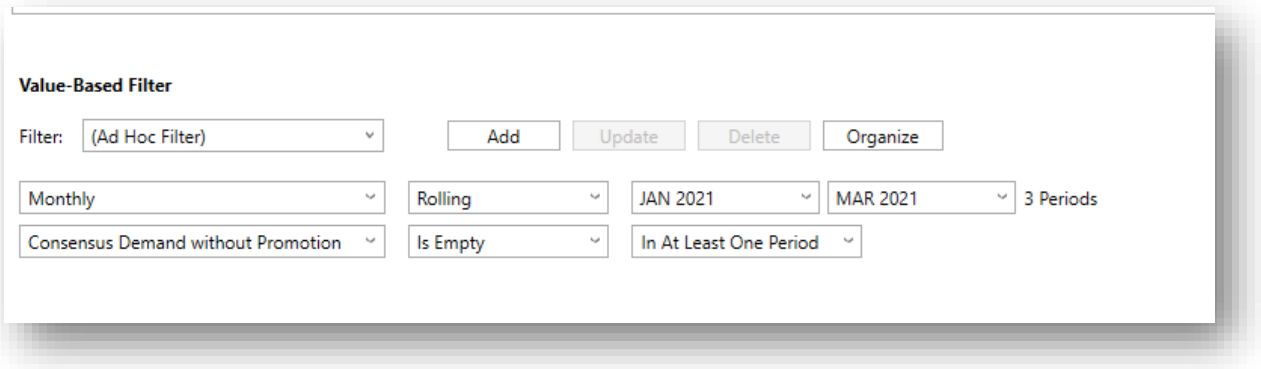
Combination-Based Planning

If planners are in charge of, for example, several product location combinations, they might check and adjust the key figure values for each combination one by one. Therefore, it would be beneficial to have only one combination in the “Planning” worksheet, where the data is reviewed and adjusted. Using the button “Save and Next” it’s possible to save the changes and get the next combination shown in the worksheet directly.

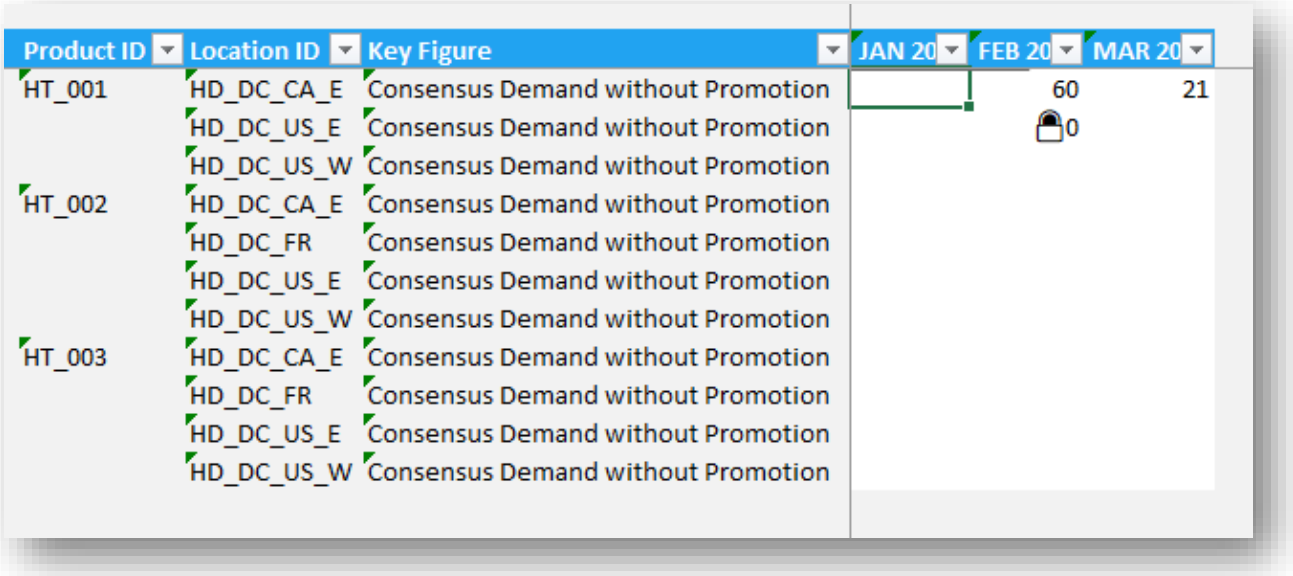
The list of combinations which need to be planned are included in a second worksheet named “List”. It’s possible to include the list as an Excel worksheet or, as we show it in our example, a planning view can be created including the list of combinations.

1. HOW TO START

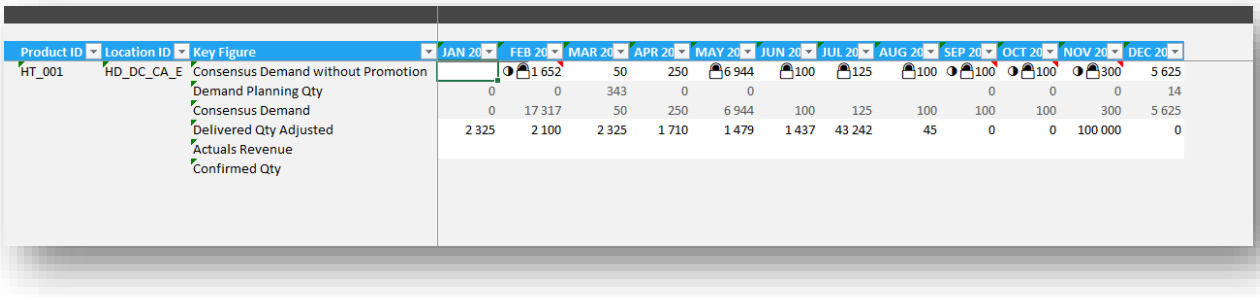
In this example, only the combinations where no Consensus Demand is planned yet within the next 3 months should be included in the list. Therefore, you need to set up a planning view for the worksheet named “List” with following value-based filter:



In this case, the key figure “Consensus Demand without Promotions” is shown at Product ID and Location ID level and monthly time settings are selected.

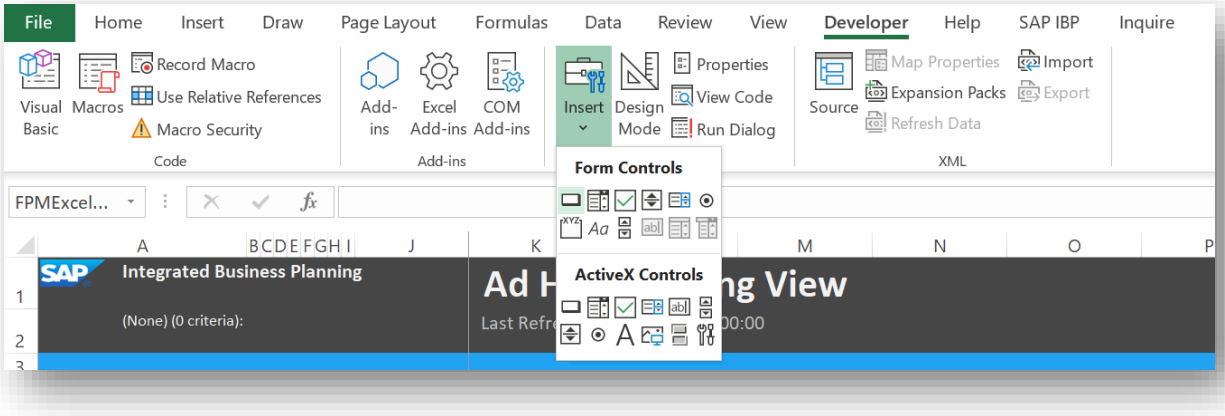


After creating the list of planning combinations, you need to create the planning view for the planning tasks (in our example “Planning” worksheet). You can select the same attribute and time settings as for the combinations list but include further key figures which are needed for the planning tasks.



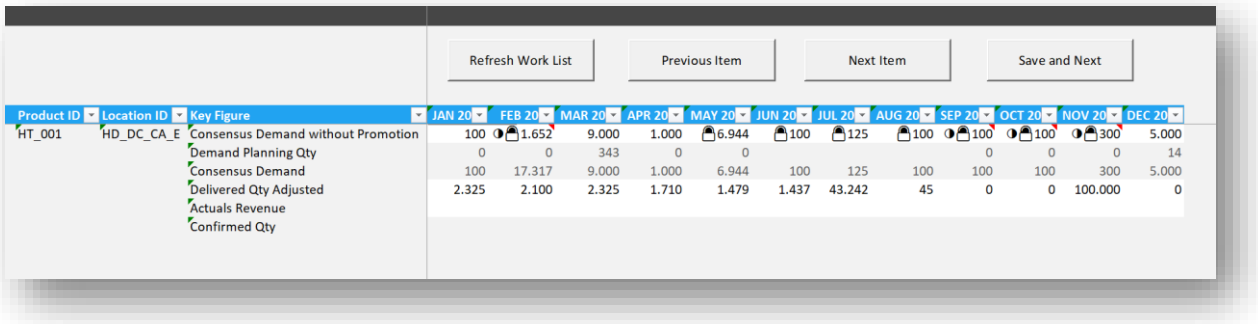
Product ID	Location ID	Key Figure	JAN 20	FEB 20	MAR 20	APR 20	MAY 20	JUN 20	JUL 20	AUG 20	SEP 20	OCT 20	NOV 20	DEC 20
HT_001	HD_DC_CA_E	Consensus Demand without Promotion		1 652	50	250	6 944	100	125	100	100	100	300	5 625
		Demand Planning Qty	0	0	343	0	0				0	0	0	14
		Consensus Demand	0	17 317	50	250	6 944	100	125	100	100	100	300	5 625
		Delivered Qty Adjusted	2 325	2 100	2 325	1 710	1 479	1 437	43 242	45	0	0	100 000	0
		Actuals Revenue												
		Confirmed Qty												

As a next step, insert four VBA buttons to the “Planning” worksheet (in the “Developer” tab of Microsoft Excel in the “Controls” section, select “Insert” and choose the button icon from “Form Controls”.



The buttons are used for the following:

- Refresh Worklist: refresh the planning view of the worksheet “List” and show the first combination of the list in the worksheet “Planning” (starting point)
- Previous Item: open the previous combination from the list
- Next Item: open the next combination from the list
- Save and Next: save the changes, and open the next combination from the list



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_001	HD_DC_CA_E	Consensus Demand without Promotion	100	1.652	9.000	1.000	6.944	100	125	100	100	100	300	5.000
		Demand Planning Qty	0	0	343	0	0				0	0	0	14
		Consensus Demand	100	17.317	9.000	1.000	6.944	100	125	100	100	100	300	5.000
		Delivered Qty Adjusted	2.325	2.100	2.325	1.710	1.479	1.437	43.242	45	0	0	100.000	0
		Actuals Revenue												
		Confirmed Qty												

After inserting the buttons and adjusting the captions, the VBA code for each button needs to be added into the Microsoft Excel Object “Sheet2 (Planning)”, as described in the following chapters.

2. REFRESH WORKLIST

Private IBPAutomationObject As Object
Private position As Range

*Private Sub **RefreshList_Click**()*

```
    Dim attributes() As String
    Dim rptID As String
    Dim topLeftCell As String
    On Error GoTo ErrorHandling:
        If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
          Application.COMAddIns("IBPXLClient.Connect").Object
        rptID = IBPAutomationObject.GetActiveReportName(ActiveSheet)
        If rptID = "" Then
            MsgBox ("The active worksheet has no planning view.")
            Exit Sub
        End If
        ActiveWorkbook.Sheets("List").Activate
        Call IBPAutomationObject.Refresh
        topLeftCell = IBPAutomationObject.getDataTopLeftCell(ActiveSheet, rptID)
        Set position = ActiveSheet.Range(topLeftCell)
        attributes = IBPAutomationObject.GetAttributeValues(position)
        Call IBPAutomationObject.SetFilterValues(attributes, ActiveWorkbook.Sheets("Planning"))
        ActiveWorkbook.Sheets("Planning").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:

Please note: Code lines which already have been explained in the tutorial for use case 1.1 “Create Customized Navigation Patterns” are not explained in detail again.

```
rptID = IBPAutomationObject.GetActiveReportName(ActiveSheet)
```

With the API GetActiveReportName it is possible to check whether the ActiveSheet contains a planning view or not. This information is needed for the API getDataTopLeftCell, which is explained below.

```
If rptID = "" Then
    MsgBox ("The active worksheet has no planning view.")
    Exit Sub
End If
```

If the return value is “” the worksheet contains no planning view, and an error message is shown.

```
ActiveWorkbook.Sheets("List").Activate
```

Activate the worksheet named “List”

```
Call IBPAutomationObject.Refresh
```

Refresh the active worksheet (“List”) to get the most recent information

```
topLeftCell = IBPAutomationObject.getDataTopLeftCell(ActiveSheet, rptID)
```

Determine the top-left cell of the data area within the planning view of the active worksheet (“List”)

```
Set position = ActiveSheet.Range(topLeftCell)
```

Remember the first cell in the data area of the planning view
To be able to use the same position as reference across all VBA methods, the variable is declared as global variable outside the function.

```
attributes = IBPAutomationObject.GetAttributeValues(position)
Call IBPAutomationObject.SetFilterValues (attributes, ActiveWorkbook.Sheets("Planning"))
```

Determine the attribute values of that cell and pass it as filter criteria to the planning view “Planning”

```
ActiveWorkbook.Sheets("Planning").Activate
```

Activate the worksheet named “Planning” to be able to start working there

3. PREVIOUS ITEM AND NEXT ITEM

```
Private Sub NextItem_Click()
   Dim attributes() As String
   Dim list As Worksheet
   If position Is Nothing Then
      MsgBox "Please first refresh the worklist, to initialize the planning view.", vbOKOnly, "Microsoft Excel:
      Custom VBA code"
      Exit Sub
   End If
   On Error GoTo ErrorHandling:
      If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
      Application.COMAddIns("IBPXLClient.Connect").Object
      Set position = position.Offset(1, 0)
      attributes = IBPAutomationObject.GetAttributeValues(position)
      Call IBPAutomationObject.SetFilterValues(attributes, ActiveWorkbook.Sheets("Planning"))
      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

Private Sub PreviousItem_Click()
   Dim attributes() As String
   Dim list As Worksheet
   If position Is Nothing Then
      MsgBox "Please first refresh the worklist, to initialize the planning view.", vbOKOnly, "Microsoft Excel:
      Custom VBA code"
      Exit Sub
   End If
   On Error GoTo ErrorHandling:
      If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
      Application.COMAddIns("IBPXLClient.Connect").Object
      Set position = position.Offset(-1, 0)
      attributes = IBPAutomationObject.GetAttributeValues(position)
      Call IBPAutomationObject.SetFilterValues(attributes, ActiveWorkbook.Sheets("Planning"))
      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:

The code which is assigned to the “Previous Item” and “Next Item” buttons does the following:

```
If position Is Nothing Then
      MsgBox "Please first refresh the worklist, to initialize the planning view.", vbOKOnly, "Microsoft Excel:
      Custom VBA code"
      Exit Sub
End If
```

Check if the worklist was refreshed and the position of the first combination was initialized

```
Set position = position.Offset(1, 0)
```
or
```
Set position = position.Offset(-1, 0)
```

Move the “position” range one row down or up

```
attributes = IBPAutomationObject.GetAttributeValues(position)
Call IBPAutomationObject.SetFilterValues (attributes, ActiveWorkbook.Sheets("Planning"))
```

Determine the attribute values of the changed position and pass it as filter criteria to the planning view “Planning”

4. SAVE AND NEXT

```
Private Sub SaveAndNext_Click()
    If position Is Nothing Then
        MsgBox "Please first refresh the worklist, to initialize the planning view.", vbOKOnly, "Microsoft Excel:
        Custom VBA code"
        Exit Sub
    End If
    On Error GoTo ErrorHandling:
        If IBPAutomationObject Is Nothing Then Set IBPAutomationObject =
        Application.COMAddIns("IBPXLClient.Connect").Object
        Call IBPAutomationObject.SaveData
        NextItem_Click
        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:

In this code, we additionally call the API SaveData before calling “NextItem_Click” to get the next combination shown in the “Planning” planning view.
For more information, see the SAP Help portal at SaveData.

Follow us

www.sap.com/contactsap

© 2022 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platforms, directions, and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See www.sap.com/trademark for additional trademark information and notices.