Introduction

Purpose of Document

This document provides a technique to auto-cascade single or multi select value prompts without refreshing the page or re-running the prompt queries.

Applicability

The steps in this document were validated using:

- IBM Cognos 10.2.1.3 using the Go Sales(query) sample package
  - Microsoft Internet Explorer 8
  - Microsoft Internet Explorer 10
- IBM Cognos 10.2.2 using the Go Sales(query) sample package
  - Microsoft Internet Explorer 11

Exclusions and Exceptions

This technique and its associated JavaScript will only work using the Microsoft Internet Explorer versions listed within the applicability section of this document. The technique will not work using any other browser such as Firefox or Chrome.
Caveats
This technique requires the use of undocumented and unsupported capabilities in IBM Cognos Business Intelligence (BI). There may be a risk associated with this technique in that support for these capabilities may change or be dropped entirely in some future releases. This technique should be thoroughly tested in a non production environment.

Auto-Cascading Prompt Overview
Cascading prompts are typically used to improve the user experience when dealing with related data items consumed by the prompt page. The cascade functionality limits the displayed prompt values based on the user's selection within the parent prompt. This built-in functionality causes the entire prompt page to be re-submitted resulting in a possible re-run of the supporting data queries and additional time to re-render the prompt page. The approach in this document queries all the data at once and performs the prompt refresh without the requirement of re-running the queries or re-rendering the prompt page.

Steps to Implement Auto Cascading without Page Refresh
1. Within IBM Cognos Report Studio, create a Query containing the use and display values of all prompts. For this example the query will contain the following query items obtained from the Go Sales(query) package:
   - [Sales(query)].[Products].[Product line]
   - [Sales(query)].[Products].[Product line code]
   - [Sales(query)].[Products].[Product type]
   - [Sales(query)].[Products].[Product type code]
   - [Sales(query)].[Products].[Product]
   - [Sales(query)].[Products].[Product number]
2. Create a prompt page and insert two blocks by dragging them from the available Toolbox items onto the Prompt Page canvas.
3. From the available Toolbox items, insert three Value Prompt objects into the first of the two previously inserted blocks. As each of the Value Prompt objects are dragged onto the canvas, IBM Cognos Report Studio will display the Prompt-Wizard-Value Prompt dialog box. Within that dialog box specify the parameter name for each of the three parameters and click the Finish button. For this example the parameter names will be Product line, Product type and Product. When completed the Prompt Page canvas should have two blocks stacked on top of each other, with three Value Prompt objects contained within the first block. This is illustrated in Figure 1.

Figure 1 - IBM Cognos Report Studio Prompt Page canvas displaying three Value Prompt objects
4. Highlight the first Value Prompt object on the Prompt Page canvas and within the bottom left Properties pane set the following properties:
   • Data > Query = Query1
   • Data > Use Value = Product line code
   • Data > Display Value = Product line
   • Miscellaneous > Name = Product line

When completed, the properties of the first Value Prompt object should appear as shown in Figure 2.

**Figure 2 - Properties of the first Value Prompt object**

5. Repeat Step 4 for each of the other two Value Prompt objects.
   For the second Value Prompt,
   • Data > Query = Query1
   • Data > Use Value = Product type code
   • Data > Display Value = Product type
   • Miscellaneous > Name = Product type

   For the third Value Prompt,
   • Data > Query = Query1
   • Data > Use Value = Product number
   • Data > Display Value = Product
   • Miscellaneous > Name = Product

6. For each of the Value Prompt objects, set whether or not it will be single-select or multi-select by setting the **Multi-Select** property within the bottom left Properties pane to **Yes** or **No** as desired. For this example, the first Value Prompt object will remain single-select, while the last two will become multi-select. This is illustrated in Figure 3.
7. Within the Prompt Page canvas, highlight the bottom block object and within the bottom left Properties pane set the Name property to `cascadeLists`.

8. From the available Toolbox objects, drag a List object into the bottom block object. When presented with the Object and Query Name dialog box, enter `List1` for the Name and select `Query1` from the Query Name dropdown box.

9. With the List object selected on the Prompt Page canvas, in the bottom left Properties pane set the Data > Rows per Page property to `9999999`. Figure 4 shows the completed Properties pane for the List object with the Query property set to `Query1`, the Rows Per Page property set to `9999999` and the Name property set to `List1`.

**Figure 3 - Prompt Page canvas with the Value Prompt object multi-select attribute set**

**Figure 4 – Completed Properties pane for the List object**
10. From the top left Data Items tab, select the query items in the order of the display and use value for each parameter, from left to right. For this example, the query items should be inserted into the List object in the following order:
   - Product line
   - Product line code
   - Product type
   - Product type code
   - Product
   - Product number

11. Click on each of the List columns and within the bottom left Properties pane, change the Data Item > Label property to match the corresponding Value Prompt object name that was set earlier. In this example, the completed List object should have three sets of two columns where each set has the same name as the Value Prompt object. This is illustrated in Figure 5.

   **Figure 5 - List Object with renamed column names to match the parameter names**

12. From the available Toolbox pane, locate the HTML Item object and drag it onto the Prompt Page canvas after the List object.

13. Copy and paste the following JavaScript into the HTML Item object in order to implement the scripting logic for the cascade and hide the Select All and Deselect All adornments.

   ```javascript
   var fW = (typeof getFormWarpRequest == "function" ? getFormWarpRequest() : document.forms["formWarpRequest"]);
   if ( !fW || fW == undefined)
   {
   fW = ( formWarpRequest_THIS_ ? formWarpRequest_THIS_ : formWarpRequest_NS_ );
   }
   var isDisableOn=true
   //change this to false if you don't wish grandchild or younger prompts to be disabled
   /*Section 1: initialize cascade:
   this section handles grabbing the block with the list(s) in it, verifying the lists were constructed as necessary, and adding an onchange functionality calling the cascade function in the second section to each relevant prompt directing it at the correct list.
   */
   /*get the block:
   To simplify matters and make the whole thing operate from a single HTML Item, the block is captured by a Cognos generated LID.
   */
   var allDivs=document.getElementsByTagName("div");
   var theDiv;
   for(var allDivsIndex=0;allDivsIndex<allDivs.length;allDivsIndex++){
   if(allDivs[allDivsIndex].LID&allDivs[allDivsIndex].LID.indexOf("cascadeLists")===0){
   ```
theDiv=allDivs[allDivsIndex];
theDiv.style.display="none"; //hide it from the world
break;
}
}

/*get the list(s)
The solution supports multiple cascade instances, so there can be more than one list.
Here is where we grab them */
var allLists=theDiv.getElementsByTagName("table");
var theLists=new Array();
for(var allListsIndex=0;allListsIndex<allLists.length;allListsIndex++)
{
    if(allLists[allListsIndex].className&&allLists[allListsIndex].className=='ls')
    {
        theLists.push(allLists[allListsIndex]);
    }
}
var numberOfCascades=theLists.length;

/*verify the lists -
there are three verifications made:
1. The list has an even number of columns (use, display)
2. The header of every pair is the same
3. There are prompts on page with the same name as said header for all three. If a verification step fails, an alert is thrown to alert the user/developer */
function verifyTheLists(){
for(var theListsIndex=0;theListsIndex<numberOfCascades;theListsIndex++){
    var theList=theLists[theListsIndex];
    var isListValid=true;
    var allRows=theList.getElementsByTagName("tr");
    var titleRow=allRows[0];
    var allHeaders=titleRow.getElementsByTagName("td");
    //verification number 1
    if(allHeaders.length%2!=0){
        alert("Each prompt should have a list column with use and display value");
        isListValid=false;
    }
    //verification number 2+3
    for(var allheadersIndex=0;allheadersIndex<allHeaders.length-1;allheadersIndex++){
        if(allheadersIndex%2==0){//even columns
            if(allHeaders[allheadersIndex].innerText!=allHeaders[allheadersIndex+1].innerText){
                isListValid=false;
                alert("each pair needs an identical label");
                break;
            }
        }
        else{//odd column
            if(!fW["_oLstChoices"+allHeaders[allheadersIndex].innerText]){ 
                alert("There is no corresponding prompt for "+allHeaders[allheadersIndex].innerText);
                isListValid=false;
                break;
            }
        }
    }
}
else{//odd column
    if(allHeaders[allheadersIndex].innerText==allHeaders[allheadersIndex+1].innerText){
        isListValid=false;
    }
}
alert("each pair needs an identical label in the list.");
break;
}

if(!isListValid){break;}
}
return isListValid;

/*the Initialize sequence is quite simple:
We go over each list, verify it, and then add the corresponding prompt from each header
an onchange function
*/
function initPromptsCascade(){
    if(verifyTheLists()){  
        for(var theListsIndex=0;theListsIndex<numberOfCascades;theListsIndex++){ 
            var theList=theLists[theListsIndex];
            var theID=theList.LID;
            var allRows=theList.getElementsByTagName("tr");
            var titleRow=allRows[0];
            var allHeaders=titleRow.getElementsByTagName("td");

            for(var titleIndex=0;titleIndex<allHeaders.length;titleIndex+=2){ 
                //remove select/deselect all links because they don't work with the cascade
                var thePrLinks=fW["_oLstChoices"+allHeaders[titleIndex].innerText].parentNode.parentNode.getElementsByTagName("a");
                for(var linksI=0;linksI<thePrLinks.length;linksI++){
                    thePrLinks[linksI].style.display="none";
                }

                if(titleIndex<allHeaders.length-2){
                    //we want to disable grandchildren?
                    fW["_oLstChoices"+allHeaders[titleIndex].innerText].isDisableOn=isDisableOn;
                    fW["_oLstChoices"+allHeaders[titleIndex].innerText].ListID=theID;

                    //do we want to disable grandchildren?
                    fW["_oLstChoices"+allHeaders[titleIndex].innerText].isDisableOn=isDisableOn;
                    fW["_oLstChoices"+allHeaders[titleIndex].innerText].location=titleIndex;
                    fW["_oLstChoices"+allHeaders[titleIndex].innerText].onchange=function()
                    {   
                        cascadePrompts(this.ListID,this.location,this.isDisableOn);
                    }
                    //disable 3rd generation onwards
                    if(titleIndex>0&&isDisableOn){
                        fW["_oLstChoices"+allHeaders[titleIndex].innerText].disabled=true;
                    }
                }
            }
        }
    }

    /*section 2: perform cascade:
This is where the magic happens. When a user selected a value in a prompt, we:
1. create an array of the selected values.
2. clear all descendant prompts
3. go to the list at the prompt's location
4. search through the list for the selected values
5. if the value on the list row matches the selected value, we add the values of the
descendants to the descendants prompts.
6. if the flag is on, disable 3rd generation descendants or younger
*/
    function getSelectedValues(promptName){
        //This could be made MUCH simpler if using the new 10.2+ prompts API
        var pr=fW["_oLstChoices"+promptName];
        var retArr=new Array();
        for(var prI=0;prI<pr.length;prI++){
            if(pr[prI].selected==true){
                retArr.push(pr[prI].dv)
            }
        }
        //we compare based on display value, just in case we didn't use the correct code
} return(retArr); }

function clearDescendants(header,location){
 for(var childrenI=location+2;childrenI<header.length;childrenI+=2){
 fw["_oLstChoices"+header[childrenI].innerText].length=0; 
 }
 }

function checkUnique(pr,display){
 //we have to do it this way in case the same child is in two different parents
 var isUnique=true;
 for(var prI=0;prI<pr.length;prI++){
   if(pr[prI].dv==display){
     isUnique=false;
     break;
   }
 }
 return isUnique;
 }

function addValueToPrompt(promptName,display,use){
 var pr=fW["_oLstChoices"+promptName];
 var isUnique=true;
 //check for uniqueness
 if(pr.length>0){ //not empty
   isUnique=checkUnique(pr,display);
 }
 if(isUnique){
   var opt=document.createElement('option');
   opt.text=opt.dv=display;
   opt.value=use;
   pr.add(opt);
 }
 }

function cascadePrompts(list,location,isDisable){
 for(var listsI=0;listsI<theLists.length;listsI++){
   if(theLists[listsI].LID==list){
     var listEl=theLists[listsI];
     break;
   }
 }
 var rows=listEl.getElementsByTagName("tr");
 var headerRow=rows[0];
 var headerRowCells=headerRow.getElementsByTagName("td");
 var currentPrompt=headerRowCells[location].innerText;
 //clear children prompts
 clearDescendants(headerRowCells,location);
 var theSelections=getSelectedValues(currentPrompt);
 var isEmpty=theSelections.length==0;
 //run through the list
 for(var rowsI=1;rowsI<rows.length;rowsI++){
   var cells=rows[rowsI].getElementsByTagName("td");
   for(var selectionsI=isEmpty?-1:0;selectionsI<theSelections.length;selectionsI++){
     //run through selected Values
     if(cells[location].innerText==theSelections[selectionsI]||isEmpty){
       //the value on the cascading prompt matches!
       for(var cellsI=location+2;cellsI<cells.length;cellsI+=2) {
         
       }
     }
   }
 }

var thePr=headerRowCells[cellsI].innerText;
var dispVal=cells[cellsI].innerText;
var useVal=cells[cellsI+1].innerText;
addValueToPrompt(thePr,dispVal,useVal);
}
}
//disable 3rd generation onwards
if(isDisable){
if(!isEmpty){//enable son
fW['_oLstChoices'+headerRowCells[location+2].innerText].disabled=false;
//disable grandchild and onwards
for(var disableI=location+4;disableI<headerRowCells.length;disableI+=2){
  fW['_oLstChoices'+headerRowCells[disableI].innerText].disabled=true;
}
}
else{
  fW['_oLstChoices'+headerRowCells[location].innerText].disabled=false;
//disable grandchild and onwards
for(var disableI=location+2;disableI<headerRowCells.length;disableI+=2){
  fW['_oLstChoices'+headerRowCells[disableI].innerText].disabled=true;
}
}
}
initPromptsCascade();
</script>

14. Click the **OK** button to return to the Report Studio canvas.
15. From the available **Run** menu, select the **Run Report – HTML** to run the report in the IBM Cognos Report Viewer.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AutoCascading.zip</td>
<td>5KB</td>
</tr>
</tbody>
</table>
About the author

Rod Avissar

Nimrod (Rod) Avissar is a Cognos consultant with over 8 years of experience and a penchant for specialized UX solutions. He developed and maintained Cognos reports, models and specialized solutions and administered Cognos environments for various customers from all industries.

© Copyright IBM Corporation 2014, 2015
Trademarks
(www.ibm.com/developerworks.ibm/trademarks/)