Tutorial

Create Excel UDF

UDF

User **D**efined **F**unction

Hermann Baum

Last update: 2024-03-19

mail@hermann-baum.de

https://hermann-baum.de/excel/hbSort/en/xlookup.php

Contents

Foreword

In Excel there are predefined functions such as SUM, IF, VLOOKUP, which can be used in the formula of a cell. In addition, it is possible to add your own functions with special tasks, the so-called UDFs (**U**ser **D**efined **F**unctions).

One of the ways to make this happen is to program them using the VBA (Visual Basic for Applications) programming language. This tutorial is about what needs to be considered and what difficulties you have to overcome. Experience in using VBA is required.

The tutorial refers to the UDF named '**XLookup2**'. An Excel file with the complete VBA code of this UDF can be downloaded from this website:

https://hermann-baum.de/excel/hbSort/en/xlookup.php

Introduction

In principle, any function programmed in a module that is not declared as 'Private' can be used as part of a formula within a cell of the worksheet - just like the predefined functions. However, in order for them to return results and not just return the #VALUE error code, some restrictions must be observed.

Restrictions:

1. The function names must not collide with the names of the predefined functions
2. The functions may only calculate and return values; they must not contain any actions such as changes in cell content or changes in cell formatting or other properties of the Excel objects
3. Arrays of a specified data type can neither be passed as a parameter nor returned as a function result; for this you have to use the data type 'Variant'

Small introductory example: The first Monday of the year

A UDF that takes a year as a parameter and returns the date of the first Monday of the New Year might look like this:

```
Function FirstMonday(Year As Long) As Date
    Dim NewYearsDay As Date
    Dim dayNr As Long
    NewYearsDay = DateSerial(Year, 1, 1)
    dayNr = NewYearsDay Mod 7   'Saturday = 0
    If dayNr < 3 Then
        FirstMonday = NewYearsDay + 2 - dayNr
    Else
        FirstMonday = NewYearsDay + 9 - dayNr
    End If
End Function
```

Such a function can be called within the VBA code from other functions or procedures, and it can also be used in a formula on the worksheet, e. g. =FirstMonday($B$4).

Focus of the tutorial: The function XLOOKUP2

This tutorial focuses on creating a more complex UDF. It's a function called **XLOOKUP2**. It expands the already diverse possibilities of the Excel function `XLOOKUP`.

It provides the following three abilities in addition to XLOOKUP:

- ✓ Search for **multiple** search criteria
- ✓ Use of **wildcards** within the search criteria
- ✓ Return **all** matches, not just the first one (optional)

The considerations of the XLOOKUP2 function are divided into four chapters:

1. Syntax of the function
2. Evaluation of the parameter list
3. Performing the search
4. Processing of the return value

1. Syntax of the function

The first six parameters of the XLOOKUP2 function are the same as those of the XLOOKUP function. After the sixth parameter, further pairs consisting of a search criterion and a search array can follow.

Syntax:

```
= XLookup2 ( lookup_value1, lookup_array1, return_array, [if_not_found],
  [match_mode], [search_mode],
  [lookup_value2], [lookup_array2], [lookup_value3], [lookup_array3], ... )
```

Parameter	Explanation
lookup_value 1	1st search criterion
lookup_array 1	Array or range in which the 1st search criterion is searched (one-dimensional - single column or single row)
return_array	Array or range from which found values are returned
if_not_found (optional)	Text returned in place of the #N/A error code if nothing was found
match_mode (optional)	Match type: 0: Exact match search (default) **1**: Invalid (generates the error #VALUE) **-1**: Invalid (generates the error #VALUE) 2: Search with wildcard-symbols, the wildcards *, ?, ~ can be used for the search
search_mode (optional)	Search mode: 1: Normal search order starting with the first item; the first match found is returned (default) -1: Reverse search order starting with the last item; the first match found is returned 2: Normal search order starting with the first item; all matches are returned -2: Reverse search order starting with the last item; all matches are returned
lookup_value 2 (optional)	2nd search criterion
lookup_array 2 (optional)	Array or range in which the 2nd search criterion is searched (one-dimensional - single column or single row)
lookup_value 3 (optional)	3rd search criterion
lookup_array 3 (optional)	Array or range in which the 3rd search criterion is searched (one-dimensional - single column or single row)
etc.	. . .

Search criteria that are not specified (just a comma) or contain the empty string have no effect on the search.

2. Evaluation of the parameter list

The first step is to convert the parameter list of the cell function XLOOKUP2 into a parameter list of the VBA function XLookup2. The second step then consists of reading the parameter values into corresponding VBA variables.

2.0 Parameter list in VBA

There are several possible structures for parameter lists in VBA.

1. Mandatory parameters only
   e. g. `Function Name (p1 As long, p2 As String) As Double`
2. Mandatory parameters and a fixed number of optional parameters
   e. g.
```
Function Name (p1 As long, p2 As String, Optional p3 As long = 1, _
               Optional p4 As long = 0)
```
3. Mandatory parameters and a indefinite number of optional parameters
   e. g. `Function Name (p1 As long, p2 As String, ParamArray arg() As Variant) As String`

In all three cases, the number of mandatory parameters can also be zero, so that one can also speak of six different cases.

A mixture of individual optional parameters and the ParamArray is not possible. The ParamArray must be of type Variant.

Since the first parameter can already be omitted in the XLOOKUP2 function, analogous to the XLOOKUP function, we are only left with case 6: the parameter list consists only of the ParamArray.

The function has the following header in VBA:
```
Function XLookup2(ParamArray arg() As Variant)
```
The return type is not specified, so it is Variant. It has to be Variant because this function returns different data types: ranges, arrays or single values - depending on the situation.

The first instruction is:
```
argCnt = UBound(arg)
```

The number of parameters is stored in the variable argCnt. Since the ParamArray is zero-based, the number 4 is stored here if 5 parameters were specified.

At least 3 parameters must be specified. Therefore, the next line of code ensures that an error code is returned if there are fewer than 3 parameters:
```
If argCnt < 2 Then EXIT_BY_ERROR
```

The small auxiliary procedure EXIT_BY_ERROR has the advantage that it can be used in different UDFs. It intentionally generates an error so that further execution of the function is aborted and the error code #VALUE is returned.

```
Private Sub EXIT_BY_ERROR()
    Dim errorArr() As Long
    errorArr(0) = 1   'exit by #VALUE error
End Sub
```

Since the Redim statement is missing here, the error 'Index out of range' is generated. Because the XLookup2 function was called as a cell function, no error message appears, but the cell function responds with the error code #VALUE.

The number of search criteria results from the number of parameters. It is determined with the following code:

```
If argCnt < 6 Then
    critCnt = 1
Else
    critCnt = (argCnt - 4) \ 2 + 1  'Explanation of the calculation term in Chap. 2.7
End If
```

The different pairs consisting of search criterion and search array should be stored in the two arrays `critArr` and `lookArr`. They are therefore dimensioned accordingly at this point:

```
ReDim critArr(1 To critCnt)
ReDim lookArr(1 To critCnt)
```

2.1 Evaluation of the 1st parameter

Reading the first parameter is easy:

```
If IsMissing(arg(0)) Then
    critArr(1) = ""
Else
    critArr(1) = arg(0)
End If
```

The IsMissing function can be used to check whether a parameter was specified or whether only a comma was set. In the second case, the empty string is entered as the first criterion. Criteria consisting of the empty string have no effect on the search.

The various search criteria are stored in the array `critArr`. Therefore, this first parameter is stored in the `critArr(1)` variable.

2.2 Evaluation of the 2nd parameter

The second parameter must be a range (object of type RANGE) or an array. This check does the statement `If Not IsArray(arg(1)) Then EXIT_BY_ERROR`

The `IsArray` function returns FALSE even if the address of a single cell or an array that consists of only one element is given as a parameter.

The next statement copies the values of the `arg(1)` variable to the `lkArr` variable:
`lkArr = arg(1)`

This is a critical point in that we must note that `arg(1)` can be either a range or an array. Of course, a user will usually specify a range as a parameter and only in special cases an array. However, there are still cases where formulas are used for this parameter that either return ranges – such as OFFSET – or often arrays of values.

The variable `lkArr` is initially an uninitialized variable of type Variant. There are now three cases to consider in the copying process mentioned above:

1. `arg(1)` is a range.
   In this case, `lkArr` becomes an array with the same dimensions as `arg(1)`, ie a two-dimensional array with 1 row and n columns or with n rows and 1 column.
2. `arg(1)` is an array consisting of n rows and one column.
   Again, `lkArr` has the same dimensions as `arg(1)`.
3. `arg(1)` is an array consisting of one row and n columns.
   This is the critical case because the `lkArr` variable now only has one dimension and querying the 2nd dimension with `UBound(lkArr, 2)` leads to a runtime error.

I don't know why a single-column array has two dimensions while a single-row array has only one dimension. Seems like a Microsoft bug to me.

Since this circumstance has to be taken into account in several places, the small procedure repairArray is used as a workaround.

```
Private Sub repairArray(ByRef arr As Variant)
    Dim res As Variant
    Dim uBnd As Long
    Dim maxCol As Long
    Dim col As Long

    If Not IsObject(arr) Then      'arr is an array, not a range
On Error GoTo UBoundError          'Workaround for a single-row array
        uBnd = UBound(arr, 2)
        GoTo GoOn                  'no error
UBoundError:
        maxCol = UBound(arr, 1)
        ReDim res(1 To 1, 1 To maxCol)
        For col = 1 To maxCol
            res(1, col) = arr(col)
        Next
        arr = res
        Resume GoOn
GoOn:
On Error GoTo 0
    End If
End Sub
```

It checks whether the relevant parameter is the critical 3rd case. If at the instruction
`uBnd = UBound(arr, 2)`
an error is thrown, the error is caught and the array is converted into a two-dimensional array
(1 row and n columns).

Now the dimensions can be determined:

```
maxrowLook = UBound(lkArr, 1)
maxcolLook = UBound(lkArr, 2)
```

If both the number of rows and the number of columns are greater than 1, the `XLOOKUP2` function returns the error code:

```
If maxrowLook > 1 And maxcolLook > 1 Then EXIT_BY_ERROR
```

The variable `byRow` stores whether it is a horizontal search in the row (`byRow = TRUE`) or vertically in the column (`byRow = FALSE`):

```
byRow = (maxcolLook > 1)
```

The variable `maxInd` contains the number of rows or the number of columns - depending on whether the search is horizontal or vertical:

```
If byRow Then
    maxInd = maxcolLook
Else
    maxInd = maxrowLook
End If
```

For each search criterion there is an associated search array in the parameter list. Just as the search criteria are stored in an array called `critArr`, the search arrays are stored in an array called `lookArr`. The variable `lookArr` is thus an array of arrays.

In the (rare) case of a horizontal search, the search array is converted from a single-row array to a single-column array (1×n matrix to n×1 matrix conversion). This is done by the following code section:

```
If byRow Then
    ReDim arr(1 To maxInd, 1 To 1)
    For ind = 1 To maxInd
        arr(ind, 1) = lkArr(1, ind)
    Next
    lookArr(1) = arr
Else
    lookArr(1) = lkArr
End If
```

In this way, the search criteria and search arrays are transferred to the actual search routines in a uniform format.

2.3 Evaluation of the 3rd parameter

First, with the help of the `isArray()` function, it is again ruled out that a single value or the address of a single cell was entered here:

```
If Not IsArray(arg(2)) Then EXIT_BY_ERROR
```

After that, this parameter also requires special treatment for a completely different reason. Microsoft has specified for its `XLOOKUP` function that, where possible, it does not return an array of values, but the range containing those values. From the VBA point of view, it is not an array that is returned in these cases, but a range object.

This has the following additional effect: A function call with the `XLOOKUP` function can be inserted as a parameter into a function that expects a range as a parameter (see example at the end of the tutorial).

In the following code section, the `isObject()` function is therefore used to check whether the third parameter is a range object or an array of values:

```
If IsObject(arg(2)) Then
    Set retArr = arg(2)
    maxrowRet = retArr.Rows.Count
    maxcolRet = retArr.Columns.Count
Else
    retArr = arg(2)
    Call repairArray(retArr)
    maxrowRet = UBound(retArr, 1)
    maxcolRet = UBound(retArr, 2)
End If
```

In the first case, the values are not simply copied to the `retArr` variable, but with the statement
Set retArr = arg(2)
the variable `retArr` points to the same object as `arg(2)`. It therefore has the type 'Range'.

The dimensions of the third parameter are stored in the variables `maxrowRet` and `maxcolRet`. The next two statements generate an error if the length of the first search array does not match the corresponding dimension of the return array:

```
If byRow And (maxcolLook <> maxcolRet) Then EXIT_BY_ERROR
If Not byRow And (maxrowLook <> maxrowRet) Then EXIT_BY_ERROR
```

2.4 Evaluation of the 4th parameter

It may be that after the 3rd parameter nothing follows in the parameter list. In this case the variable `argCnt` has the value 2. Therefore the evaluation starts with the IF statement
`If argCnt >= 3.`

The VBA code for evaluating the 4th parameter is:

```
If argCnt >= 3 Then
    If IsMissing(arg(3)) Then
        notFnd = CVErr(xlErrNA)
    Else
        notFnd = arg(3)
    End If
Else
    notFnd = CVErr(xlErrNA)
End If
```

Since the variable `notFnd` can be assigned an error type as well as a string, it must be of the variant type, otherwise a runtime error may occur.

The expression `CVErr(xlErrNA)` returns the error #N/A. If the 4th parameter is not specified, the error code #N/A appears in the cells in the case of zero hits. Otherwise, the value that was used for this parameter appears. Outputting a fixed string (e.g. "#N/A" or "#NV" in German) instead of the expression `CVErr(xlErrNA)` would be the worse solution, since the `CVErr` function outputs the error #N/A in the respective national language.

2.5 Evaluation of the 5th parameter

The following VBA code ensures that the default value is 0 and values other than 0 or 2 result in an error:

```
If argCnt >= 4 Then
    If IsMissing(arg(4)) Then
        mMode = 0
    Else
        mMode = arg(4)
    End If
Else
    mMode = 0
End If
If mMode <> 0 And mMode <> 2 Then EXIT_BY_ERROR
```

2.6 Evaluation of the 6th parameter

The default value is 1 and values other than 1, -1, 2, or -2 result in an error.

```
If argCnt >= 5 Then
    If IsMissing(arg(5)) Then
        sMode = 1
    Else
        sMode = arg(5)
    End If
Else
    sMode = 1
End If
If sMode <> 1 And sMode <> 2 And sMode <> -1 And sMode <> -2 Then EXIT_BY_ERROR
```

2.7 Evaluation of the remaining parameters

After the sixth parameter, an indefinite number of pairs consisting of search criterion and search array can follow. The evaluation, i.e. checking and saving in the variables `critArr` and `lookArr`, therefore takes place in a loop.

```
    If argCnt >= 6 Then
        For argNr = 6 To argCnt
            critNr = (argNr - 4) \ 2 + 1
            If argNr Mod 2 = 0 Then     'lookup criterion
                If IsMissing(arg(argNr)) Then
                    critArr(critNr) = ""
                Else
                    critArr(critNr) = arg(argNr)
                End If
            Else         'lookup array
                If Not IsArray(arg(argNr)) Then EXIT_BY_ERROR
                lkArr = arg(argNr)
                Call repairArray(lkArr)
                maxrowLook = UBound(lkArr, 1)
                maxcolLook = UBound(lkArr, 2)
                If maxrowLook > 1 And maxcolLook > 1 Then EXIT_BY_ERROR

                If byRow Then
                    ReDim arr(1 To maxInd, 1 To 1)
                    For ind = 1 To maxInd
                        arr(ind, 1) = lkArr(1, ind)
                    Next
                    lookArr(critNr) = arr
                Else
                    lookArr(critNr) = lkArr
                End If
            End If
        Next
    End If
```

The number of arguments has already been determined and saved in the variable `argCnt` (see Section 2.0). Because the ParamArray is zero-based, the FOR loop starts with 6, which is the 7th parameter.

With the instruction
`critNr = (argNr - 4) \ 2 + 1`
the serial number of the criteria is determined. The first two parameters have the indices 0 and 1 in the array `arg()`. They are given the index 1 in the arrays `critArr` and `lookArr` (1st pair of criteria). The second pair of criteria, if any, has indices 6 and 7 in the `arg()` parameter list, the third pair of criteria has indices 8 and 9, and so on.

This means that parameter indices 6 and 7 must lead to criterion index 2, parameter indices 8 and 9 to criterion index 3, etc. For example, the number 4 is subtracted from the parameter index 7, the result, the number 3, is divided by 2 using integer division (results in 1) and the number 1 is added. The result of the calculation is 2.
In the same way, the parameter indices 8 and 9 lead to the criteria index 3.

The evaluation of the search criteria is analogous to the evaluation of the first search criterion (see Chapter 2.1) and the evaluation of the search arrays analogous to the evaluation of the first search array (see Chapter 2.2).

3. Performing the search

The two parameters compare mode (0 or 2) and search mode (1, -1, 2, or -2) allow eight different cases.

match_mode (optional)	Match type: 0: Exact match search (default) 2: Search with wildcard-symbols, the wildcards *, ?, ~ can be used for the search
search_mode (optional)	Search mode: 1: Normal search order starting with the first item; the first match found is returned (default) -1: Reverse search order starting with the last item; the first match found is returned 2: Normal search order starting with the first item; all matches are returned -2: Reverse search order starting with the last item; all matches are returned

The search for matches runs according to the following schedule:

For example, if it is a vertical search (search in the column), it is first checked whether all search criteria match the respective first element of the corresponding search array (match of all search criteria in the first row). This row is only considered a hit and the row number is saved in a "hit array" (variable `indArr`) if all search criteria match.

In search modes 1 and -1, the search is aborted after the first hit found. In search modes 2 and -2, the search continues to the end of the list and the row numbers of all hits are stored in the `indArr` array.

For performance reasons, four individual search routines with similar VBA code tailored to the four search modes are used instead of a single search routine.

```
If sMode = 1 Then
    indArr = SeqLookupAsc2Break(lookArr, critArr, mMode)
ElseIf sMode = -1 Then
    indArr = SeqLookupDesc2Break(lookArr, critArr, mMode)
ElseIf sMode = 2 Then
    indArr = SeqLookupAsc2All(lookArr, critArr, mMode)
Else    'sMode = -2
    indArr = SeqLookupDesc2All(lookArr, critArr, mMode)
End If
```

Within each of these four search routines, a distinction is made between mMode = 0 (exact search) and mMode = 2 (search with wildcards):

```
Function SeqLookupAsc2Break(lookArr As Variant, critArr As Variant, mMode As Long)
    ...
    If mMode = 0 Then
    ...
    Else  'mMode = 2
    ...
    End If
    SeqLookupAsc2Break = indArr
End Function
```

The following code excerpt shows the case of an exact search (match mode = 0) with return of all hits found (search mode = 2), i.e. call of the search routine `SeqLookupAsc2All`.

```
If mMode = 0 Then
    ind = 1
    k = 0
    While ind <= maxInd
        gefunden = True
        For critNr = 1 To critCnt
            If (lookArr(critNr)(ind, 1) <> critArr(critNr)) And (critArr(critNr) <> "")
            Then
                gefunden = False
                Exit For
            End If
        Next
        If gefunden Then
            k = k + 1
            indArr(k) = ind
        End If
        ind = ind + 1
    Wend
Else  'mMode = 2
```

In the while loop, the variable `ind` iterates through all values from 1 to `maxInd`. The variable `maxInd` stores the number of elements in the search array. For a vertical search (search in the column) this would be the number of rows.

We stay with the example of a vertical search:
All search criteria must be checked in each row. A hit has only been found if all search criteria are met in a row. This check of all search criteria within a row takes place in the inner FOR loop. The variable `gefunden` initialized with `TRUE` is set to `FALSE` at the first mismatch. If it is still `TRUE` at the end of the FOR loop, there is a hit.

The additional condition
`... and (critArr(critNr) <> "")`
ensures that search criteria consisting of the empty string have no effect.

The ELSE part (`mMode = 2`) only differs in a single line:
The test condition is no longer
`If (lookArr(critNr)(ind, 1) <> critArr(critNr)) And (critArr(critNr) <> ""),`
but
`If (Not (lookArr(critNr)(ind, 1) Like critArr(critNr))) And (critArr(critNr) <> "").`

The Like operator compares a string to a pattern.
The syntax is:
<result> = <string> Like <pattern>.
Make sure that the string is to the left of the like operator and the pattern with the wildcards to the right.

Each of the four search routines returns an array with the variable name `indArr`, which contains the indices of the hits found. If only the first hit is searched for (`sMode` = `1` or `sMode` = `-1`), the array contains at most one hit index, which is stored in the `indArr(1)` variable.

If no hit was found, `indArr(1) = 0` .

The array `indArr` forms the basis for the subsequent processing of the return value of the `XLOOKUP2` function.

4. Processing the return value

The simplest case is when no match is found. Then the string specified in the 4th parameter or the default value "#N/A" is returned. For this reason, processing begins with the following IF statement:

```
If indArr(1) < 1 Then
    ret = notFnd
Else
...
End If
```

In any case, the variable `ret` should contain the return value. Since it is of type Variant, it can contain a string, a numeric value, an array of values, or even a found range.

A range can be returned if a single match was searched for (search mode equals 1 or -1) and a range was passed as the return array (3rd parameter).

The following code snippet shows how the individual rows found are assembled into a return array in the event that all matches were searched for:

```
        If sMode = 2 Or sMode = -2 Then
            ReDim ret(1 To maxrowRet, 1 To maxcolRet)
            If byRow Then
            ...
            Else
                For row = 1 To maxrowRet
                    For col = 1 To maxcolRet
                        If indArr(row) >= 1 Then
                            ret(row, col) = retArr(indArr(row), col)
                        Else
                            ret(row, col) = ""
                        End If
                    Next
                Next
            End If
        Else      'sMode = 1 or -1
```

The variable `ret` is resized to have the same dimensions as the return array (3rd parameter). All rows found are transferred to the array ret in the order found. Empty strings are written in the remaining rows.

If only the first match was searched for (search mode equals 1 or -1), the code looks a little different:

```
Else     'sMode = 1 or -1
   If byRow Then
   ...
   Else
      If IsObject(retArr) Then
         Set ret = Range(retArr.Cells(indArr(1), 1), retArr.Cells(indArr(1), maxcolRet))
      Else
         ReDim ret(1 To 1, 1 To maxcolRet)
         For col = 1 To maxcolRet
            ret(1, col) = retArr(indArr(1), col)
         Next
      End If
   End If
End If
```

The following should be noted here:
If the return array (3rd parameter) is a range and not an array, then the XLOOKUP2 function should return the **found range** and not an array.

It is therefore checked whether the variable retArr is an object – and therefore a range. If so, the ret variable is assigned a reference to the found Range object:

```
Set ret = Range(retArr.Cells(indArr(1), 1), retArr.Cells(indArr(1), maxcolRet))
```

As a reminder: The variable indArr(1) contains the row number of the row found. With their help, a Range object is formed that contains the row found.

If the return array (the 3rd parameter) is not a Range object, the hit row values are copied into the ret variable. In this case, the variable ret is a two-dimensional array that only consists of one row – or in other words: a 1×n matrix.

Now you might think that you only need to add the final statement XLookup2 = ret. But then the function would return an array of values in any case. If you want to return a range object, the statement is:
Set XLookup2 = ret.
Since this would lead to a runtime error in the case of an array, a case distinction is required:

```
   If IsObject(ret) Then
       Set XLookup2 = ret
   Else
       XLookup2 = ret
   End If
```

In this way, the XLOOKUP2 function, analogous to the XLOOKUP function, returns a range in the event of a successful search for a single hit. This range could then be used as a parameter in a function that requires a range as a mandatory parameter, such as the OFFSET function.

Here's an **example**:
The following formula is a valid function call:

```
=OFFSET(XLookup2($I$25,$C$4:$C$23,$B$4:$G$23,,0,1),0,2,1,3)
```

The XLOOKUP2 function returns a hit row from the range $B$4:$G$23, which serves as the basis for the OFFSET function.