Custom Values in the 4D Ajax Framework

By Tim Penner, Technical Services Team Member, 4D Inc.

Technical Note 08-23

Abstract

The 4D Ajax Framework is a powerful and flexible tool with many features. This Technical Note covers the custom values feature in the 4D Ajax Framework. An example is provided in the form of a car dealership internal quoting system to demonstrate the use of custom values.

Introduction

This Technical Note covers custom values in the 4D Ajax Framework. An example in the form of a car dealership internal quoting system is presented to demonstrate sending data to and from the frontend to the backend via custom values. A sample database is included.

What is it?

A custom value is a variable, in the form of a name/value pair defined by the developer on the frontend and the backend. Custom values can be sent from the web browser (frontend) to the 4D Database (backend) and vice versa.

Why use it?

Custom Values can be used to send or retrieve data from the web browser (frontend) to the 4D Database (backend). This allows custom data to be saved to the database. They can also be used to retrieve data from the database. This provides flexibility for the developer in that they are not limited to transferring data via 4DAF objects such as a Data Grid, but rather with a single variable.

Data Bridge 2.0

The Technical Note example database uses the new Data Bridge 2.0 object within the 4D Ajax Framework. This object replaces the previous object, Bridge. At the time this Technical Note was released, Data Bridge 2.0 was still in development and marked as Beta.

How does it work?

Custom Values can be sent to the 4D Database (backend) as part of a chart, grid, or query. The 4D Database (backend) can act upon the custom values it receives and send additional information back to the web browser (frontend) also as custom values. This process is three fold:

1) Web Page (frontend) sends custom values to the backend via JavaScript
2) 4D Database (backend) receives custom values from the frontend, acts upon the values according to what the developer coded in the ***DAX_DevHook_OnQuery*** method, then sends data back to the frontend via 4D commands
3) Web Page (frontend) receives custom values from the backend via JavaScript

Frontend: Sending custom values to the backend

Chart

When sending custom values to the backend via a chart, the code would look like:

```
// setup the chart
var myChart = new chartViewer('myChart', $('myChartDiv'));

// clear all existing values
myChart.clearCustomValues();

// add values to the chart
myChart.addCustomValue('valueName', 'myCustomValue');

// refresh chart to include custom values
myChart.refresh();
```

Since the initial chart call will not include custom values, make sure that a .refresh is called to update the chart with custom values sent.

Grid

When sending custom values to the backend via a grid, the code would look like:

```
// setup the grid
var myDataGrid = new dax_dataGrid(‘myTable’, $(‘myGridDiv’));

// clear all existing values
myDataGrid.clearCustomValues();

// add values to the query
myDataGrid.addCustomValue('valueName', 'myCustomValue');

// run query with stored custom values to 4D
myDataGrid.runQuery();
```

Query

When sending custom values to the backend via a query, the code would look like:

```
// clear all existing values
myQuery.clearCustomValues();

// add values to the query
myQuery.addCustomValue('valueName', 'myCustomValue');

// run query with stored custom values to 4D
myQuery.runQuery();
```

As you can see, all of the methods of adding custom values are pretty similar. The all have a ***.clearCustomValues()*** function and a ***.addCustomValues()*** function.

Backend: Working with custom values

To act upon custom values sent to the backend you must modify the ***DAX_DevHook_OnQuery*** project method and use the ***DAX_Dev_GetWebVar*** command with the custom value name as the parameter.

Let's assume a custom value was sent to the backend with a name of "modelID".

```
` get custom value named "modelID" and set it to $modelID_t
$modelID_t:=DAX_Dev_GetWebVar ("modelID")

If ($modelID_t#"") ` if $modelID_t is not empty
    ` set $queryDone_b to true
    $queryDone_b:=True

    ` do something
    ` build matching "customVarName_at" and "customVarValue_at" text arrays

    ` send custom values to the Web Browser (frontend)
    DAX_Dev_SetCustomVariables (->customVarName_at;->customVarValue_at)

End if
```

The above code sample uses the ***DAX_Dev_GetWebVar*** command to set *$modelID_t* to the *value* of the custom value identified by the *name* "modelID". A check is then made to make sure that the value retrieved is not empty, and if so, *$queryDone_b* is set to true. The "*do something*" section is where you would actually act upon the custom value and build the information to be sent back to the frontend. The information to send to the frontend should be assembled in two matching text arrays; *customVarName_at* and *customVarValue_at*. When the values are received from the frontend these two arrays will be merged together, so it is very important that the order of *customVarName_at* matches the order of *customVarValue_at*, meaning that *customVarName_at[5]* is the matched pair of *customVarValue_at[5]*. The matched pair is sent to the frontend using the ***DAX_Dev_SetCustomVariables*** command.

Frontend: Getting custom values from the backend

When getting the custom values back from the 4D database backend, the code would look like:

```
myCustomValues = myDataGrid.getCustomValuesFrom4D(); // get values from 4D
```

The above code would assign the custom values from 4D to the 'myCustomValues' object. To get the total number of returned values, the code would look like:

```
myCustomValues.length // number of returned values
```

Custom Values are in name/value pairs. Each pair is assigned as array elements. Each array element has a .name and a .value property that is accessible via code like so:

```
myCustomValues[0].name // first custom value 'name'
myCustomValues[0].value // first custom value 'value'
myCustomValues[1].name // second custom value 'name'
myCustomValues[1].value // second custom value 'value'
```

Note *JavaScript arrays are numbered with integers starting from 0*

Example

The example database included with this Technical Note comes in three flavors:

- Interpreted Source Database for 4D 2004
- Interpreted Source Database for 4D v11 SQL
- Merged Database for 4D 2004

The interpreted source databases allow access to the Developer Hooks and to perform further customization in the back end. Otherwise, feel free to connect to the merged version of this database.

Note *The example provided is based on a BETA version of the 4D Ajax Framework. In order to use the interpreted source databases you will need this version. As this version has not been released, for the purposes of testing the example application, please use the merged version of the example.*

For the latest information regarding 4D product releases please visit: http://www.4d.com/corporate/releases.html

The example can be thought of as a car dealership internal quoting system. The database holds pricing information pertaining to cars and the colors/options available for each model. Color and option prices vary from model to model. All of this is presented with a web frontend that accurately displays the prices reflected in the database backend, constantly updated each time a different model is selected.

Note *The price values represented in this database are fictional.*

Installing the framework

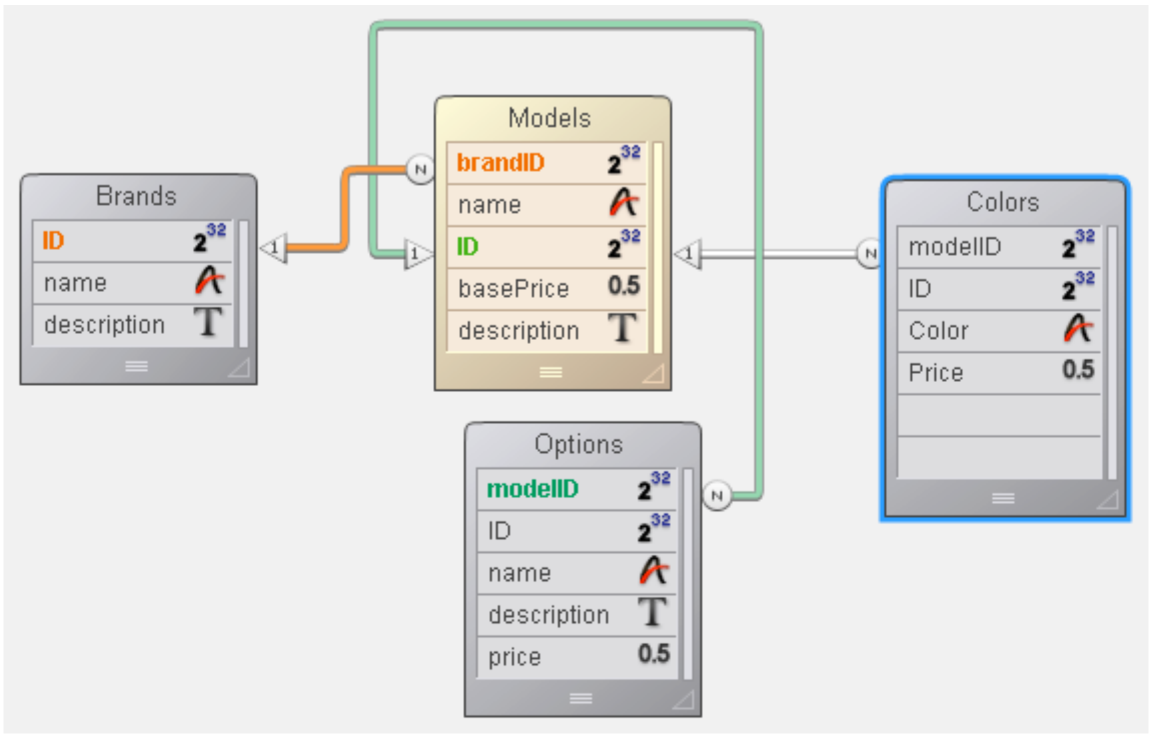
The source databases included with this Technical Note do not include the 4D Ajax Framework component. So in order to use them, you must install the framework. When installing the framework, make sure to use the version marked 11.2 or higher, and follow the installation instructions that are included with the framework. The only special customization that needs to be made is a modification to the ***DAX_DevHook_OnQuery*** project method at line 53:

```
myDevHook (->$queryDone_b)
```

The above line of code needs to be added to the ***DAX_DevHook_OnQuery*** project method at line 53 (after installing the framework).

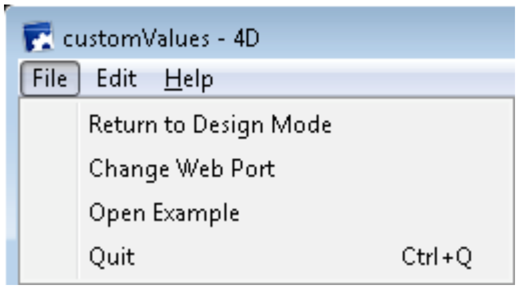
Database Structure

The sample database has four tables; [Brands], [Models], [Options], and [Colors]. The relationships are made visible in the following screenshot:



Database Menu

Inside the 4D Database there are a couple options available from the File drop down menu.

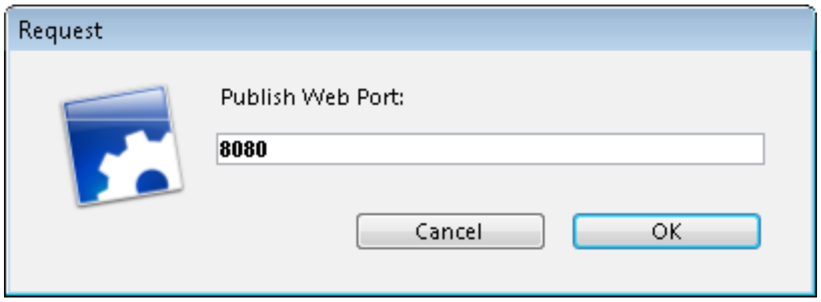


Return to Design Mode

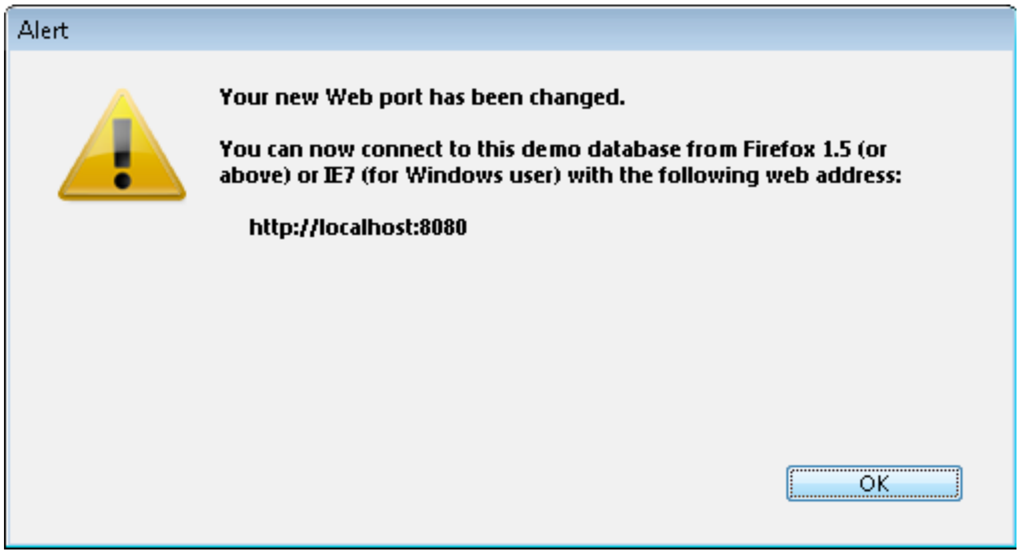
The ***Return to Design Mode*** option returns the user to Design Mode.

Change Web Port

The ***Change Web Port*** option is used to change the port that the database's webserver is running on. Choosing this option will prompt the user with the following dialog:



If the cancel button is pressed the port will remain unchanged; if the OK button is pressed the port will be changed and you will be alerted with the following dialog:


Alert

Your new Web port has been changed.

You can now connect to this demo database from Firefox 1.5 (or above) or IE7 (for Windows user) with the following web address:

http://localhost:8080

Open Example

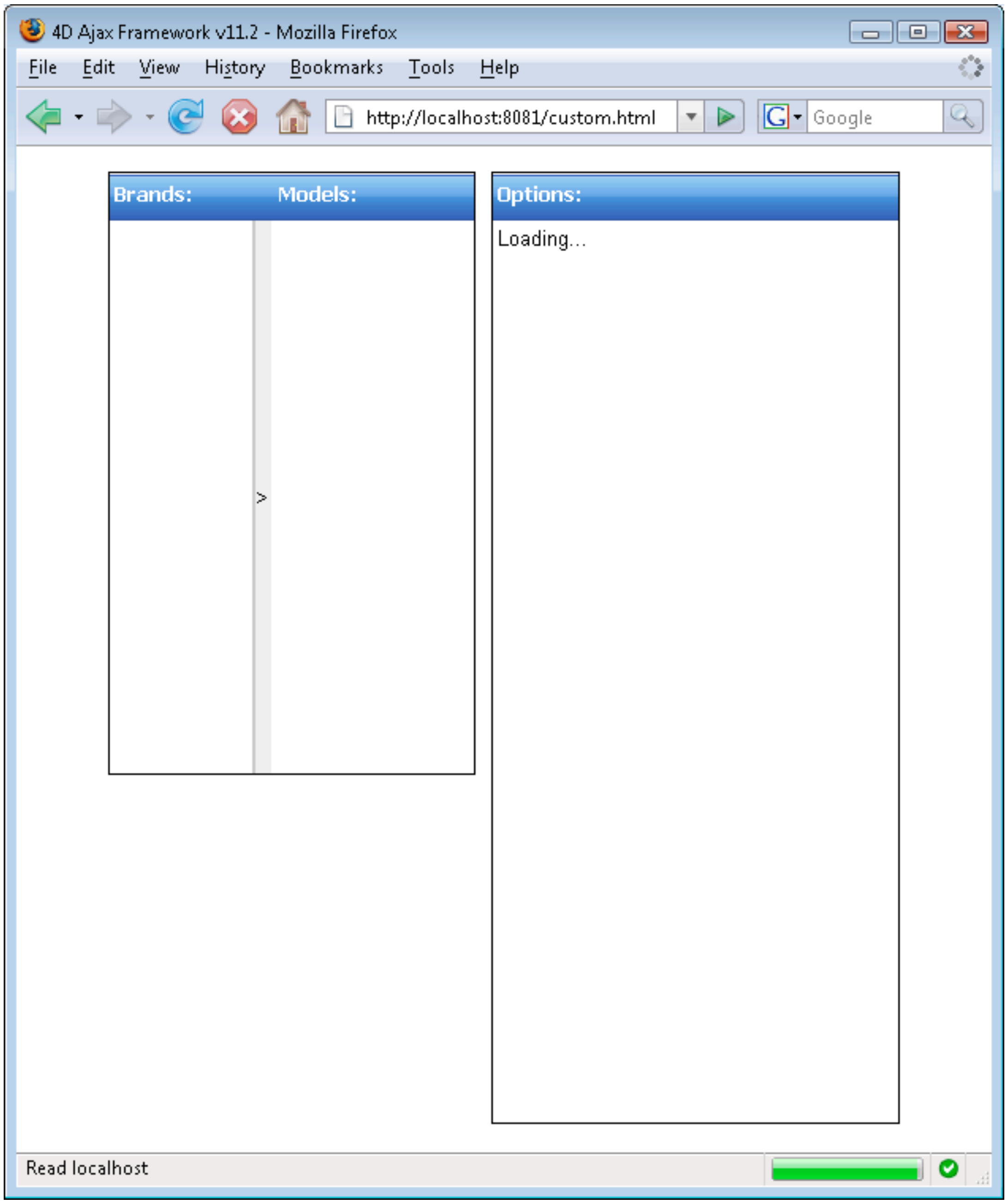
The ***Open Example*** option launches the example web page in a web browser.

Quit

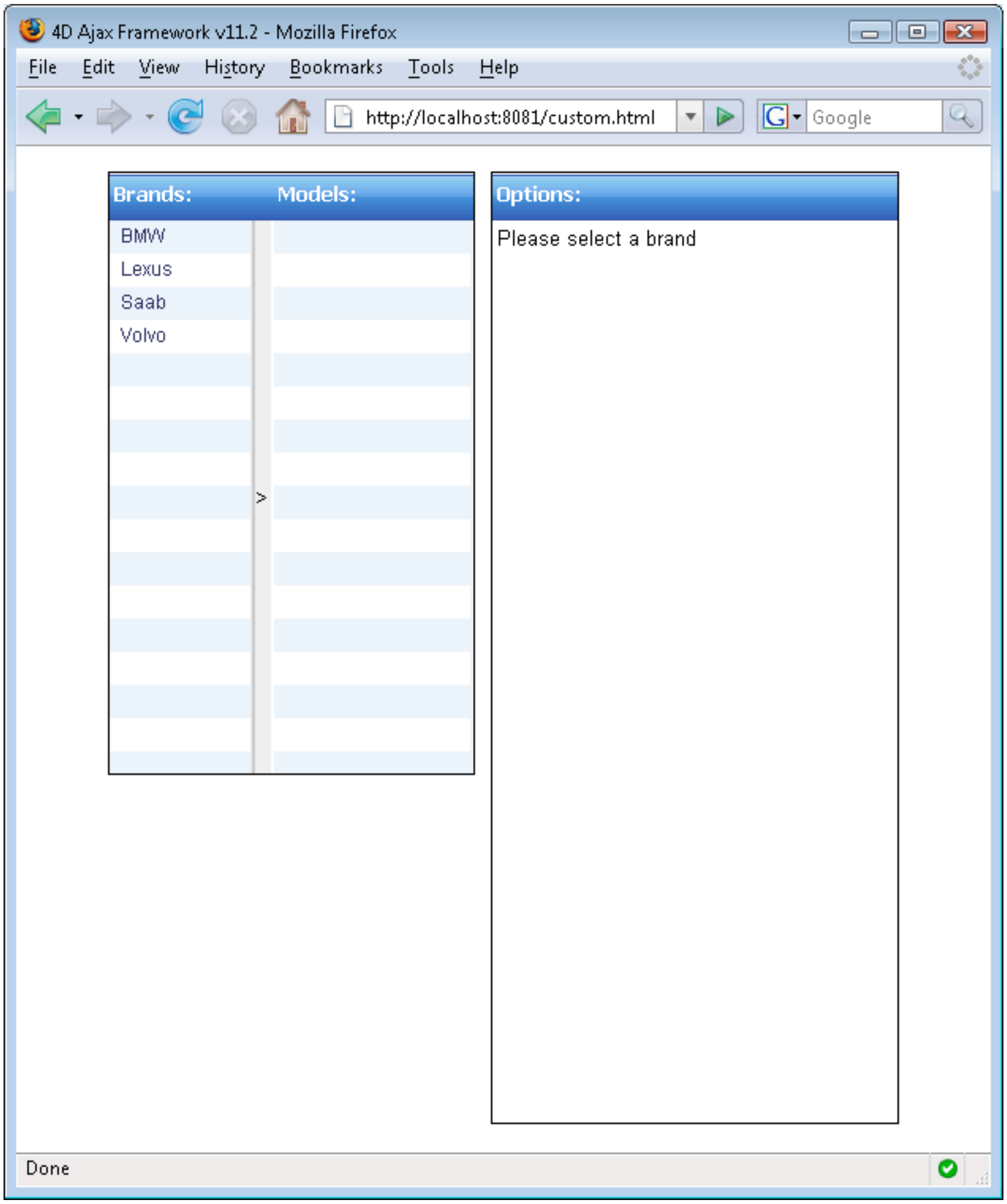
The ***Quit*** option exits the database.

Web Frontend

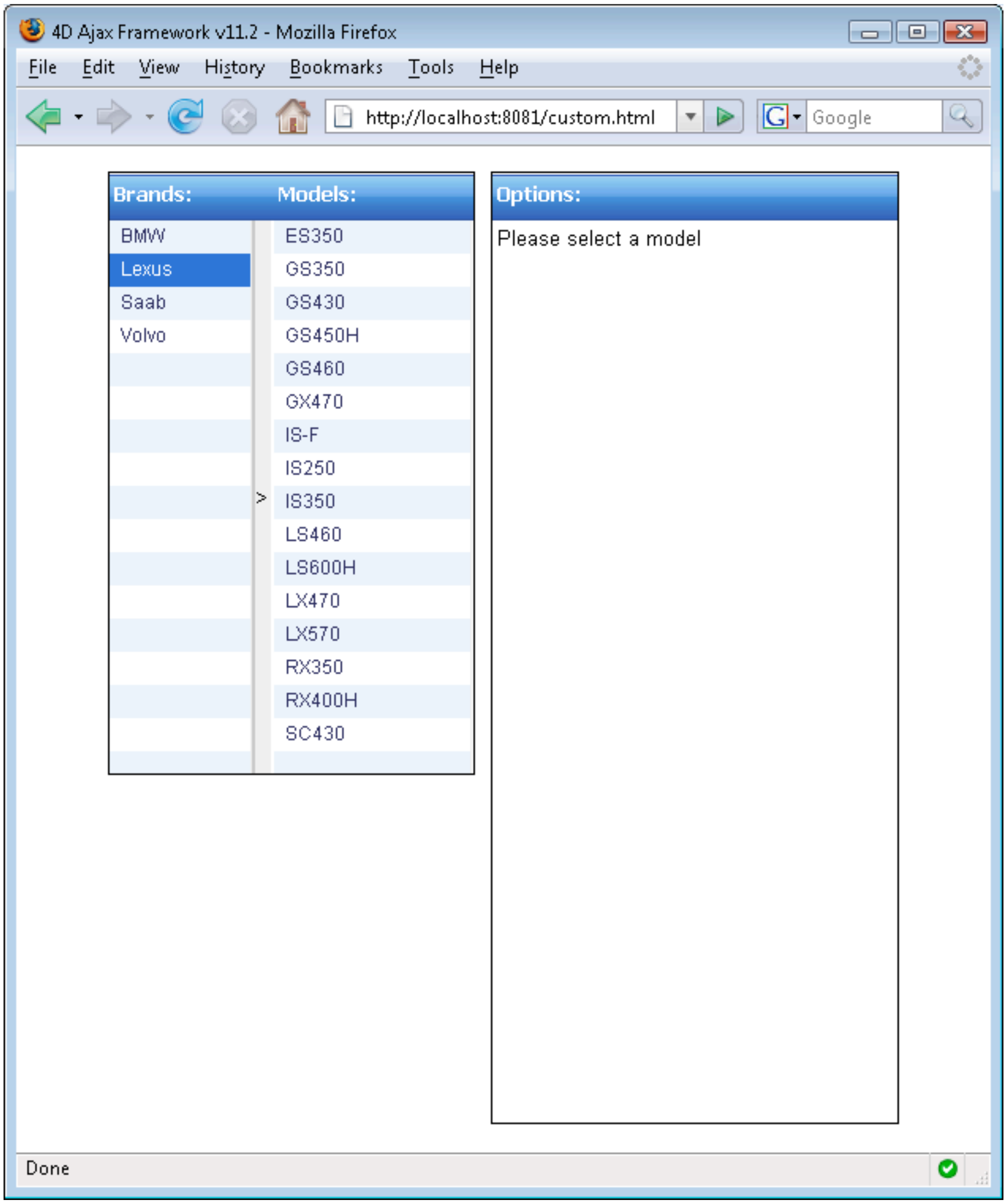
When a user navigates to the web page they are presented with a list of Brands, Models, and Options. When the web page is loading the Brands and Models columns are empty and the Options div displays "Loading…".


4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Brands:
Models:
Options:
Loading...
>
Read localhost

The Brands column is populated with a list of car makers found in the database when the web page finishes loading.


4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Brands:
Models:
Options:
BMW
Lexus
Saab
Volvo
Please select a brand
Done

The Models column is populated with the cars found in the database each time a different brand from the Brand column is clicked.

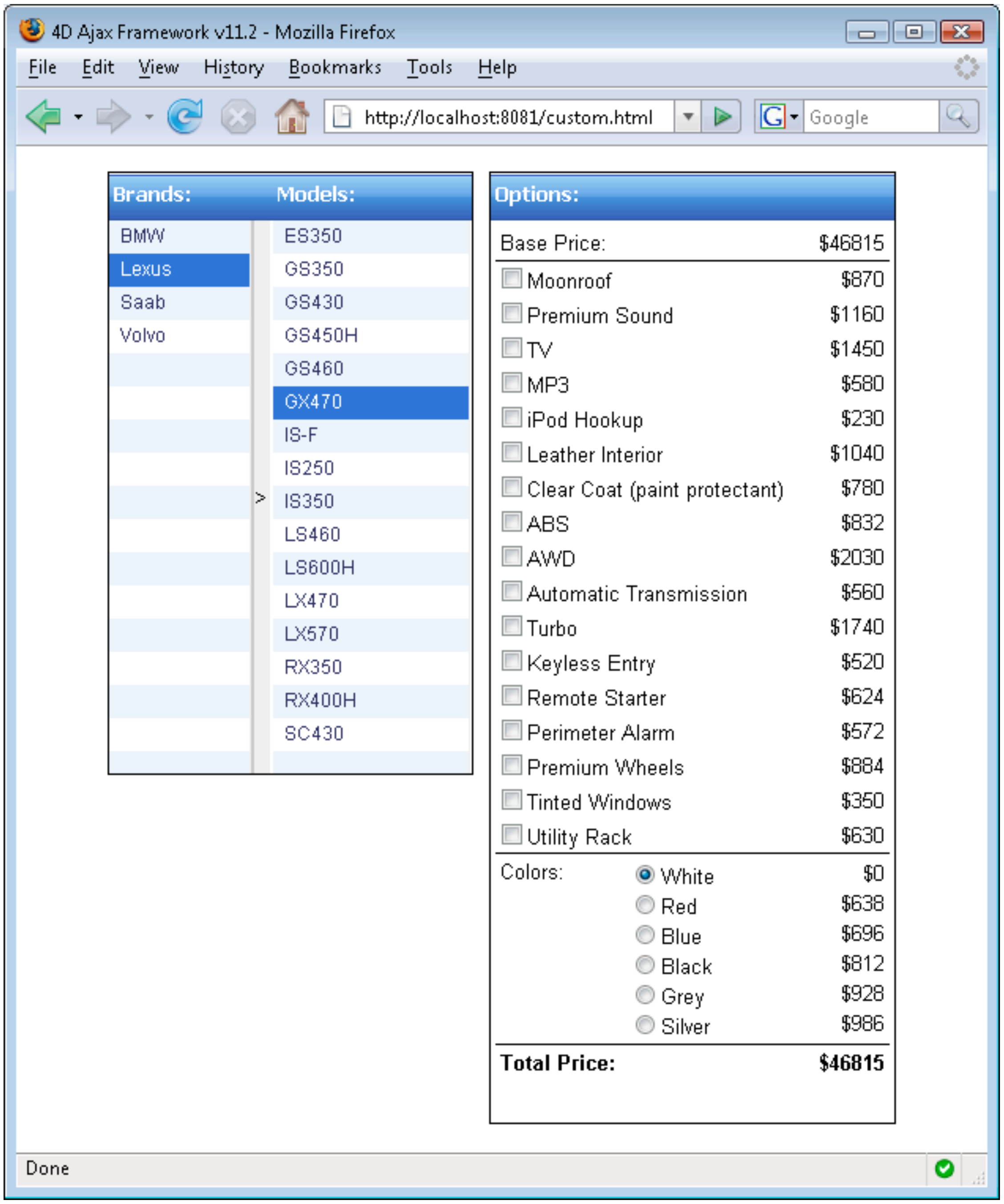

4D Ajax Framework v11.2 - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
http://localhost:8081/custom.html
Google
Done

Brands:	Models:
BMW	ES350
Lexus	GS350
Saab	GS430
Volvo	GS450H
	GS460
	GX470
	IS-F
	IS250
	IS350
	LS460
	LS600H
	LX470
	LX570
	RX350
	RX400H
	SC430

Options:

Please select a model

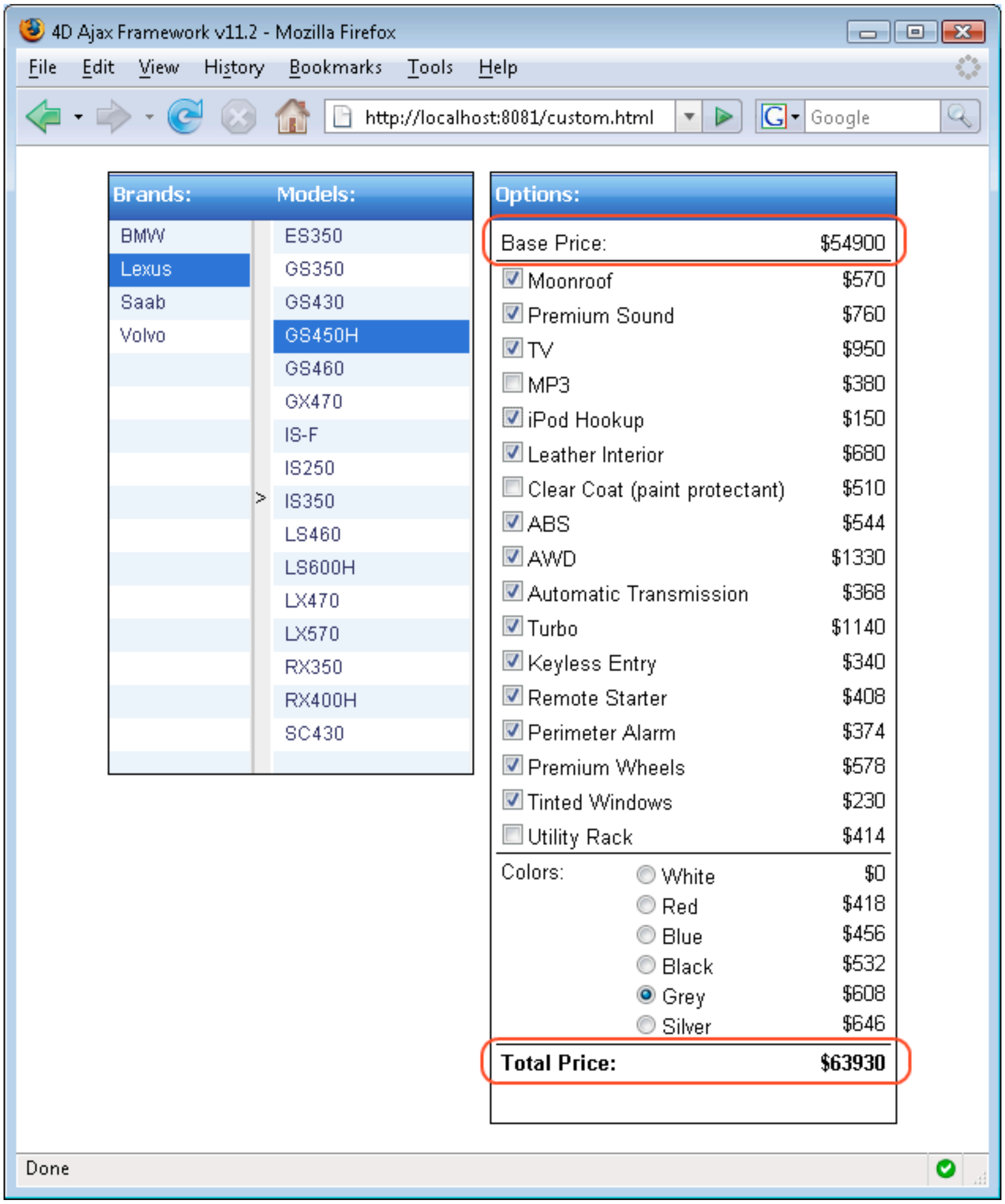
The Options pane displays the options available for the specific car model selected. The options are dynamically loaded each time a different car model is selected.


4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Done

Brands:	Models:
BMW	ES350
[✓ Lexus]	GS350
Saab	GS430
Volvo	GS450H
	GS460
	[✓ GX470]
	IS-F
	IS250
	IS350
	LS460
	LS600H
	LX470
	LX570
	RX350
	RX400H
	SC430

Options:		
Base Price:		$46815
☐ Moonroof		$870
☐ Premium Sound		$1160
☐ TV		$1450
☐ MP3		$580
☐ iPod Hookup		$230
☐ Leather Interior		$1040
☐ Clear Coat (paint protectant)		$780
☐ ABS		$832
☐ AWD		$2030
☐ Automatic Transmission		$560
☐ Turbo		$1740
☐ Keyless Entry		$520
☐ Remote Starter		$624
☐ Perimeter Alarm		$572
☐ Premium Wheels		$884
☐ Tinted Windows		$350
☐ Utility Rack		$630
Colors:	[✓ White]	$0
	Red	$638
	Blue	$696
	Black	$812
	Grey	$928
	Silver	$986
Total Price:		**$46815**

As the user selects the Options they would like for the car shown, the Total price listed in the bottom right of the Options pane is updated.


4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Done

Brands:	Models:
BMW	ES350
[✓ Lexus]	GS350
Saab	GS430
Volvo	[✓ GS450H]
	GS460
	GX470
	IS-F
	IS250
	IS350
	LS460
	LS600H
	LX470
	LX570
	RX350
	RX400H
	SC430

Options:	
Base Price:	$54900
✓ Moonroof	$570
✓ Premium Sound	$760
✓ TV	$950
☐ MP3	$380
✓ iPod Hookup	$150
✓ Leather Interior	$680
☐ Clear Coat (paint protectant)	$510
✓ ABS	$544
✓ AWD	$1330
✓ Automatic Transmission	$368
✓ Turbo	$1140
✓ Keyless Entry	$340
✓ Remote Starter	$408
✓ Perimeter Alarm	$374
✓ Premium Wheels	$578
✓ Tinted Windows	$230
☐ Utility Rack	$414
Colors: ○ White	$0
○ Red	$418
○ Blue	$456
○ Black	$532
[✓ Grey]	$608
○ Silver	$646
Total Price:	**$63930**

Each time that the user selects a different Brand or Model the Options pane, and the values stored in it, it is wiped clean so that the values for the new selection can be displayed.

Behind the scenes

When a user visits the web site the custom.html page begins to load.

Anatomy of the custom.html page:

First the browser needs to be told that this is valid XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Then the start of the HTML section:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Start of the head secton:

```
<head>
```

Set the character set to UTF-8:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Set the title.

```
<title>4D Ajax Framework v11.2</title>
```

Load the required 4DAF JavaScript files:

```
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/dev/callbacks.js"></script>
<script language="javascript" type="text/javascript" charset="utf-8"
src="dax/js/localization/resources_en.js"></script>
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/js/compile.js"></script>
```

Load the Stylesheets.

```
<!-- stylesheets -->
<link rel="stylesheet" charset="ISO-8859-1" href="dax/themes/basic/basic_gz.css"
media="all" type="text/css" title="XPress" />
<link rel="alternate stylesheet" charset="ISO-8859-1"
href="dax/themes/vista/vista_gz.css" media="all" type="text/css" title="Vista" />
<link rel="alternate stylesheet" charset="ISO-8859-1"
href="dax/themes/osx/osx_gz.css" media="all" type="text/css" title="Mack Daddy" />
<link rel="alternate stylesheet" charset="ISO-8859-1"
href="dax/themes/leopard/leopard_gz.css" media="all" type="text/css"
title="Leopard" />
<link rel="stylesheet" charset="ISO-8859-1" href="dax/themes/dash/dash.css"
media="all" type="text/css" />
<link rel="stylesheet" charset="ISO-8859-1" href="dax/dev/custom.css" media="all"
type="text/css" />
```

Start the custom JavaScript section:

```
<script>
<!--
```

Set a few custom variables for the price calculation script:

```
var $totalPrice = 0;
var $lastColorPrice = 0;
```

Start the dax_loginSuccess() function definition; this function is evaluated upon successful login to the framework:

```
function dax_loginSuccess(){
```

Set the Options div to display “Please select a brand”:

```
    $('CarOptionsDiv').innerHTML = "Please select a brand";
```

Setup the Brands grid:

```
    var BrandsGrid = new dax_dataGrid('Brands',$('CarBrandDiv'), 0, 0, false);
            BrandsGrid.go();                  // go
            BrandsGrid.disableAutoRefresh();  // disable auto refreshing the grid
            BrandsGrid.hideColumn(0);         // ID column
            BrandsGrid.setColumnWidth(1,80);  // Name
            BrandsGrid.onDataRowClick = BrandClicked; // call BrandClicked() function
when the row is clicked
```

Setup the Models grid:

```
    var ModelsGrid = new dax_dataGrid('Models',$('CarModelsDiv'), 0, 0, false);
            ModelsGrid.go();                  // go
            ModelsGrid.disableAutoRefresh();  // disable auto refreshing the grid
            ModelsGrid.setColumnWidth(0,110);        // Model Name
            ModelsGrid.hideColumn(1);                // model ID
            ModelsGrid.newQuery();                   //start a new query
            ModelsGrid.addQuery('brandID', '=', 0); // query to get 0 results
            ModelsGrid.runQuery();                   // run the query
```

Begin the BrandClicked function definition, within dax_loginSuccess():

```
    function BrandClicked(row, column, recordId, fieldReference){
            $totalPrice = 0;
            $lastColorPrice = 0;
            $('CarOptionsDiv').innerHTML = "Please select a model";
            var BrandID = BrandsGrid.getCellValue(row, 0); // get Cell value from
field 0 of the selected row
            ModelsGrid.newQuery();                        // start a new query
            ModelsGrid.addQuery('brandID','=',BrandID);   // Query based on BrandID
            ModelsGrid.runQuery();                      // run the query
            ModelsGrid.onDataRowClick = ModelClicked; // set on clicked event
    } // end BrandClicked
```

Begin the ModelClicked function definition, within dax_loginSuccess():

```
    function ModelClicked(row, column, recordId, fieldReference){
            $totalPrice = 0;
            $lastColorPrice = 0;
            $('CarOptionsDiv').innerHTML = "Getting options from backend";
            var ModelID = ModelsGrid.getCellValue(row, 1); // get cell value from
field 3 of the selected row
            prepOptions(ModelID);
    } //end ModelClicked
```

Begin the prepOptions function definition, within dax_loginSuccess():

```
    function prepOptions(modelID){
            myQuery = new dax_query('Models');
            myQuery.clearCustomValues();
            myQuery.addCustomValue('modelID', modelID);
            myQuery.handler = getOptionValues;
            myQuery.runQuery();
    } // end prepOptions
```

Begin the getOptions function definition, within dax_loginSuccess():

```
    function getOptionValues(){
            myCustomValues = myQuery.getCustomValuesFrom4D();
            var len = myCustomValues.length;
            if(len!=0){
                    $optionsHTML = "<form><table border=0 cellspacing=0 cellpadding=0
width=100%>\r";
                    for(var i = 0; i<len;i++){
                            if(myCustomValues[i].name == "Base Price"){
                                    $basePrice=myCustomValues[i].value;
                                    $optionsHTML = $optionsHTML + "<tr><td
style=\"padding: 3px\">Base Price:</td>\r<td align=right style=\"padding: 3px\">$"+
$basePrice +"</td></tr>\r";
                                    $optionsHTML = $optionsHTML + "<tr height=1
bgcolor=#000000><td colspan=2></td></tr>\r";
                            }else{ // end if / start else
                                    $name = myCustomValues[i].name;
                                    $value = myCustomValues[i].value;
                                    if($name=="Colors"){
                                            $spanForColors = $value;
                                    }else{ // end COLOR
                                            $optionsHTML = $optionsHTML + "<tr>\r";
                                            $optionsHTML = $optionsHTML + "<td><input
type=checkbox value=\""+ $value +"\" ID=\"DaxCarOption\" name=\""+ $name +"\"
onclick=\"calcTotal(this);\">"+ $name +"</td>\r";
                                            $optionsHTML = $optionsHTML + "<td align=right
style=\"padding: 3px\">$"+ $value +"</td>\r";
                                            $optionsHTML = $optionsHTML + "</tr>\r";
                                    } // end COLOR else
                            } // end BASE PRICE else
                    } // end for loop
                    $optionsHTML = $optionsHTML + "<tr height=1 bgcolor=#000000><td
colspan=2></td></tr>\r";
                    $optionsHTML = $optionsHTML + "<tr><td valign=top colspan=2
style=\"padding: 3px\"><span ID=\"ColorsDropDown\"></span></td></tr>\r";
                    $optionsHTML = $optionsHTML + "<tr height=1 bgcolor=#000000><td
colspan=2></td></tr>\r";
                    $optionsHTML = $optionsHTML + "<tr><td><b style=\"padding:
3px\">Total Price:</b></td>\r<td align=right style=\"padding: 3px\"><b><div
ID=\"TotalPrice\"></div></b></td></tr>\r";
                    $optionsHTML = $optionsHTML + "</table>\r</form>";
                    $('CarOptionsDiv').innerHTML = $optionsHTML;
                    $('ColorsDropDown').innerHTML = $spanForColors;
                    $totalPrice = $basePrice;
                    $('TotalPrice').innerHTML = "$"+$totalPrice;
                    $('White').click();
            } // end IF len # 0
    } // end getOptionValues
```

End the dax_loginSuccess function definition:

```
} // end dax_loginSuccess
```

Begin the calcTotal function definition:

```
function calcTotal(item){
    if(item.checked==true){
            $totalPrice = parseInt($totalPrice) + parseInt(item.value);
            $('TotalPrice').innerHTML = "$"+$totalPrice;
    }else{
            $totalPrice = parseInt($totalPrice) - parseInt(item.value);
            $('TotalPrice').innerHTML = "$"+$totalPrice;
    }
}
```

Begin the calcColor function definition:

```
function calcColor(item){
  $totalPrice=parseInt($totalPrice)+parseInt(item.value)-parseInt($lastColorPrice);
  $('TotalPrice').innerHTML = "$"+$totalPrice;
  $lastColorPrice = parseInt(item.value); // update the price of the last color
                                          // used for the next calculation
}
```

Begin the dax_loginFail function; which is evaluated if the login to the framework fails:

```
function dax_loginFail(){

}
```

End the custom JavaScript section:

```
-->
</script>
```

End the head section:

```
</head>
```

Start the body section and set the onload and onunload browser events:

```
<body onload="dax_login('Guest','');" onunload="dax_logout();">
```

Format the web page:

```
<br>
<div id="mainContent" style="width: 500px; margin-left: auto; margin-right: auto;">
    <div style="float: left; width: 230px; border-left: 1px solid #000000; border-
right: 1px solid #000000; border-top: 1px solid #000000; border-bottom: 1px solid
#000000;">
            <div style="float: left;">
                    <div style="float: left; width: 90px;">
                            <div class="window_t_a" style="float: top; width: 90px;">
                                    <div class="window_title" style="width: 90px;
padding-left: 3px;">Brands:</div>
                            </div>
                            <div style="float: left;">
                                    <div id="CarBrandDiv" style="width: 90px; height:
350px;"></div>
                            </div>
                    </div>
                    <div style="float: left;">
                            <div class="window_t_a" style="float: top; width:
14px;"></div>
                            <div style="background-color: #eeeeee; border-right: 1px
solid #ffffff; border-left: 1px solid #cccccc;">
                                    <div id="indicatorone" style="display: table-cell;
vertical-align: middle; width: 10px; height: 350px; background-color: #eeeeee;
border-right: 1px solid #ffffff; border-left: 1px solid #cccccc;">&gt;</div>
                            </div>
                    </div>
                    <div style="float: left;">
                            <div class="window_t_a" style="float: top; width: 126px;">
                                    <div class="window_title" style="padding-left:
3px;">Models:</div>
                            </div>
                            <div>
```

```
                              <div id="CarModelsDiv" style="width: 125px; height: 
350px;"></div>
                        </div>
                  </div>
            </div>
      </div>
      <div style="float: left; padding-left: 10px;">
            <div style="border-left: 1px solid #000000; border-right: 1px solid 
#000000; border-top: 1px solid #000000;border-bottom: 1px solid #000000;">
                  <div class="window_t_a" style="float: top; width: 256px;">
                        <div class="window_title" style="padding-left: 3px; width: 
250px;">Options:</div>
                  </div>
                  <div ID="CarOptionsDiv" style="width: 250px; height: 565px; 
padding:3px">Loading...</div>
            </div>
      </div>
</div>
```

End the body section:

```
</body>
```

End the HTML section:

```
</html>
```

Walking through the code:

When the custom.html web page is loaded into the browser, the code is parsed. First a few headers are sent to the browser, then the head section starts to load the 4DAF JavaScript includes. Then the stylesheets are loaded followed by the custom JavaScript section. When the custom JavaScript section is evaluated, the two variable declarations at the top level of the scope are set:

```
var $totalPrice = 0;
var $lastColorPrice = 0;
```

Finally the main content layout is loaded. While the page is loading, the Brands and Models divs are empty and the Options div says “Loading…”. Here is the code section for the div layout (the content of the CarOptionsDiv is highlighted in bold):

```
<div id="mainContent" style="width: 500px; margin-left: auto; margin-right: auto;">
      <div style="float: left; width: 230px; border-left: 1px solid #000000; border-
right: 1px solid #000000; border-top: 1px solid #000000; border-bottom: 1px solid 
#000000;">
            <div style="float: left;">
                  <div style="float: left; width: 90px;">
                        <div class="window_t_a" style="float: top; width: 90px;">
                              <div class="window_title" style="width: 90px; 
padding-left: 3px;">Brands:</div>
                        </div>
                        <div style="float: left;">
                              <div id="CarBrandDiv" style="width: 90px; height: 
350px;"></div>
                        </div>
                  </div>
                  <div style="float: left;">
```

```
                        <div class="window_t_a" style="float: top; width:
14px;"></div>
                        <div style="background-color: #eeeeee; border-right: 1px
solid #ffffff; border-left: 1px solid #cccccc;">
                              <div id="indicatorone" style="display: table-cell;
vertical-align: middle; width: 10px; height: 350px; background-color: #eeeeee;
border-right: 1px solid #ffffff; border-left: 1px solid #cccccc;">&gt;</div>
                        </div>
                  </div>
                  <div style="float: left;">
                        <div class="window_t_a" style="float: top; width: 126px;">
                              <div class="window_title" style="padding-left:
3px;">Models:</div>
                        </div>
                        <div>
                              <div id="CarModelsDiv" style="width: 125px; height:
350px;"></div>
                        </div>
                  </div>
            </div>
      </div>
      <div style="float: left; padding-left: 10px;">
            <div style="border-left: 1px solid #000000; border-right: 1px solid
#000000; border-top: 1px solid #000000;border-bottom: 1px solid #000000;">
                  <div class="window_t_a" style="float: top; width: 256px;">
                        <div class="window_title" style="padding-left: 3px; width:
250px;">Options:</div>
                  </div>
                  <div ID="CarOptionsDiv" style="width: 250px; height: 565px;
padding:3px">Loading...</div>
            </div>
      </div>
</div>
```
As you can see in the code snippet, the .innerHTML of the div identified as ID=CarOptionsDiv says "Loading…"

Note *The .innerHTML property of a div is the section between <div> and </div>*

Once all of the page content is finished loading, the browser's onload event is triggered, which logs in to the 4D Ajax Framework:
```
<body onload="dax_login('Guest','');" onunload="dax_logout();">
```

Upon successful login to the framework the dax_loginSuccess() function is triggered. The first line of code in the dax_loginSuccess() function modifies the .innerHTML property for the *CarOptionsDiv* to equal "Please select a brand":
```
$('CarOptionsDiv').innerHTML = "Please select a brand";
```

The Brands div is then populated with a Data Grid displaying the car brands found in the database:
```
var BrandsGrid = new dax_dataGrid('Brands',$('CarBrandDiv'), 0, 0, false);
   BrandsGrid.go();                         // go
   BrandsGrid.disableAutoRefresh();         // disable auto refreshing the grid
   BrandsGrid.hideColumn(0);                // ID column
   BrandsGrid.setColumnWidth(1,80);         // Name
   BrandsGrid.onDataRowClick = BrandClicked; // call BrandClicked() function when
                                             // the row is clicked
```
The BrandsGrid object registers an onDataRowClick event to fire the BrandClicked() function each time a row is clicked.

The Models div is then populated with a Data Grid displaying the 0 results:

```
var ModelsGrid = new dax_dataGrid('Models',$('CarModelsDiv'), 0, 0, false);
   ModelsGrid.go();                        // go
   ModelsGrid.disableAutoRefresh();        // disable auto refreshing the grid
   ModelsGrid.setColumnWidth(0,110);       // Model Name
   ModelsGrid.hideColumn(1);               // model ID
   ModelsGrid.newQuery();                  //start a new query
   ModelsGrid.addQuery('brandID', '=', 0); // query to get 0 results
   ModelsGrid.runQuery();                  // run the query
```

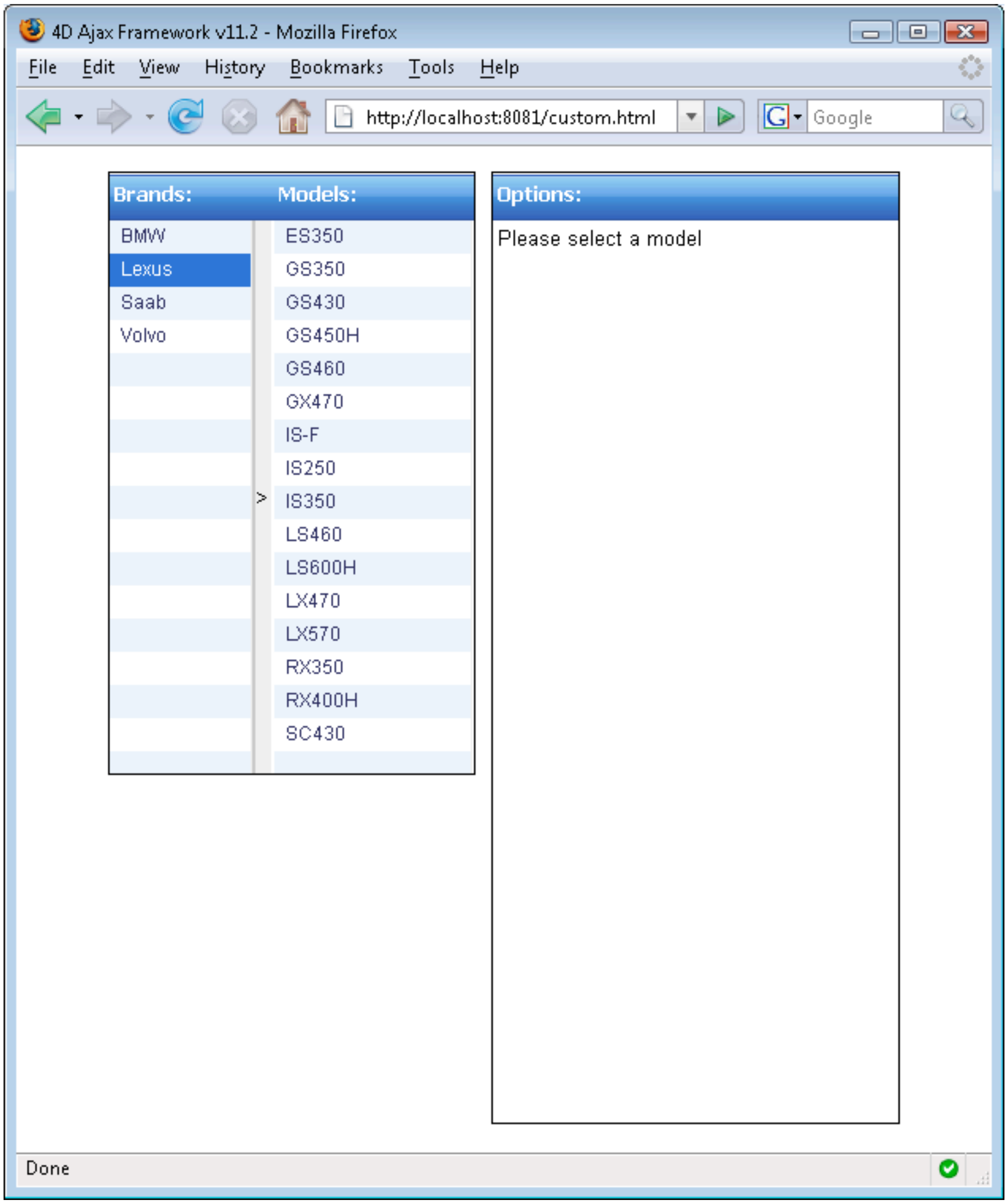
A query on the ModelsGrid is issued to query the database for models with brandID=0; since there are no models with brandID=0 the grid initially displays 0 results.

Each time the user clicks on a brand the BrandClicked() function is triggered:

```
function BrandClicked(row, column, recordId, fieldReference){
   $totalPrice = 0;
   $lastColorPrice = 0;
   $('CarOptionsDiv').innerHTML = "Please select a model";
   var BrandID = BrandsGrid.getCellValue(row, 0); // get Cell value from field 0
                                                  // of the selected row
   ModelsGrid.newQuery();                         // start a new query
   ModelsGrid.addQuery('brandID', '=', BrandID);  // Query based on BrandID
   ModelsGrid.runQuery();                         // run the query
   ModelsGrid.onDataRowClick = ModelClicked;      // set on clicked event
} // end BrandClicked
```

The BrandClicked() function first sets the value of both $totalPrice and $lastColorPrice to 0. Then the .innerHTML property of the CarOptionsDiv is set equal to "Please select a model". The variable BrandID is then set equal to the cell value from field 0 of the row that was clicked. A New query is then issued on the ModelsGrid object to query the database to display only the models whose brandID is equal to the currently selected brand. Lastly, an onDataRowClick event is registered for the ModelsGrid to trigger the ModelClicked() function each time a row is clicked.

At this point the Brands and Models columns are populated with data while the Options pane says "Please select a model".


4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Done

Brands:	Models:
BMW	ES350
Lexus	GS350
Saab	GS430
Volvo	GS450H
	GS460
	GX470
	IS-F
	IS250
	IS350
	LS460
	LS600H
	LX470
	LX570
	RX350
	RX400H
	SC430

Options:

Please select a model

Each time the user clicks on a row in the Models column, the ModelClicked() function will fire:

```
function ModelClicked(row, column, recordId, fieldReference){
    $totalPrice = 0;
    $lastColorPrice = 0;
    $('CarOptionsDiv').innerHTML = "Getting options from backend";
    var ModelID = ModelsGrid.getCellValue(row, 1); // get cell value from field 3
                                                   // of the selected row
    prepOptions(ModelID);
} //end ModelClicked
```

The ModelClicked() function first sets the value of both $totalPrice and $lastColorPrice to 0. Then the .innerHTML property of the CarOptionsDiv is set equal to “Getting options from backend”. The variable ModelID is then set equal to the cell value from field 1 of the row that was clicked. The ModelID variable is then passed along to the prepOptions() function:

```
function prepOptions(modelID){
            myQuery = new dax_query('Models');
            myQuery.clearCustomValues();
            myQuery.addCustomValue('modelID', modelID);
            myQuery.handler = getOptionValues;
            myQuery.runQuery();
    } // end prepOptions
```

The prepOptions() function builds a new query of the Models table to send a custom value to the 4D database backend. The .clearCustomValues function is issued to clear any values that may be left over from the last time this function was executed. Then the custom value is added to the query with name equal to the string “modelID” and value equal to the modelID of the currently selected Model. The getOptionValues() function is set as the handler for the query; this is the function that will be executed when data from this query comes back from 4D. The query is then executed with the .runQuery function and the custom values are sent to the 4D Database (backend).

The project method ***DAX_DevHook_OnQuery*** is executed in the 4D Database backend when the .runQuery command is executed. The developer modifications to ***DAX_DevHook_OnQuery*** have been simplified by adding the following method call at line 53:

```
myDevHook (->$queryDone_b)
```

Which will execute the myDevHook method and set the value of $queryDone_b.

The myDevHook method looks like:

Type variables used in our custom query:

```
  ` variable declarations for modelID custom value query
C_POINTER($1;$queryDone_bp)
C_TEXT($modelID_t;$colorsSpan_t)
C_LONGINT($modelCount_li;$i;$colorCount_li;$x)
C_REAL($basePrice_r)
  ` end variable declarations for modelID custom values query
```

Set the pointer $queryDone_bp to point to $1 ($queryDone_b of the calling method):

```
$queryDone_bp:=$1
```

Set the variable $modelID_t equal to the custom value identified with name “modelID” using the command ***DAX_Dev_GetWebVar***(“modelID”):

```
$modelID_t:=DAX_Dev_GetWebVar ("modelID")
```

If $modelID_t is not blank then handle the query and set the pointer $queryDone_bp to true:

```
If ($modelID_t#"")
   $queryDone_bp:=True
```

Query the [Options] table for the model selected:

```
   ALL RECORDS([Options])
   QUERY([Options];[Options]modelID=$modelID_t)
```

Count the options that were returned:

```
   $modelCount_li:=Records in selection([Options])
```

Type the arrays to have 2 elements more than the number of options for the model selected (1 extra for the base price, and 1 extra for the colors):

```
   ARRAY TEXT(customVarName_at;($modelCount_li+2))
   ARRAY TEXT(customVarValue_at;($modelCount_li+2))
```

Find the base price:

```
   ALL RECORDS([Models])
   QUERY([Models];[Models]ID=$modelID_t)
   $basePrice_r:=[Models]basePrice
```

Set the first custom name/value pair as the base price for the model:

```
   customVarName_at{1}:="Base Price"
   customVarValue_at{1}:=String($basePrice_r)
```

Loop through each option and set the custom name/value pair accordingly:

```
   FIRST RECORD([Options])
   For ($i;1;$modelCount_li)
          customVarName_at{($i+1)}:=[Options]name
          customVarValue_at{($i+1)}:=String([Options]price)
          NEXT RECORD([Options])
   End for
```

Query the [Colors] table for the colors available for the selected model:

```
     ` compose HTML for colors and send as name="Colors" and value="composedHTML"
   ALL RECORDS([Colors])
   QUERY([Colors];[Colors]modelID=$modelID_t)
```

Count the colors returned:

```
   $colorCount_li:=Records in selection([Colors])
   FIRST RECORD([Colors])
```

Start building the $colorsSpan_t variable. The contents of this variable will be sent to the web page (frontend) as the value in our last element's name/value pair. The contents of this variable is an HTML formatted table:

```
   $colorsSpan_t:="<table border=0 cellpadding=0 cellspacing=0 border=0 
width=100%>\r"
   $colorsSpan_t:=$colorsSpan_t+"<tr><td rowspan="+String($colorCount_li+1)+" 
valign=top align=left>Colors:</td></tr>"
```

Loop through each of the colors available:

```
    For ($x;1;$colorCount_li)
```

If the current color is White make the radio button selected by default using the *checked=true* attribute as well as identify this element with *ID=White*:

```
          If ([Colors]Color="White")
                  $colorsSpan_t:=$colorsSpan_t+"<tr><td align=left><input
type=\"radio\" id=\"White\" checked=true onclick=\"calcColor(this);\"
value=\""+String([Colors]Price)+"\" name=\"color\">"+[Colors]Color+"</td><td
align=right>$"+String([Colors]Price)+"</td></tr>\r"
```

All other colors do not have the *checked=true* attribute:

```
          Else
                  $colorsSpan_t:=$colorsSpan_t+"<tr><td align=left><input
type=\"radio\" onclick=\"calcColor(this);\" value=\""+String([Colors]Price)+"\"
name=\"color\">"+[Colors]Color+"</td><td
align=right>$"+String([Colors]Price)+"</td></tr>\r"
          End if
          NEXT RECORD([Colors])
    End for
```

End the HTML formatted table for the $colorsSpan_t variable:

```
    $colorsSpan_t:=$colorsSpan_t+"</table>"
```

Set the last element in the custom value/name paired arrays; the name is set as "Colors" and the value is set to the HTML formatted table contained within $colorsSpan_t

```
    customVarName_at{($modelCount_li+2)}:="Colors"
    customVarValue_at{($modelCount_li+2)}:=$colorsSpan_t
```

Send the custom values to the web page (frontend) using the DAX_Dev_SetCustomVariables project method making sure to pass customVarName_at and customVarValue_at as pointers

```
    DAX_Dev_SetCustomVariables (->customVarName_at;->customVarValue_at)
```

End the $modelID_t#"" if statement:

```
End if
```

When data comes back from 4D, the getOptionsValues() function is executed:

```
function getOptionValues(){
```

The myCustomValues variable is set to equal the custom values returned from the 4D Database (backend):

```
    myCustomValues = myQuery.getCustomValuesFrom4D();
```

Check the length of the myCustomValues variable, affectively counting the number of custom value/name pairs returned:

```
    var len = myCustomValues.length;
```

If length is not 0:

```
    if(len!=0){
```

Start building the block of HTML code that will display the options in the CarOptionsDiv div:

```
          $optionsHTML = "<form><table border=0 cellspacing=0 cellpadding=0 
width=100%>\r";
```

Loop through each of the custom values returned:

```
          for(var i = 0; i<len;i++){
```

If the current custom value is the base price format the cells accordingly:

```
                 if(myCustomValues[i].name == "Base Price"){
                        $basePrice=myCustomValues[i].value;
                        $optionsHTML = $optionsHTML + "<tr><td style=\"padding: 
3px\">Base Price:</td>\r<td align=right style=\"padding: 3px\">$"+ $basePrice 
+"</td></tr>\r";
```

Add a black line as a spacer to the table:

```
                        $optionsHTML = $optionsHTML + "<tr height=1 
bgcolor=#000000><td colspan=2></td></tr>\r";
```

All other custom values:

```
                 }else{ // end if / start else
                        $name = myCustomValues[i].name;
                        $value = myCustomValues[i].value;
```

If the current custom value name is Colors, set $spanForColors to the value returned from the database:

```
                        if($name=="Colors"){
                               $spanForColors = $value;
```

All other custom values are added to the table on their own row with a checkbox and a name. Each check box has a browser event registered to it to call the ***CalcTotal()*** function passing *this* as the parameter:

```
                        }else{ // end COLOR
                               $optionsHTML = $optionsHTML + "<tr>\r";
                               $optionsHTML = $optionsHTML + "<td><input 
type=checkbox value=\""+ $value +"\" ID=\"DaxCarOption\" name=\""+ $name +"\" 
onclick=\"calcTotal(this);\">"+ $name +"</td>\r";
                               $optionsHTML = $optionsHTML + "<td align=right 
style=\"padding: 3px\">$"+ $value +"</td>\r";
                               $optionsHTML = $optionsHTML + "</tr>\r";
                        } // end COLOR else
                 } // end BASE PRICE else
          } // end for loop
```

Add another black line as a spacer to the table:

```
          $optionsHTML = $optionsHTML + "<tr height=1 bgcolor=#000000><td 
colspan=2></td></tr>\r";
```

Add another row with a <span> inside of it that will be filled at the bottom of this function:

```
          $optionsHTML = $optionsHTML + "<tr><td valign=top colspan=2 
style=\"padding: 3px\"><span ID=\"ColorsDropDown\"></span></td></tr>\r";
```

Add another black line as a spacer to the table:

```
          $optionsHTML = $optionsHTML + "<tr height=1 bgcolor=#000000><td
colspan=2></td></tr>\r";
```

Add another row to the table, set the left cell to say Total Price (bold, right justified) and place a div inside the right cell identified with *ID=TotalPrice* to be modified later:

```
          $optionsHTML = $optionsHTML + "<tr><td><b style=\"padding: 3px\">Total
Price:</b></td>\r<td align=right style=\"padding: 3px\"><b><div
ID=\"TotalPrice\"></div></b></td></tr>\r";
```

End the table in the HTML formatted variable $optionsHTML

```
          $optionsHTML = $optionsHTML + "</table>\r</form>";
```

Set the CarOptionsDiv's .innerHTML property equal to the $optionsHTML that was just built:

```
          $('CarOptionsDiv').innerHTML = $optionsHTML;
```

Set the .innerHTML property of the span identified as *ID=ColorsDropDown* to the $spanForColors variable:

```
          $('ColorsDropDown').innerHTML = $spanForColors;
```

Set the $totalPrice variable equal to that of the $basePrice variable:

```
          $totalPrice = $basePrice;
```

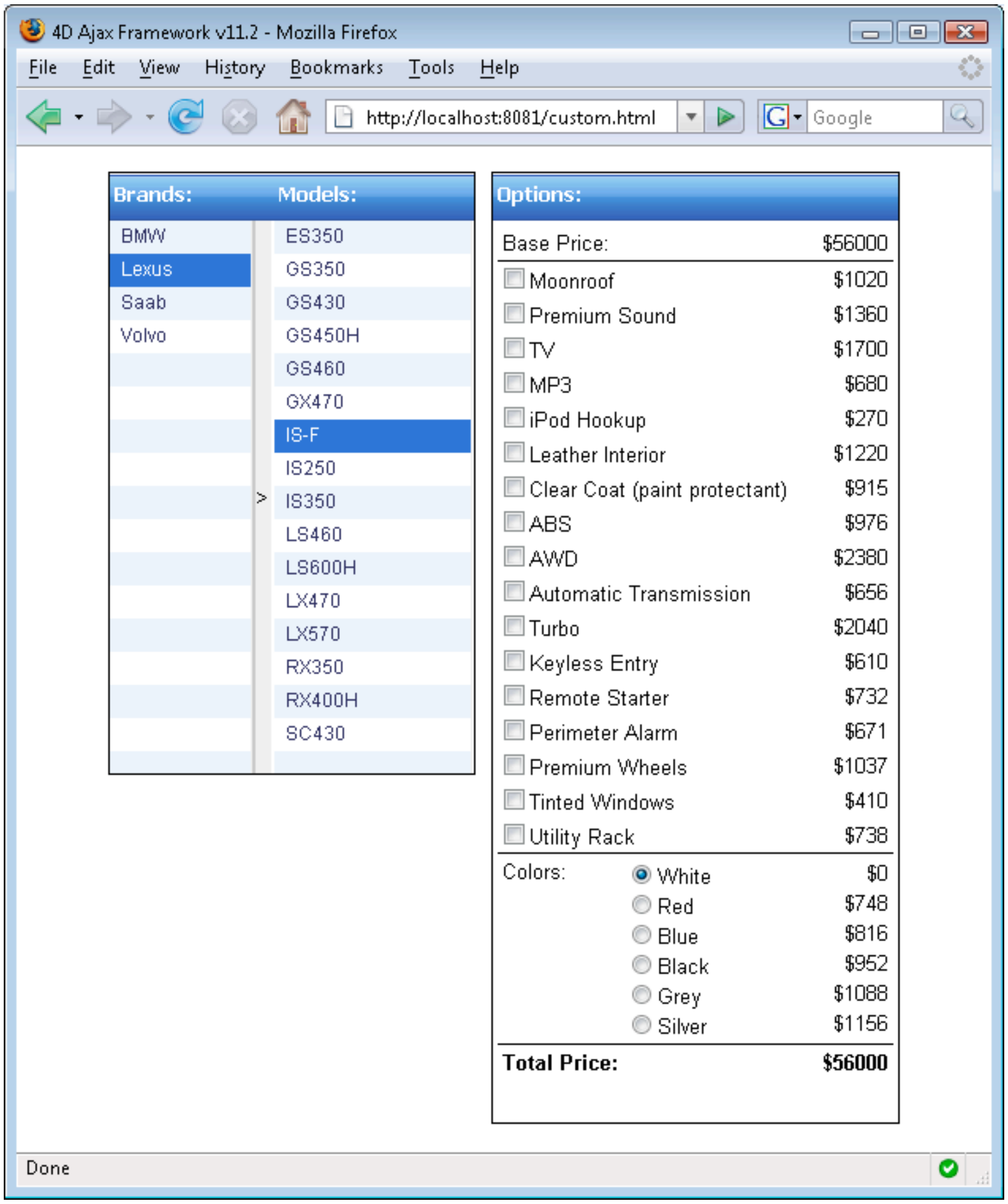
Set the .innerHTML property of the div identified by *ID=TotalPrice* equal to that of the $totalPrice variable:

```
          $('TotalPrice').innerHTML = "$"+$totalPrice;
```

Fire a click on the element identified as *ID=White*:

```
          $('White').click();
    } // end IF len # 0
} // end getOptionValues
```

At this point the user is presented with a list of available options and their respective prices like in the following screen shot:



4D Ajax Framework v11.2 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8081/custom.html
Google
Done

Brands:	Models:
BMW	ES350
Lexus	GS350
Saab	GS430
Volvo	GS450H
	GS460
	GX470
	IS-F
	IS250
	IS350
	LS460
	LS600H
	LX470
	LX570
	RX350
	RX400H
	SC430

Options:

Base Price:		$56000
☐ Moonroof		$1020
☐ Premium Sound		$1360
☐ TV		$1700
☐ MP3		$680
☐ iPod Hookup		$270
☐ Leather Interior		$1220
☐ Clear Coat (paint protectant)		$915
☐ ABS		$976
☐ AWD		$2380
☐ Automatic Transmission		$656
☐ Turbo		$2040
☐ Keyless Entry		$610
☐ Remote Starter		$732
☐ Perimeter Alarm		$671
☐ Premium Wheels		$1037
☐ Tinted Windows		$410
☐ Utility Rack		$738
Colors:	● White	$0
	○ Red	$748
	○ Blue	$816
	○ Black	$952
	○ Grey	$1088
	○ Silver	$1156
Total Price:		**$56000**

The ***CalcTotal()*** function is triggered each time the user clicks on any of the checkboxes:

```
function calcTotal(item){
```

The **CalcTotal()** function first checks to see if the checkbox has been checked or unchecked:

```
    if(item.checked==true){
```

If the checkbox has been checked the $totalPrice variable is increased by the amount associated with the checkbox:

```
        $totalPrice = parseInt($totalPrice) + parseInt(item.value);
```

Then the .innerHTML property of the div identified as *ID=TotalPrice* is modified to equal the new value of the *$totalPrice* variable:

```
        $('TotalPrice').innerHTML = "$"+$totalPrice;
    }else{
```

If the checkbox is unchecked, the value associated with the checkbox is subtracted from the *$totalPrice* variable

```
        $totalPrice = parseInt($totalPrice) - parseInt(item.value);
```

Then the .innerHTML property of the div identified as *ID=TotalPrice* is modified to equal the new value of the *$totalPrice* variable:

```
        $('TotalPrice').innerHTML = "$"+$totalPrice;
    }
}
```

The calcColor() function is triggered each time the user selects a different radio box associated with the colors available:

```
function calcColor(item){
```

Each time the function is called, the $totalPrice variable is increased by the value of the currently selected color and then also decreased by the value of $lastColorPrice variable:

```
  $totalPrice=parseInt($totalPrice)+parseInt(item.value)-parseInt($lastColorPrice);
```

Then the .innerHTML property of the div identified as *ID=TotalPrice* is modified to equal the new value of the *$totalPrice* variable:

```
  $('TotalPrice').innerHTML = "$"+$totalPrice;
```

The $lastColorPrice variable is then set to equal the value of the currently selected color; this is used for the next calculation:

```
  $lastColorPrice = parseInt(item.value);   // update the price of the last color
                                            // used for the next calculation
}
```

Conclusion

This Technical Note described the general concepts required to understand custom values in the 4D Ajax Framework. An example database and web page was presented. This information should allow the 4D Ajax Framework developers to write their own custom values.

A Note about 4D Web 2.0 Pack

The products in 4D Web 2.0 Pack are a departure from most other 4D products. As 4D Web 2.0 pack is a subscription-based product it is expected that incremental releases will be made. Thus, please note that this Technical Note is based on 4D Ajax Framework v11 release 2. As new features are implemented this Technical Note may become obsolete (faster than most other 4D products).

Related Resources

4D Ajax Framework Data Bridge 2.0:
http://daxipedia.4d.com/index.php/Bridge_2.0

Note: *Data Bridge 2.0 is still in beta therefore information in the link provided is subject to change when released.*

For the latest information on the 4DAF consult the latest documentation and also check the 4D Web 2.0 Pack Wiki:
http://daxipedia.4d.com

For the latest news about 4D Web 2.0 Pack or to find out how to purchase it see:
http://www.4d.com/products/4dweb20pack.html