Create Customized Navigation Patterns

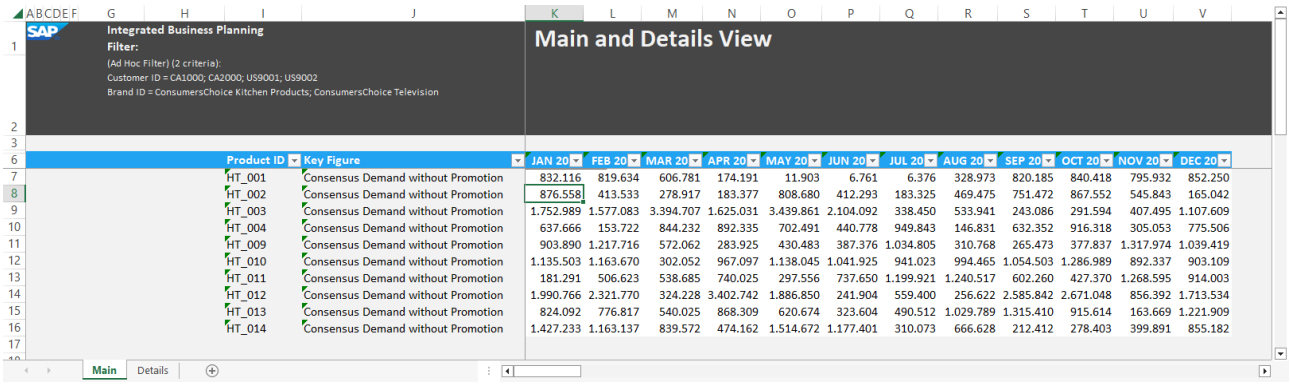
In this use case, we explain how navigation patterns between worksheets in a planning view template can be set up. In the first example, we explain how the navigation from a main planning view worksheet to a more detailed planning view worksheet by double clicking a cell can be set up. The second example we are providing for this use case is to navigate from a main planning view worksheet to different key figure-specific worksheets by double clicking the key figure.

1. MAIN AND DETAILS VIEW (EXAMPLE 1)

1.1. How to start

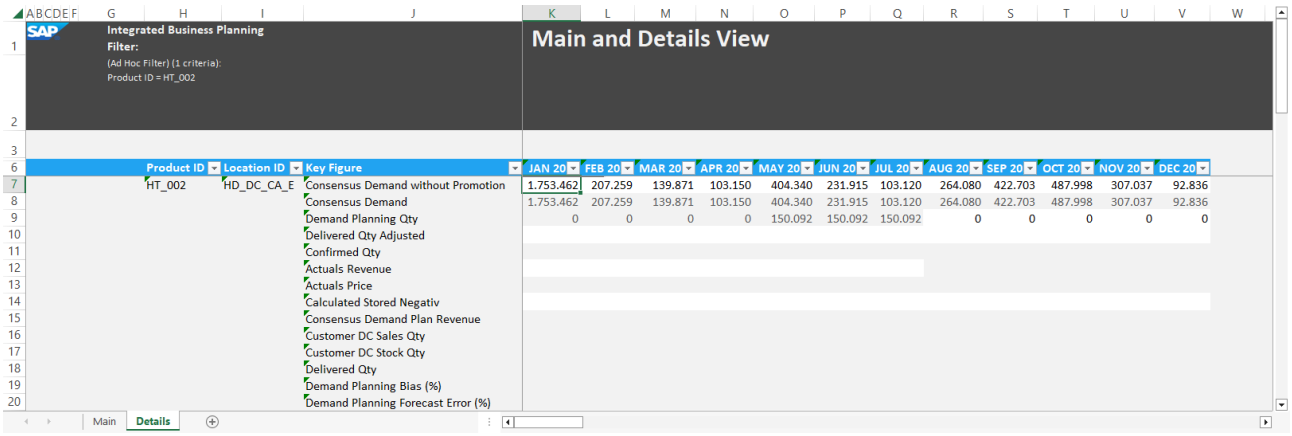
As a first step, the planning views for which you want to create navigation patterns need to be created.

In the first worksheet, create the main planning view with the key figure that is relevant for planning, at the planning level you want to see. In our example, we are using the key figure “Consensus Demand without Promotion” at Product ID level.



Product ID	Key Figure	JAN 20	FEB 20	MAR 20	APR 20	MAY 20	JUN 20	JUL 20	AUG 20	SEP 20	OCT 20	NOV 20	DEC 20
HT_001	Consensus Demand without Promotion	832.116	819.634	606.781	174.191	11.903	6.761	6.376	328.973	820.185	840.418	795.932	852.250
HT_002	Consensus Demand without Promotion	876.558	413.533	278.917	183.377	808.680	412.293	183.325	469.475	751.472	867.552	545.843	165.042
HT_003	Consensus Demand without Promotion	1.752.989	1.577.083	3.394.707	1.625.031	3.439.861	2.104.092	338.450	533.941	243.086	291.594	407.495	1.107.609
HT_004	Consensus Demand without Promotion	637.666	153.722	844.232	892.335	702.491	440.778	949.843	146.831	632.352	916.318	305.053	775.506
HT_009	Consensus Demand without Promotion	903.890	1.217.716	572.062	283.925	430.483	387.376	1.034.805	310.768	265.473	377.837	1.317.974	1.039.419
HT_010	Consensus Demand without Promotion	1.135.503	1.163.670	302.052	967.097	1.138.045	1.041.925	941.023	994.465	1.054.503	1.286.989	892.337	903.109
HT_011	Consensus Demand without Promotion	181.291	506.623	538.685	740.025	297.556	737.650	1.199.921	1.240.517	602.260	427.370	1.268.595	914.003
HT_012	Consensus Demand without Promotion	1.990.766	2.321.770	324.228	3.402.742	1.886.850	241.904	559.400	256.622	2.585.842	2.671.048	856.392	1.713.534
HT_013	Consensus Demand without Promotion	824.092	776.817	540.025	868.309	620.674	323.604	490.512	1.029.789	1.315.410	915.614	163.669	1.221.909
HT_014	Consensus Demand without Promotion	1.427.233	1.163.137	839.572	474.162	1.514.672	1.177.401	310.073	666.628	212.412	278.403	399.891	855.182

In the second worksheet, create a second planning view that contains all the key figures which might be needed to understand the main key figure. In this example, the second planning view is created at a different planning level (Product ID and Location ID).



Product ID	Location ID	Key Figure	JAN 20	FEB 20	MAR 20	APR 20	MAY 20	JUN 20	JUL 20	AUG 20	SEP 20	OCT 20	NOV 20	DEC 20
HT_002	HD_DC_CA_E	Consensus Demand without Promotion	1.753.462	207.259	139.871	103.150	404.340	231.915	103.120	264.080	422.703	487.998	307.037	92.836
		Consensus Demand	1.753.462	207.259	139.871	103.150	404.340	231.915	103.120	264.080	422.703	487.998	307.037	92.836
		Demand Planning Qty	0	0	0	0	150.092	150.092	150.092	0	0	0	0	0
		Delivered Qty Adjusted												
		Confirmed Qty												
		Actuals Revenue												
		Actuals Price												
		Calculated Stored Negativ												
		Consensus Demand Plan Revenue												
		Customer DC Sales Qty												
		Customer DC Stock Qty												
		Delivered Qty												
		Demand Planning Bias (%)												
		Demand Planning Forecast Error (%)												

After the preparation is done, we build the connection from the “Main” worksheet to the “Details” worksheet. The trigger for the navigation is a double-click on the data area of the “Main” worksheet. To implement the navigation, we use the Worksheet_BeforeDoubleClick VBA event in combination with the SAP IBP APIs GetAttributeValues and SetFilterValues. Using the double-click event, we get the attribute combination of the specific cell in the “Main” worksheet and set it as a filter for the second worksheet, so that data for this combination only is shown.
The code for this use case, is part of the worksheet “Main” and looks like the following:

```
Private IBPAutomationObject As Object

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)

On Error GoTo ErrorHandling:
    Dim attributes() As String

    If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
    Application.COMAddIns("IBPXLClient.Connect").Object

    'get Attributes
    attributes = IBPAutomationObject.GetAttributeValues(Target)
    'set Filter for Details
    Call IBPAutomationObject.SetFilterValues (attributes, Worksheets("Details"))
    Worksheets("Details").Activate
    Exit Sub

ErrorHandling:
'Implement an error handling to help the user to understand what went wrong
MsgBox Err.Description, vbOKOnly, "Microsoft Excel: Custom VBA code"

End Sub
```

Code explained row by row:

Private IBPAutomationObject As Object
The SAP IBP automation object enables you to call the SAP IBP APIs. The variable for the SAP IBP automation object needs to be declared as object data type in the VBA code, see SAP IBP Automation Object on the SAP Help Portal.

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
We are using the Microsoft Excel Worksheet_BeforeDoubleClick event to trigger the navigation using double-click, see the Microsoft documentation.

On Error GoTo ErrorHandling:
If an error occurs when the VBA code runs, continue at section “ErrorHandling”.

Dim attributes() As String
Variable declared as string array to save the returned attribute values of the cell, which has been double-clicked.

If IBPAutomationObject Is Nothing Then Set IBPAutomationObject = Application.COMAddIns("IBPXLClient.Connect").Object
If the SAP IBP automation object was not set yet, we retrieve it from the SAP IBP, add-in for Microsoft Excel (Excel add-in).

attributes = IBPAutomationObject.GetAttributeValues(Target)
Getting the attribute values for the range object called Target. This object defines the cell that has been double-clicked. For more information about the API GetAttributeValues, see GetAttributeValues on the SAP Help Portal.

Call IBPAutomationObject.SetFilterValues (attributes, Worksheets("Details"))
Setting the determined attribute values as an attribute-based filter for the “Details” worksheet. Predefined filter settings in the “Details” worksheet are overwritten. For more information about the API SetFilterValues and its parameters, see SetFilterValues on the SAP Help Portal.

Worksheets("Details").Activate
In the last step, the “Details” worksheet needs to be activated. With this, the worksheet is opened and the data gets automatically refreshed and updated with the new filter setting. (No additional Refresh action is needed.)

Exit Sub
Exit the method without running the ErrorHandling section if no error occurred in between.

ErrorHandling:
Marker of the error handling section.

MsgBox Err.Description, vbOKOnly, "Microsoft Excel: Custom VBA code"
Shows an error message with the error description. If an error occurs when calling the SAP IBP APIs, the Excel add-in provides an error description, which explains what went wrong. Please make sure to include appropriate error handling, depending on the additional custom code/methods you are implementing. If you do not, the users will get technical VBA error messages that they might not understand.

Further comments to the code:

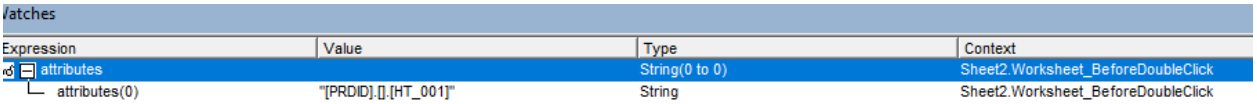
- To be able to use the SAP IBP APIs, an active connection to the SAP IBP system is needed.
- **GetAttributeValues**: The range which is provided as parameter must be part of the data area of the planning view. If, for example, the key figure cell is double-clicked, an error message comes up.
- **SetFilterValues:** If no worksheet parameter is passed, the filter settings of the active worksheet (Excel.ActiveSheet) are changed. The referenced worksheet must contain a planning view, and at least one filter criterion must be passed.

For more information, see SAP IBP APIs on the SAP Help Portal.

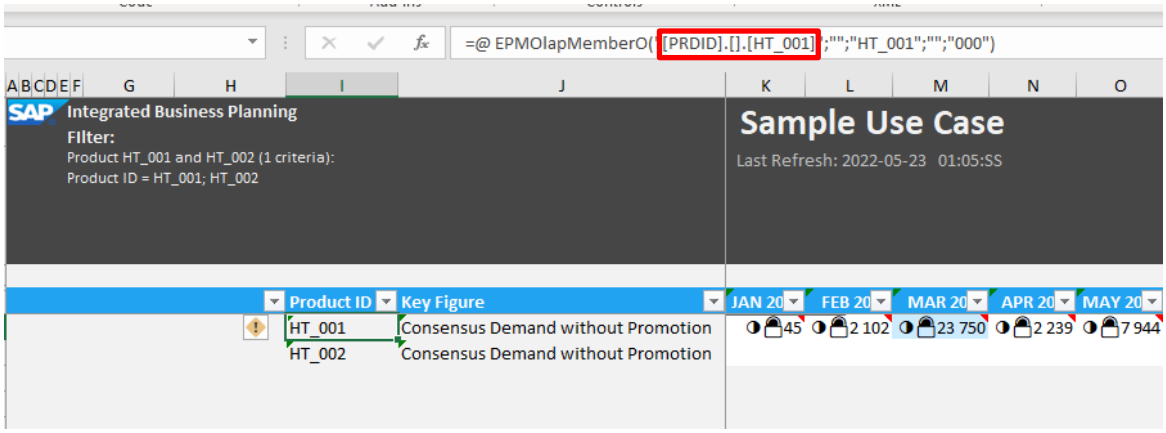
1.2. Format for attribute values

We aimed to define the SAP IBP APIs to be as simple as possible, to reduce the code lines which need to be written, and to make them easy to use. The input parameters, as well as the return values are using simple data types used in Microsoft Excel. This is the case for the two methods we’ve used so far.

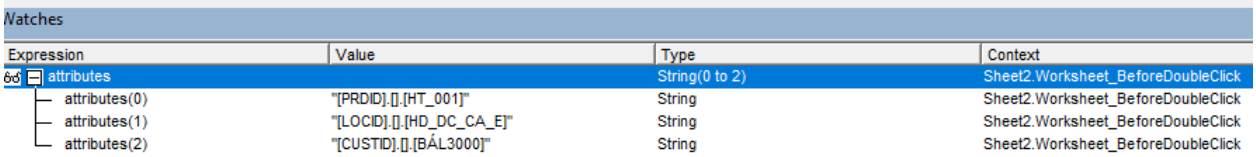
A string array is used as return value for the attribute values of the API GetAttributeValues. The same string array can be used as input parameter for the API SetFilterValues. What does the format we are using look like?



Each entry of the array has the format **[attribute ID].[].[attribute value].** In the example above, you see [PRDID].[].[HT_001]. In this case, PRDID is the attribute ID, the square brackets in the middle stay empty and the square brackets at the end include the attribute value, which is HT_001. This is the same format that is used in the formula when selecting an attribute value in an open planning view, as shown in the screenshot below.



If the respective planning view contains 3 different attributes, this is what the return value of GetAttributeValues for one specific cell would look like.

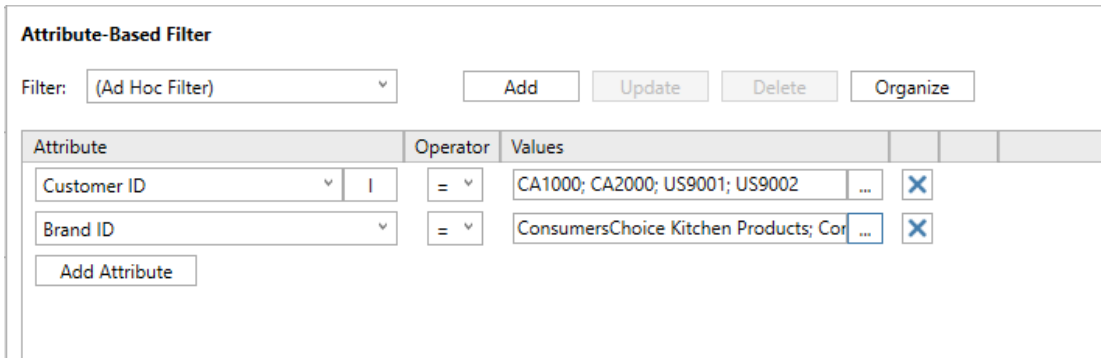


A one-dimensional string array with three entries, one for each attribute, is returned.
With the same approach, you can define filter settings in a one-dimensional string array, and use it as input parameter for the API SetFilterValues.

1.3. Merge Filter settings

The API GetFilterValues can be used to determine the attribute-based filter settings that have been defined in a planning view (see GetFilterValues on the SAP Help Portal). This enables you to get the filter settings of a specific planning view and merge them with additional filter criteria.

Given the case that you defined a filter for an attribute, and the filter is not included in the planning view settings of the “Main” worksheet (non-visible filter). This filter does not reduce the list of products, but it affects the values which are shown. In our example, a filter is included for a few customers.



If you want to pass the filter settings from the planning view including the combination chosen using double-click as filter to another worksheet, the existing VBA code needs to be adjusted.

In this case, first we determine the filter settings from the “Main” planning view definition using the API GetFilterValues. Then we merge the retrieved filter settings with the attribute values of the combination chosen using double-click and pass them to the “Details” planning view.
See below the VBA code with the extension explained above. The changes are highlighted in yellow:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)

On Error GoTo ErrorHandling:
    Dim attributes() As String
    Dim filterValues() As String
    Dim filterToPass As Variant

    If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
     Application.COMAddIns("IBPXLClient.Connect").Object

    'get Attributes
    attributes = IBPAutomationObject.GetAttributeValues(Target)
    filterValues = IBPAutomationObject.GetFilterValues
    'set Filter for Details
    filterToPass = MergeArray(attributes, filterValues)

    Call IBPAutomationObject.SetFilterValues (filterToPass, Worksheets("Details"))
    Worksheets("Details").Activate
    Exit Sub

ErrorHandling:
'Implement an error handling to help the user to understand what went wrong
MsgBox Err.Description, vbOKOnly, "Microsoft Excel: Custom VBA code"

End Sub
```

Code Explained Row by Row:

filterValues = IBPAutomationObject.GetFilterValues

Getting the filter values from the active sheet, in this case, the “Main” worksheet. The format is the same as explained in section 1.3:

filterValues		String(0 to 5)	Sheet2.Worksheet_BeforeDoubleClick
filterValues(0)	"[CUSTID].[].[CA1000]"	String	Sheet2.Worksheet_BeforeDoubleClick
filterValues(1)	"[CUSTID].[].[CA2000]"	String	Sheet2.Worksheet_BeforeDoubleClick
filterValues(2)	"[CUSTID].[].[US9001]"	String	Sheet2.Worksheet_BeforeDoubleClick
filterValues(3)	"[CUSTID].[].[US9002]"	String	Sheet2.Worksheet_BeforeDoubleClick
filterValues(4)	"[BRAND].[].[ConsumersChoice Kitchen Product	String	Sheet2.Worksheet_BeforeDoubleClick
filterValues(5)	"[BRAND].[].[ConsumersChoice Television]"	String	Sheet2.Worksheet_BeforeDoubleClick

filterToPass = MergeArray(attributes, filterValues)

Merging the two arrays, to pass them as a filter to the “Details” worksheet.
As there is no standard VBA function available to merge two arrays, we created a custom method to do so. The method might be further improved, if needed:

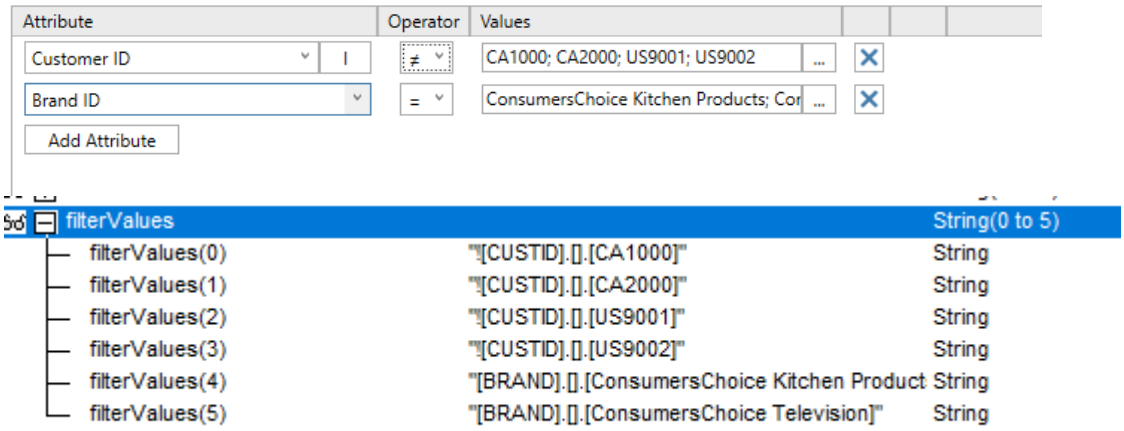
```
Private Function MergeArray(list1 As Variant, list2 As Variant) As Variant
    Dim result As Variant
    Dim size1 As Long
    Dim size2 As Long

    size1 = UBound(list1)
    size2 = UBound(list2)
    result = list1

    ReDim Preserve result(size1 + size2 + 1)
    Dim i As Integer
    For i = 0 To size2
       result(size1 + 1 + i) = list2(i)
    Next i
    MergeArray = result
End Function
```

Further comments to the code:

In the SAP IBP, add-in for Microsoft Excel, you can choose “not equal to” filter settings. The API GetFilterValues considers such filter settings as well. This is what the return value looks like:



To define “not equal to” filter settings, in the string array the character “!” is included before the attribute value definition, as described in 1.2.

Please note: The example code does not work as it should if the filter settings from the planning view definition include an attribute value from the combination chosen using double-click. By merging the arrays, the attribute value is added twice. If needed, doubled entries should be filtered out when merging the arrays.

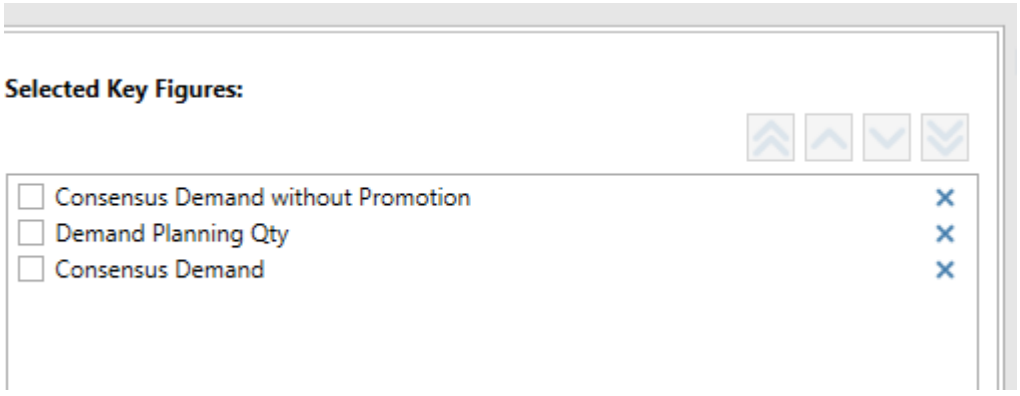
As an additional example, instead of retrieving the filter settings of the “Main” planning view, you could also retrieve predefined filter settings from the “Details” planning view and merge them with the filter for the combination chosen using double-click. As always, feel free to adjust the use cases to your business needs.

2. KEY FIGURE DEPENDENT NAVIGATION (EXAMPLE 2)

Let’s change the scenario a bit. In this example, you have a “Main” planning view created in the first worksheet. It contains several key figures, and it is mainly used for planning tasks.
In some cases, the planner needs further information about specific key figures, maybe even at a different planning level. Therefore, you want to include navigation patterns to additional worksheets with more detailed information about specific key figures. In our example, the user can double-click a key figure, and if there is a predefined worksheet for this key figure, the worksheet is opened.

2.1. How to set-up this use case

As a first step, you would need to add the needed key figures to the “Main” planning view.



Then create additional worksheets with detailed planning views for all the key figures you want to navigate to. In our example, we create two additional worksheets for the key figures “CONSENSUSDEMAND” and for “DEMANDPLANNINGQTY”. Please name the Excel worksheet the same way as the ID of the key figure. This makes the navigation and the VBA code easier.

In the next step, the VBA code from the former use case needs to be extended, as shown below. The changes are highlighted in yellow:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
    Dim formula As String
    Dim keyFigure As String
    Dim targetSheet As Worksheet

    formula = Target.Formula2
    If InStr(1, formula, "=@ EPMOlapMemberO(""[KEY_FIGURES].[].[") = 0 Then
       Exit Sub
    End If

    keyFigure = Replace(formula, "=@ EPMOlapMemberO(""[KEY_FIGURES].[].[", "")
    keyFigure = Left(keyFigure, InStr(1, keyFigure, "]""") - 1)

    On Error GoTo NWS:
          Set targetSheet = ActiveWorkbook.Sheets(keyFigure)

    On Error GoTo ErrorHandling:
       Dim attributes() As String
       Dim filterValues() As String
       Dim filterToPass As Variant

       If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
          Application.COMAddIns("IBPXLClient.Connect").Object

       'get Attributes
       attributes = IBPAutomationObject.GetAttributeValues(Target.Offset(0, 1))
       filterValues = IBPAutomationObject.GetFilterValues
       'set Filter for Details
       filterToPass = MergeArray(attributes, filterValues)

       Call IBPAutomationObject.SetFilterValues (filterToPass, targetSheet)
       targetSheet.Activate
       Exit Sub

ErrorHandling:
    'Implement an error handling to help the user to understand what went wrong
    MsgBox Err.Description, vbOKOnly, "Microsoft Excel: Custom VBA code"
    Exit Sub

NWS:
```

```
    MsgBox "No detailed planning view was found for the selected key figure.", vbOKOnly, "Microsoft Excel:
        Custom VBA code"
End Sub
```

Code explained row by row:

```
formula = Target.Formula2
If InStr(1, formula, "=@ EPMOlapMemberO(""[KEY_FIGURES].[].[") = 0 Then
   Exit Sub
End If
```
Check whether the formula of the target cell contains a key figure.

```
keyFigure = Replace(formula, "=@ EPMOlapMemberO(""[KEY_FIGURES].[].[", "")
keyFigure = Left(keyFigure, InStr(1, keyFigure, "]""") - 1)
```
Extract the key figure ID.

```
On Error GoTo NWS:
```
If an error occurs, go to the section NWS.

```
Set targetSheet = ActiveWorkbook.Sheets(keyFigure)
```
Determine the worksheet with the name of the key figure ID. We included a specific error handling for the case that no detailed planning view was configured for the specific key figure ID, see below:

```
NWS:
MsgBox "No detailed planning view was found for the selected key figure.", vbOKOnly, "Microsoft Excel:
Custom VBA code"
```
A message box is shown to explain the user that there is no preconfigured planning view available.

```
attributes = IBPAutomationObject.GetAttributeValues(Target.Offset(0, 1))
```
As the range which is provided as parameter for GetAttributeValues needs to be part of the data area of the planning view, we need to shift the Target range by one column (to the right).

```
Call IBPAutomationObject.SetFilterValues (filterToPass, targetSheet)
targetSheet.Activate
```
The variable targetSheet is now passed to the SetFilterValues API, instead of one specific worksheet (hard coded). This makes it possible to navigate to different worksheets. Then the targetSheet is activated.

Further comments to the code:

- This example code works only if the key figure is in the last column, (what we call our standard layout). If the key figure is in another column, you can adjust the code accordingly to determine the column, where the data area of the planning view starts.
- The advantage of this code is that it can be easily extended by business users. If they want to have an additional detailed view, they can add a new worksheet and create a planning view by themselves. The only thing which needs be considered is that the name of the worksheet should be the same as the key figure ID. To add additional worksheet navigations, no code extension is needed, and therefore the template administrator doesn't need to be involved.
- Please note that sheet names can be only 31 characters long, and SAP IBP key figure IDs can have up to 32 characters. Code adjustments would be needed, if key figures IDs with 32 characters should be used for the navigation.

Follow us

www.sap.com/contactsap