is expanded or collapsed.

This article describes how you can provide dynamic, collapsible categories in relatively small views for Netscape 4.7x/6.2 and Internet Explorer 5 and above. To do this, we use Domino formulas to create an HTML view. This view is embedded in a Domino page that contains the necessary JavaScript to interact with the view and to expand and collapse the view sections dynamically. First, we create a sample Domino database and its data. Next, we create a view with the necessary HTML contained in a column formula. Then we describe the completed page and the HTML and JavaScript used to hide and show the sections. Finally, we explain what happens when this page is viewed in IE and Netscape browsers.

This article assumes you're an experienced Notes/Domino designer with a basic knowledge of HTML. All examples used in this article are available for you to download from the **Sandbox**.

How Netscape and IE display Web pages

Before we begin, it's useful to understand how Netscape and IE display Web pages. Netscape uses *layers* to hide and show elements on a page. IE uses a cascading style sheet attribute called *display* to hide or show page elements. Note that IE ignores any layer tags on a page. This means you can create HTML to support both IE and Netscape on the same page, a fact we will take advantage of.

To demonstrate this, I created a very simple Domino database containing a list of music CD titles. This database includes a form containing fields for the artist name, the CD title, and a music category. The database has a categorized view by music category. Under each category is the CD title. Clicking the title opens a page that displays the CD data. (Obviously most production Domino databases are more sophisticated than this, but this simple example works for demonstration purposes.)

The following screen shows the resulting view in the IE browser with the Jazz category expanded:

Let's examine how to create the views in Domino.

Creating the view

In Domino Designer, create a new view named DHTML view and set the Web Access view property on the Advanced tab to Treat View Contents as HTML. The view includes two columns, each represented by a formula. The formula for the first column creates the layers and divs. (A *div* is an HTML element for defining content on a page.) The formula for the second column creates the anchor tags for each category item.

The first column is set to sort ascending and categorized. Enter this formula as the Column value formula for the first column:

```
catValue :=@ReplaceSubstring(Category;" ":"'":"\"":"/";"_");
endTags := "</div></layer></layer><br>";
outerLayer := "<layer name='" + catValue + "'>";
imgLink := "<a href=\"javascript: void 0\" onclick=\"hideShow('" + catValue + "'); return false;\"><img alt='click to
expand/collapse' border='0' height='20' width='30' src='expand.gif' name='" + catValue + "Img'></a>";
innerLayer := "<layer visibility='hidden' name='L" + catValue + "'>";
div := "<div onClick='window.event.cancelBubble=true' style='margin-left:50;' id='D" +catValue+ "'>";
REM {the pieces are combined to create the HTML that is output for each category};
endTags + outerLayer + imgLink + "<span style=\"font-size:16pt; font-weight:bold;\">" + Category + "</span>" +
innerLayer + div +"<br>"
```

Let's examine each piece of this code. First, we need the category name to be used for naming the various HTML objects. The name must be the same for all objects, so store it in catValue. Use replaceSubstring() to replace any JavaScript "special" characters (space, double quote, single quote, and slash) with an underscore. (If the category contained any special characters, this JavaScript would not work properly.)

The first column formula creates the layer and div objects. But we can't close these objects until we add the data created by the formula in the second column. Therefore, we need to close the objects from the previous category before we start the current category. To do this, store the HTML to close all of the object in endTags. The outerLayer creates the layer tag with the name attribute equal to our scrubbed category name. ImgLink contains the <a> tag and associated image. The innerLayer variable creates the inner layer tag named *L* + catValue. The div variable creates the div tag named *D* + the cat value. Finally all of the pieces are put together to create the formula with each category separated by a
:

```
endTags + outerLayer + imgLink + "<span style=\"font-size:16pt; font-weight:bold;\">" + Category + "</span>" +
innerLayer + div +"<br>"
```

When this formula is evaluated for the Jazz category in a Web browser, the browser generates the following HTML:

```
</div></layer></layer><br><layer name='Jazz'><a href=javascript:hideShow('Jazz');><img alt='click to
expand/collapse' border='0' height='20' width='30' src='expand.gif' name='JazzImg'></a>
<span style="font-size:16pt; font-weight:bold;">Jazz</span>
<layer visibility='hidden' name='LJazz'><div onClick='window.event.cancelBubble=true' style='margin-left:50;'
id='DJazz'><br>
```

Next, enter the following formula as the Column value formula for the second column. This formula is shorter because it simply generates the <a> tag to create the link surrounding the title.

```
"<A HREF=\"./by+Category/"+@Text(@DocumentUniqueID)+"?OpenDocument\">"+ Title+"</A> by
"+Artist+"<BR>"
```

Our view looks like this in Domino Designer:

Creating the page

Now that we've created the DHTML view, we need to embed it on a page with the necessary HTML and JavaScript. To do this, open Domino Designer and go to the Pages view. Click New Page to create the new page. Then right-click your mouse and select Page Properties. On the Page Info tab, set the Web Access Type to HTML. Then copy the following HTML onto the page. It contains JavaScript that we'll take a closer look at in the

next section:

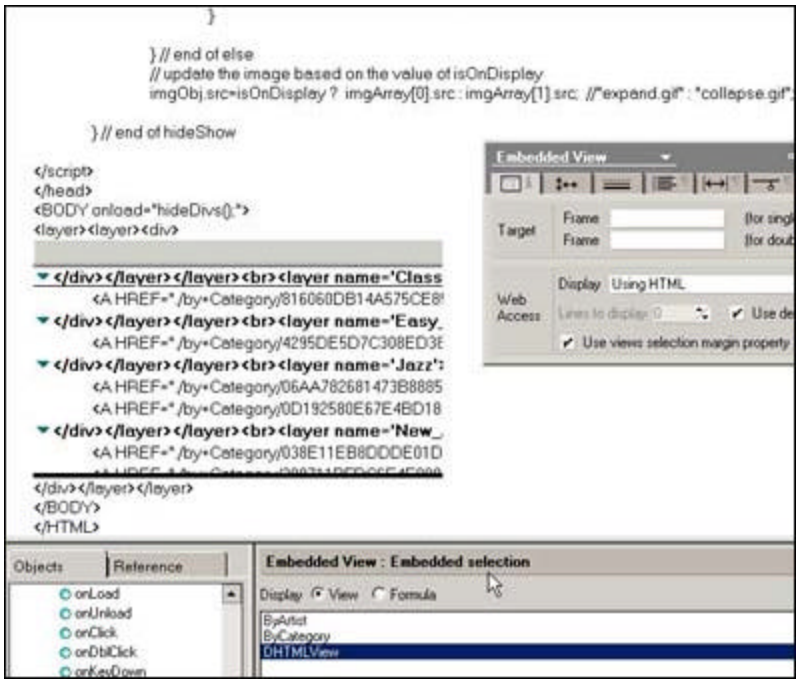
```
<HTML>
<HEAD>
<TITLE>DHTML View for All</TITLE>
<!-- Copyright (c) 2002 Iris Associates, Inc. All rights reserved.
<script language="JavaScript">
// set up the variables used
var hasLayers = false;
var objPrefix = "D";
if (document.layers) {

                hasLayers = true;
                objPrefix="L"

}
// preload the images - need this for IE 6

                var imgArray = new Array();
                imgArray[0] = new Image(30,20);
                imgArray[1] = new Image(30,20);
                imgArray[0].src = "expand.gif";
                imgArray[1].src = "collapse.gif";


function hideDivs() {

                if (hasLayers == false) {
                        var divArray = document.getElementsByTagName("DIV");
                        for (var i=0; i < divArray.length; i++) {
                                divArray[i].style.display="none";
                        }
                }

} // end of hideDivs()

                function hideShow(name) {
                        var imgName = name + "Img";
                        var imgObj;
                        var toggle;
                        var isOnDisplay = true;
                        if (hasLayers == false) {
                                imgObj = document.images[imgName];

                                toggle=document.getElementById(objPrefix + name);
                                isOnDisplay = toggle.style.display=="";
                                toggle.style.display = isOnDisplay ? "none" : "";
                        } // end of if does NOT use layers
                        else {
                                imgObj = document.layers[name].document.images[imgName];
                                toggle = document.layers[name].document.layers[objPrefix + name];
                                isOnDisplay = toggle.visibility == "show";
                                toggle.visibility = isOnDisplay ? "hide" : "show";
                                yOffset = isOnDisplay ? -toggle.clip.height : toggle.clip.height;

                                numLayers = window.document.layers.length;
```

```
                    var layerObj = null;
                    var index=null;
                    for (var i =0; i < numLayers; i++) {
                        layerObj = document.layers[i];
                        if (layerObj.name == name) {
                                index = i;
                                break;
                        }
                    } // end of for to find index
                    // next move all subsequent layers by yOffset
                    for(var j=index+1; j < numLayers; j++) {
                        layerObj = document.layers[j];
                        // move remaining objects up or down
                        layerObj.moveBy(0,yOffset);
                    }

                } // end of else
                // update the image based on the value of isOnDisplay
                imgObj.src=isOnDisplay ? imgArray[0].src : imgArray[1].src;
            } // end of hideShow


</script>
</head>
<BODY onload="hideDivs();"
<layer><layer><div>
```

Now embed the DHTML view into this page by choosing Create - Embedded Element - View and selecting DHTML view from the list of view choices. Then right-click on the embedded view and select Embedded View. On the Info tab of the Embedded View Properties box, set the Web Access Display property to Using HTML. Then, below the embedded view, add the following HTML to the page:

```
</div></layer></layer>
</BODY>
</HTML>
```

Your page should resemble the following:



You can now check out the new page by choosing Design - Preview in Web Browser to preview this page in different browser versions. Use HTML to reference this page in your application.

How the page is processed in IE and Netscape

Now let's look at how the page is processed in IE and Netscape 6.2/4.7x. IE and Netscape 6.2 both support dynamically hiding and showing elements on the page using the display style attribute. This makes the code for these browsers simpler than the code required for Netscape 4.7x.

For IE and Netscape 6.2

If we view source on this page, we see the following HTML for the Jazz section:

```
<layer name='Jazz'>
<a href=javascript:hideShow('Jazz');><img alt='click to expand/collapse' border='0' height='20' width='30'
src='expand.gif' name='JazzImg'></a>
<span style="font-size:16pt; font-weight:bold;">Jazz</span>
<layer visibility='hidden' name='LJazz'><div onClick='window.event.cancelBubble=true' style='margin-left:50;'
id='DJazz'><br>
<A HREF="./by+Category/06AA782681473B8885256B8700756E94?OpenDocument">
Bebop</A> by Charlie Parker<BR><A HREF="./by+Category/0D192580E67E4BD185256B8700758D3C?
OpenDocument">
Ken Burns Jazz</A> by Various<BR>
</div></layer></layer><br>
```

Let's go through the code item by item. IE ignores the layer tags, so we'll skip them for now. To create a clickable icon in both Netscape and IE, you must embed an image within an <a> tag. The href of the anchor tag is a call to the JavaScript function hideShow(name). The parameter passed into this function is the name of the current category Jazz. The anchor tag encloses the image used to represent the expanded and collapsed state of the section, an opened or closed book. The name attribute of the image is the category name with *Img* appended to it: JazzImg. We'll use this name in the hideShow() JavaScript function to find the image and change its src attribute to show the appropriate gif. The category itself is displayed within a . This allows us to use a style attribute to format the category as desired.

Another layer follows, which IE ignores. Next is a div; this is the HTML element that we will hide and show. It encloses the items within the Jazz category. The id attribute of the div is the letter *D* prepended to the category

name, in this case Jazz. This results in an id value of Djazz. Inside the div are a set of anchor tags, representing the CDs within the Jazz category. The href of each anchor is a Domino OpenDocument url to open the document for each CD. Finally, we close the div and two layers. This is the structure for each category:

A layer named after the category

An anchor tag enclosing a clickable image that calls our JavaScript function
Another layer
A div
An anchor tag surrounding each item in the category
End div
End layer

End layer

Next, let's examine the JavaScript function hideShow(name), which is called when the image is clicked.

```
function hideShow(name) {

        var imgName = name + "Img";
        var imgObj;
        var toggle;
        var isOnDisplay = true;
        if (hasLayers == false) {
            imgObj = document.getElementById(imgName);

            // this gets the div since its id is "D" + the category name (objPrefix is defined above)
            toggle=document.getElementById(objPrefix + name);
            isOnDisplay = toggle.style.display=="";
            toggle.style.display = isOnDisplay ? "none" : "";
        } // end of if does NOT use layers
        else {
            // Netscape code goes here

        } // end of else
        // update the image based on the value of isOnDisplay
        imgObj.src=isOnDisplay ? imgArray[0].src : imgArray[1].src; //"expand.gif" : "collapse.gif";

    } // end of hideShow
```

The name of the category is passed into the hideShow(name) function in the name parameter variable. Our example includes variables to hold the image name, image object and the div we will be hiding and showing. These are called imgName, imgObj, and toggle, respectively. Each click of the image will either hide or show the associated category elements, so we created the variable isOnDisplay to determine if the category elements are on display (and thus need to be hidden) or are hidden and need to be shown.

Earlier in the page, we initialized a variable called hasLayers to determine the functionality of the browser. We'll look at the details later. For now, we just need to know that if the browser is IE or Netscape 6.2, the hasLayers variable is false, and we enter into the if clause of the function.

Use the document.getElementById() method to get the image object and the div object. Here is where the naming becomes important. We can find the correct img and div object associated with the selected category because we used the name of the category in the name of each HTML item. The id of the div element is named the same as the category but with a *D* in front of it. The *D* is stored in the objPrefix variable at the same time that the hasLayers variable is initialized. After we have the div object, we can determine whether or not it is currently hidden and store the value in the isOnDisplay variable. The cascading style sheet element display is set to none

when an element is hidden or "" (the empty string) when an element is visible. To hide or show the div element and the <a> tags it encloses, toggle the value of the div's display attribute. When an element is hidden, all of the surrounding elements on the page are moved up to fill in the space previously used by the visible object. Likewise, when the object is made visible, all other objects are moved to accommodate it. Thus, it is very easy to hide or show an object on the page by toggling its display style attribute from none to "".

Finally, we need to adjust the image that is displayed for the category. When the category is collapsed and the details hidden, show the closed book image, expand.gif. When the category is expanded (or visible), show the open book image, collapse.gif. We can change the displayed image by changing the src attribute on the image object. Set the image src based on the value of the isOnDisplay attribute. In this code, we have preloaded the images into an array called imgSrc.

It took very few lines of code to implement this expand/collapse behavior because IE and Netscape 6.2 support dynamically hiding and showing elements on the page using the display style attribute. Unfortunately, Netscape 4.7x browsers do not fully support the display attribute in this manner, so the code for this is a bit more complicated. We need to use the Netscape 4.7x layer object, which can be hidden and shown.

Here is the else clause of the hideShow(name) function:

```
else {

        imgObj = eval("document." + name + "." + "document." + imgName);
        toggle = eval("document." + name + "." + "document." + objPrefix + name);
        isOnDisplay = toggle.visibility == "show";
        toggle.visibility = isOnDisplay ? "hide" : "show";

        yOffset = isOnDisplay ? -toggle.clip.height : toggle.clip.height;


numLayers = window.document.layers.length;

        var layerObj = null;
        var index=null;
        for (i =0; i < numLayers; i++) {
                layerObj = document.layers[i];
                if (layerObj.name == name) {
                        index = i;
                        break;
                }
        } // end of for to find index
        // next move all subsequent layers by yOffset
        for(j=index+1; j < numLayers; j++) {
                layerObj = document.layers[j];
                layerObj.moveBy(0,yOffset);
        }

} // end of else
```

This is the clause that executes when the browser supports layers and the hasLayers attribute is true.

For Netscape 4.7*x*
It is somewhat more complex to get the objects in Netscape 4.7x. In this case, the object of which we want to change the visibility is the inner layer object. This is obtained by evaluating document.layername.document.innerlayername or document.Jazz.document.LJazz. The inner layer object is stored in the toggle variable. Again, we use the naming convention to assist finding the object. The objPrefix variable is initialized to *L* in Netscape 4.7, so the name of the inner layer is always *L* plus the category name.

Netscape layers have a visibility attribute. We can change the visibility of the layer by toggling the visibility attribute value between hide and show. Store the current visibility of the inner layer in the isOnDisplay variable. Originally, all of the inner layers are created with visibility="hide" so they are not visible when the page is first loaded. Unlike IE and Netscape 6.2, hiding a Netscape 4.7 layer does not close up the space that was used, so we need to do that. Use the move() API on the layer object to move all subsequent layer objects up or down when a layer is hidden or shown.

The clip.height attribute gives the height of the inner layer. Use this height as the offset needed to move all of the subsequent layers on the page. If we're hiding the current layer, move all subsequent layers up on the page, so the offset is negative. If we're showing (or expanding) a category, the remaining layers are moved down—the offset is positive.

Netscape contains an array object that contains all of the layer objects in the document. We need to find the index of the current layer in this array. Do this by looping through all of the layers in the document until we find one with the name of the current category. In this case, the category name is Jazz. This was the parameter passed into the hideShow() function. Remember, we named the outer layer the same as the category name.

This code gets the index of the current layer:

```
numLayers = window.document.layers.length;
var layerObj = null;
var index=null;
for (i =0; i < numLayers; i++) {

        layerObj = document.layers[i];
        if (layerObj.name == name) {
                index = i;
                break;
        }

} // end of for to find index
```

Now that we have the index, use the same for loop technique to move each remaining layer by offset. The loop begins at the next outer layer object on the page or index plus one.

```
for(j=index+1; j < numLayers; j++) {

        layerObj = document.layers[j];
        // move remaining objects up or down
        layerObj.moveBy(0,yOffset);

}
```

We've moved all of the remaining layers either up or down, so all that remains is to set the correct image. This code is the same for both IE and Netscape; it only needs to update the src attribute of the image object.

Because we are moving many of the objects on the page when we hide or show a category, this technique is not appropriate for very large views with many categories. If the view becomes too large, there is a several second delay while all of the remaining objects on the page are adjusted.

A few more details

There are just a few more JavaScript details that need to be covered. At the top of the <script> section on the page you see the following code:

```
// set up the variables used
var hasLayers = false;
var objPrefix = "D";
```

```
if (document.layers) {

		hasLayers = true;
		objPrefix="L"

}
// preload the images - need this for IE 6

		var imgArray = new Array();
		imgArray[0] = new Image(30,20);
		imgArray[1] = new Image(30,20);
		imgArray[0].src = "expand.gif";
		imgArray[1].src = "collapse.gif";
```

This is where we initialize the hasLayers and objPrefix variable. Assume IE or Netscape 6.2, and initialize the hasLayers variable to true and the objPrefix variable to *D*. The objPrefix variable is used to create the name of the object of which we are going to toggle visibility. It is *D* plus the category name in IE and Netscape 6.2 to name the div object. It is *L* plus the category name in Netscape 4.7x to name the inner layer object. If document.layers is true, we know the current browser supports layers, so we can initialize the variables appropriately.

I also had difficulty with the images in IE 6. The images did not display properly so I preloaded them into an array.

There is one more step needed to display the page properly when it is first loaded. Use the display property on the div object to toggle the visibility of the category details. When the page is first loaded, we want the value of the display style on the div object to be none. Netscape 4.7x honors the display:none style attribute when a page is first loaded. But Netscape 4.7x does not allow the value of this property to be dynamically changed. If we set all of the divs to style=”display:none” there would be no way to show them in Netscape. Even though we toggle the visibility of the layer containing the div, if the div is hidden, it remains that way unless we reload the page. So we can’t initially hide all of the divs. Instead, if the browser is IE or Netscape 6.2, hide all of the divs when the page is first loaded. This is done by the hideDivs() function. This function is called from the onload event of the body object: <BODY onload=”hideDivs();”>.

Here is the code for hideDivs():

```
function hideDivs() {

		if (hasLayers == false) {
			var divArray = document.getElementsByTagName("DIV");
			for (i=0; i < divArray.length; i++) {
				divArray[i].style.display="none";
			}
		}

} // end of hideDivs()
```

Use the document.getElementsByTagName(“DIV”) method to get all of the div elements into an array. Then use a for loop to change the display style to none. Note that this can be very slow in Netscape 6.2—another reason this technique is not appropriate for very large views.

Notice the <layer><layer> tags above. These are needed because our first category formula contains the end tags for these objects, so we must create the initial version. Also note that you need to close the div and layers for the final category. Save this page as DhtmlView_page.

Also note that in my example, opening a document from an expanded folder, then going back with your browser's Back button closes all open folders. You can avoid this by adding formula and JavaScript code that "remembers" the open/closed state. (To keep our example simple and clear, we omitted this code.) Also note that my example

provides only one level of collapsible categories—it doesn't nest multiple levels of collapsible categories.

Conclusion

This article describes a simple example, but it shows the basic concepts for creating collapsible, categorized views using DHTML. I have covered the HTML layout using layers and divs that can be hidden and shown through JavaScript. Use the power of Domino formulas to create a view displayed as HTML. Embed this view into a page that contains your JavaScript functions, and you have a page to display your categorized view in any of the current popular browsers. You can easily use this same technique in your more complicated applications. Add formulas to calculate your categories, and add cascading style sheets to spiff up your HTML. Feel free to experiment—and let us know how it works!

ABOUT THE AUTHOR

Becky Gibson is a Senior Software Engineer at IBM, currently working on Web application development. Previously she worked on the Domino Web Server and Notes Client teams. Becky joined the Notes/Domino team in July 1997 after 12+ years working on various versions of Lotus 1-2-3. Her current expertise is in Java, JavaScript, DHTML, and Web development. She has a BS in Natural Resource Management, with a minor in Soil Science(!) from the University of Maine, and a MS in Computer Science from Boston University Metropolitan College. In her free time, Becky enjoys outdoor and travel photography and has recently taken up figure skating.