Customize the Look and Feel of BW Reports in SAP BusinessObjects Analysis (Office) by Using API Calls to Expand the Functionality and User Experience

Session #0308

Sara Langenfeld and Sarah Klobe
Introduction to Analysis for Office at General Mills

Analysis for Office Introduction

Solving Reporting Challenges with the Analysis API

Guidelines for Working with the Analysis API
General Mills implemented a SAP BW solution to support reporting requirements for analyzing coupon data.

After previous attempts to create complex customized Excel reports using other reporting technologies, SAP Business Objects Analysis Edition for Office was implemented.

SAP Business Objects Analysis Edition for Office was able to successfully fulfill all user requirements and allow the project to be successfully completed.
Overview: Analysis Edition for Office

- **Advantages**
  - Familiar MS Office functionality
  - Utilize existing BEx queries
  - Easily filter and sort
  - Apply conditional formatting

- **Data Source** – BEx Query

Follow @ASUG365 and #ASUG2013 on Twitter
“As a marketing user I need to apply very specific formatting to my reports”

Challenge: Excel formatting of crosstab data does not persist across data refreshes

Value Add – Maintaining consistency across Marketing reports is required to reduce training costs and increase productivity
Example of Multiple Source Report

Data From SAP BW

User Entered Data
Example of Multiple Source Report

Data from BW and non-BW data sources combined on new worksheet

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
<td><strong>C</strong></td>
<td><strong>D</strong></td>
<td><strong>E</strong></td>
<td><strong>F</strong></td>
<td><strong>G</strong></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Marketing Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4/9/2012</td>
<td></td>
<td>Key Date 1</td>
<td>4/16/12</td>
<td>Key Date 2</td>
<td>4/23/12</td>
</tr>
<tr>
<td>3</td>
<td>8:46:02 AM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>Project Name</td>
<td>Start Date</td>
<td>Status</td>
<td>Parent Program</td>
</tr>
<tr>
<td>5</td>
<td>Project 1</td>
<td>08/01/2006</td>
<td>Planning</td>
<td>Project 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Project 2</td>
<td>12/09/2007</td>
<td>Planning</td>
<td>Project 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Project 3</td>
<td>01/01/2010</td>
<td>Planning</td>
<td>Project 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Project 4</td>
<td>10/01/2010</td>
<td>Planning</td>
<td>Project 4</td>
<td></td>
<td>Not assigned</td>
</tr>
<tr>
<td>9</td>
<td>Project 5</td>
<td>10/01/2010</td>
<td>Planning</td>
<td>Project 5</td>
<td></td>
<td>Not assigned</td>
</tr>
</tbody>
</table>

Follow @ASUG365 and #ASUG2013 on Twitter
Creating formulas referencing an Analysis based datasheet allow custom formatting to be maintained across data refreshes.

Use Named Ranges when referencing custom data sources to improve maintainability of reports.
The Analysis for Office API includes the “Register Callback” method that invokes a “Callback_AfterREDISPLAY” event each time the underlying SAP data source is refreshed.
Example of Fill Down Routine

Function FillDownFormulas(sourceSheetName As String, destSheetName As String, _
    startC As String, endC As String) As Integer
    Dim WSD As Worksheet
    Set WSD = Worksheets(sheetName)
    Dim WSS As Worksheet
    Set WSS = Worksheets(sourceSheetName)
    ' Find the last row with data in the source datasheet by checking column 1
    Dim finalRow As Long
    finalRow = WSS.Cells(Application.Rows.Count, 1).End(xlUp).Row
    If finalRow <= 2 Then Exit Function
    ' Fill formulas down
    Dim sourceRange As Range
    Set sourceRange = WSD.Range(WSD.Cells(2, startC), WSD.Cells(2, endC))
    Dim fillRange As Range
    Set fillRange = WSD.Range(WSD.Cells(2, startC), WSD.Cells(finalRow, endC))
    sourceRange.AutoFill Destination:=fillRange, Type:=xlFillDefault
    FillDownFormulas = finalRow
    End Function

Function RemoveFillDownFormulas(sheetName As String, startC As String, endC As String) As Boolean
    Dim WSD As Worksheet
    Set WSD = Worksheets(sheetName)
    ' Find the last row with data
    Dim finalRow As Long
    finalRow = WSD.Cells(Application.Rows.Count, 3).End(xlUp).Row
    If finalRow <= 3 Then Exit Function
    ' Fill formulas down
    Dim fillRange As Range
    Set fillRange = WSD.Range(WSD.Cells(1, startC), WSD.Cells(9999, endC))
    fillRange.Clear
    End Function

Determine the number of rows in the SAP BW based data sheet and copy the formula down accordingly

Before performing the copy remove the existing rows. Choose a number of rows to delete that is larger than the possible number of rows in the report.
“As a marketing user I need to be able to add custom report navigation to my reports”

Challenge: The “Display” tab in Analysis Edition for Office does not allow customization for advanced navigation options such as column groups.

Value Add: Expanded navigational capabilities are required in order to roll out marketing reports to a broad audience.
Example of a Marketing report that contains a very large number of columns. The report required the ability to hide and show columns in groups.

To assign a VBA method to the button, right click on the button and select “Assign Macro”. This method is where the Analysis Edition for Office API will be invoked.
Public FirstRefresh As Boolean

Public Sub Callback_AfterRedisplay()

Worksheets("Data").Activate

'Check the IsConnected property to determine if the datasource
'has been refreshed
Iret = Application.Run("SAPGetProperty", "IsConnected", "DS_1")

If Iret = True Then

'The datasource is connected so the refresh event has fired at least once
If FirstRefresh = True Then

'This is the first time the code has executed
Call PerformInitializationActivities

FirstRefresh = False
Exit Sub

End If

'Add Specialized Formatting to the columns
Call FormatSections

End Sub
The standard Excel API contains a variety of functions that allow control of the displayed format of the report.

```vba
Sub FormatSections()
    Dim reportlastrow As Integer
    Dim Formatcolumn As Integer

    'Define the columns that define each group
    Call FillArray

    'Determine the checkbox setting on the form
    IfWorksheets("Data").Shapes("Check Box 1").DrawingObject.Value = xlOn Then
        On Error Resume Next

    'Show all columns in the group
    For Each PlanField In PlanDataArray()
        Err.Clear
        ThisWorkbook.Worksheets("Data").Rows(5).Select
        SResults = Selection.Find(What:=PlanField, After:=ActiveCell, LookIn:=xlFormulas, _
                                   LookAt:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
                                   MatchCase:=False, SearchFormat:=False).Activate

        'Set the default style for the columns
        Formatcolumn = ActiveCell.Column
        With Worksheets("Data").Range(Cells(5, Formatcolumn), Cells(reportlastrow, Formatcolumn))
            .Interior.Color = RGB(155, 255, 200)
            .Font.Bold = True
        End With
    Next PlanField
End If

Sub FillArray()
    'Define an array of columns that define group 1
    ReDim Group1Array(3)
    Group1Array(0) = "Column 1"
    Group1Array(1) = "Column 2"
    Group1Array(2) = "Column 3"
    Group1Array(3) = "Column 4"
End Sub
```
Sub ShowGroup1_Click()

' The form checkbox is unchecked
If Worksheets("Data").Shapes("Check Box 2").DrawingObject.Value = xlOff Then
    On Error Resume Next
    'Hide each column in the Group
    For Each PlanField In PlanDataArray()
        Err.Clear
        Rows(5).Select
        SResults = Selection.Find(What:=PlanField, After:=ActiveCell, LookIn:=xlFormulas, _
                            LookAt:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
                            MatchCase:=False, SearchFormat:=False).Activate
        Columns(ActiveCell.Column).EntireColumn.Hidden = True
        Next PlanField
    Else
    ' The form checkbox is checked
    On Error Resume Next
    'Show each column in the group
    For Each PlanField In PlanDataArray()
        Err.Clear
        Rows(5).Select
        SResults = Selection.Find(What:=PlanField, After:=ActiveCell, LookIn:=xlFormulas, _
                            LookAt:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, _
                            MatchCase:=False, SearchFormat:=False).Activate
        Columns(ActiveCell.Column).EntireColumn.Hidden = False
        Next PlanField
    End If
End Sub
“As a finance user I need to create a pivot table on the warehouse data regardless of whether it is a key figure or a characteristic”

Challenge: Aggregation within Analysis for Office is only possible on key figures

Value Add: Expanded range of pivot table options is necessary for Finance community to perform required analysis tasks
Customer report that displays sales data according to a rating system based on closed sales.

The report contains a pivot table that displays the total units and dollars aggregated by “Month” and “Rating Category”

Aggregation within an Analysis Edition for Office crosstab is limited to Key Figures only. Replicating this pivot table requires combining an Analysis driven crosstab with Excel pivot functionality.
Interacting with the API in Action!

Demo
“As a finance user I need to be able to consolidate Excel reports for my month end reporting activities.”

Challenge: Activities that involve complex process models can be difficult to replicate in an Analysis workbook.

Value Add: Consolidating reports programmatically is necessary to support business continuity as new employees are brought in to the organization.
The Callback_AfterRefresh event may not be appropriate for all automation activities. The event is fired after several different actions such as viewing the Display panel. For long running automation activities, a more manual approach is necessary.

By using the Design Form Controls within Excel, it is possible to programmatically interact with the Analysis for Office API and take a greater deal of control over generated reports.
To assign a VBA method to the button, right click on the button and select “Assign Macro”. This method is where the Analysis for Office API will be invoked.
Configuring the Excel Display properties will prevent screen flickering while reports are being generated and provide a much better end user experience.

```vba
Sub GenerateReport_Click()
    Call SetWorkbookSettingsForLoad
    Call BuildReport
    Call SetWorkbookSettingsForDisplay
End Sub
```

```vba
Sub SetWorkbookSettingsForLoad()
    ' disable events, alerts, automatic calculation & screen updating
    With Application
        .EnableEvents = False
        .DisplayAlerts = False
        .Calculation = xlCalculationManual
        .ScreenUpdating = False
    End With
End Sub
```

```vba
Sub SetWorkbookSettingsForDisplay()
    ' disable events, alerts, automatic calculation & screen updating
    With Application
        .EnableEvents = True
        .DisplayAlerts = True
        .Calculation = xlCalculationAutomatic
        .ScreenUpdating = True
    End With
End Sub
```
Sub BuildReport()

' Clear data from all cells

' Turn the Pause/Refresh functionality Off
lResult = Application.Run("SAPSetRefreshBehaviour", "Off")

Dim snapshotPeriod As String
Dim fiscalYearWeek As String
Dim startDropPeriod As String

snapshotPeriod = Worksheets("Report Options").Range("B3:B3").Value
fiscalYearWeek = Worksheets("Report Options").Range("B11:B11").Value
startDropPeriod = Worksheets("Report Options").Range("B14:B14").Value

' Populate all of the start and end date variables on the prompt screen
' from the selected values on the worksheet
selectedType = "INPUT_STRING"

dimension = "ZC_CPNFPMR"
c = Application.Run("SAPSetVariable", dimension, snapshotPeriod, selectedType)
dimension = "ZS_FYW03"
c = Application.Run("SAPSetVariable", dimension, fiscalYearWeek, selectedType)
dimension = "ZS_FISCALYEAR"
c = Application.Run("SAPSetVariable", dimension, startDropPeriod, selectedType)

' Set the refresh behavior on to force the data to refresh
lResult = Application.Run("SAPSetRefreshBehaviour", "On")
lResult = Application.Run("SAPSetRefreshBehaviour", "Off")

End Sub
Percentage of reports in General Mills SAP BW initiative by technique used:

- **Standard Analysis Reports**: 50%
- **Analysis Driven Reports**: 30%
- **Excel Driven Reports**: 20%
Analysis Driven Reports

Use the Callback_AfterRefresh event when report modifications do not require a lot of time to complete.

When combining Analysis Crosstab data with Excel formulas, utilize VBA routines to fill down formulas to the appropriate number of rows.

Refer to the SAPCrosstab name generated by Analysis for Office when creating formulas referencing crosstab data.

To ensure that code only executes one time, use the “IsConnected” property.
Excel Driven Reports

If the default CallBack_AfterRefresh event does not provide enough flexibility for a report, consider launching the report via an Excel form control.

Utilize the Pause/Refresh functionality of Analysis for Office by setting the “SAPSetRefreshBehavior” property before refreshing a query.

Use the “SAPSetVariable” property to add flexibility to the setting of prompt variable values

Set up prompt variables that return little or no data initially to speed up report load times

Clear out all data on Excel Worksheets before reloading Analysis for Office crosstab data to reduce conflict errors.
Follow @ASUG365 and ASUG CEO Bridgette Chambers @BChambersASUG on Twitter to keep up to date with everything at ASUG.

Follow the ASUGNews team of Tom Wailgum: @twailgum and Courtney Bjorlin: @cbjorlin for all things SAP.
THANK YOU FOR PARTICIPATING

Please provide feedback on this session by completing a short survey via the event mobile application.

SESSION CODE: 0308

For ongoing education on this area of focus, visit www.ASUG.com