Independent REGEX for Excel VBA

These instructions describe how to install and use Independent REGEX for Excel VBA

Requirements

- Windows 8.1 or later
- Excel 2007 or later
- At least 5.87 GB of useable RAM
- At least these reference libraries activated in Excel:
    - Visual Basic for Applications
    - Microsoft Excel (12.0 or 16.0) Object Library
    - Microsoft Forms 2.0 Object Library

Guide to Regular Expressions

Regular expressions (regex) are coded strings that represent patterns that could be found in literal strings. Regex are mixes of literal characters (e.g., “a”) and metacharacters.

Metacharacters in this version of regex:

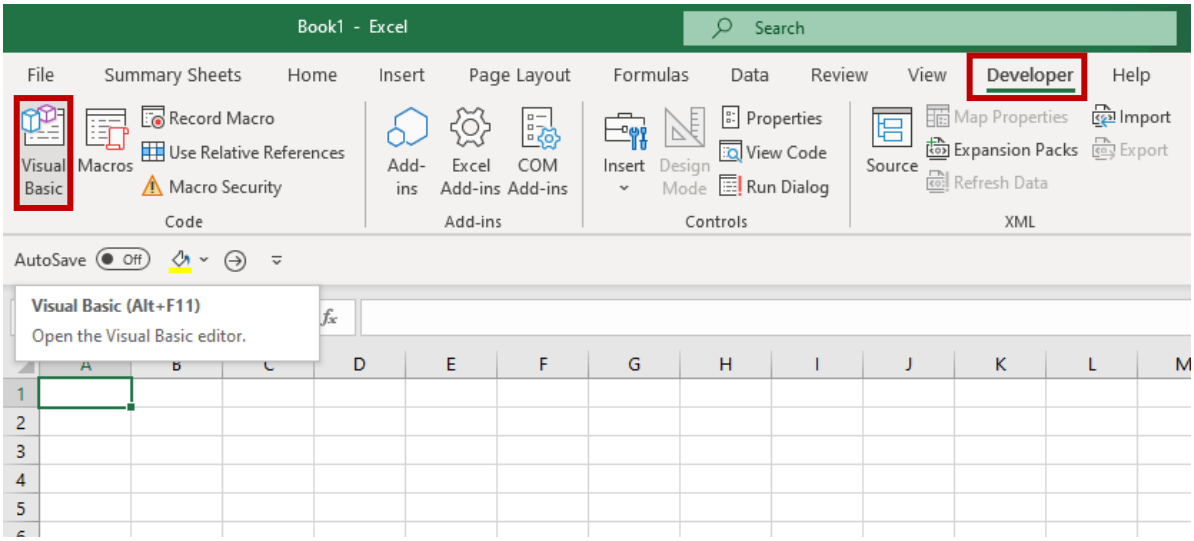
- "^" = front anchor (to match, must be at front of search string)
    - “^shark” matches “shark”, but not “whaleshark”
- "$" = end anchor (to match, must be at end of search string)
    - “r$” matches “toaster”, but not “rake”
- "(" and ")" = used for grouping, not necessarily expected to be in literal string
    - “(hi)” matches “(hi)” and “hi”
- "|" = or operator
    - “a|b” matches with “a” and “b”
- "?" = operator indicating 0-1 instances of preceding element
    - “a?” matches with null string and “a”
- "*" = operator indicating 0+ instances of preceding element
    - “a*” matches with null string, “a”, “aa”, “aaa”, etc.
- "+" = operator indicating 1+ instances of preceding element
    - “a*” matches with “a”, “aa”, “aaa”, etc.
- {X}, with X representing a range (see below) = operator indicating a certain number of instances of preceding element
    - {P}, where P is a positive integer = P instances or preceding element
        - “a{5}” matches with “aaaaa”
    - {P,}, where P is a positive integer = P+ instances or preceding element
        - “a{2}” matches with “aa”, “aaa”, “aaaa”, etc.
    - {,P}, where P is a positive integer = 0-P instances or preceding element
        - “a{,2}” matches with null string, “a”, and “aa”
    - {N,P}, where N is a non-negative integer, and P is a positive integer = N-P instances or preceding element
        - “a{2,4}” matches with “aa”, “aaa”, and “aaaa”
- " [X] ", with X representing a string = character class (aka “character group”)

    - Everything inside considered literal
    - Can include ranges with dash in middle (e.g., “a-z”); either end of range must be in ASCII order
    - [a-z] = any single char from a to z
    - [a-z0-9] = any single char from a to z or 0 to 9
    - [a-z0-9!_] = any single char that’s “!”, “_”, from a to z, or from 0 to 9
- "." = any character except new line
- "\d" = digit character (0-9)
- "\D" = non-digit character
- "\s" = whitespace character
- "\S" = non-whitespace character
- "\w" = alphanumeric character or “_”
- "\W" = character that’s not alphanumeric or “_”
- "\|" = "|" (escaped to be literal in search string)
- "\(" = "(" (escaped to be literal in search string)
- "\)" = ")" (escaped to be literal in search string)
- "\?" = "?" (escaped to be literal in search string)
- "*" = "*" (escaped to be literal in search string)
- "\+" = "+" (escaped to be literal in search string)
- "\." = "." (escaped to be literal in search string)
- "\[" = "[" (escaped to be literal in search string)
- "\]" = "]" (escaped to be literal in search string)
- "\{" = "{" (escaped to be literal in search string)
- "\}" = "}" (escaped to be literal in search string)
- "\\" = "\" (escaped to be literal in search string)
- "\^" = "^" (escaped to be literal in search string)
- "\$" = "$" (escaped to be literal in search string)
- "\\d" = "\d" (escaped to be literal in search string)
- "\\D" = "\D" (escaped to be literal in search string)
- "\\s" = "\s" (escaped to be literal in search string)
- "\\S" = "\S" (escaped to be literal in search string)
- "\\w" = "\w" (escaped to be literal in search string)
- "\\W" = "\W" (escaped to be literal in search string)

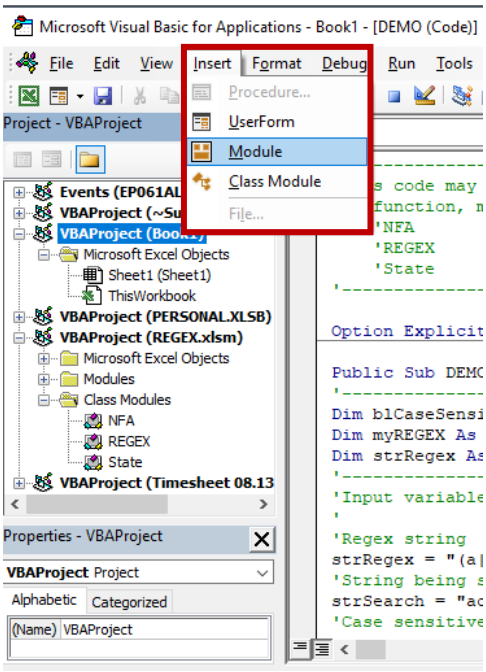
Adding Independent REGEX Class Modules to Excel VBA Project

Note: shown with Excel 365. Red boxes added for emphasis.

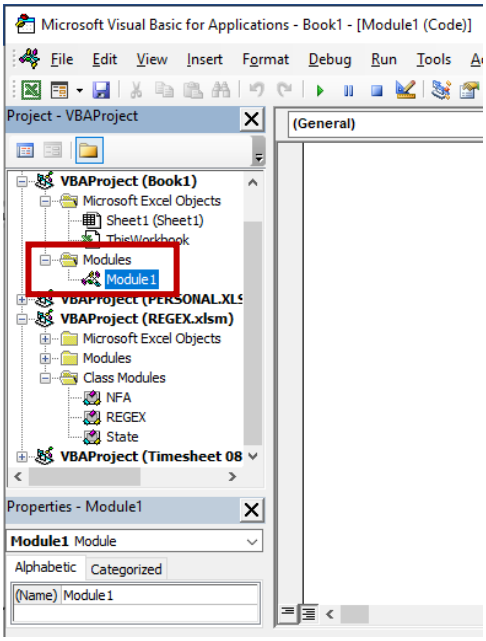
1. Open Excel
2. In Excel, open workbook containing class modules (i.e., "REGEX.xlsm")
3. In Excel,
    a. open add-in or Excel-enabled workbook that will use class modules
    b. or create new workbook.
4. Access Visual Basic for Applications GUI (graphic user interface). In Excel 365:
    a. If Developer tab does not exist:
        i. Right click on ribbon
        ii. Click "Customize the Ribbon"
        iii. Under "Customize the Ribbon", choose Main Tabs from dropdown menu
        iv. Click checkbox next to Developer
        v. Click OK
    b. Click on Developer tab.
    c. Under Code group, click Visual Basic.



5. If code does not already exist in desired workbook, create module. In Excel 365's VBA GUI, click on Insert tab, then Module.

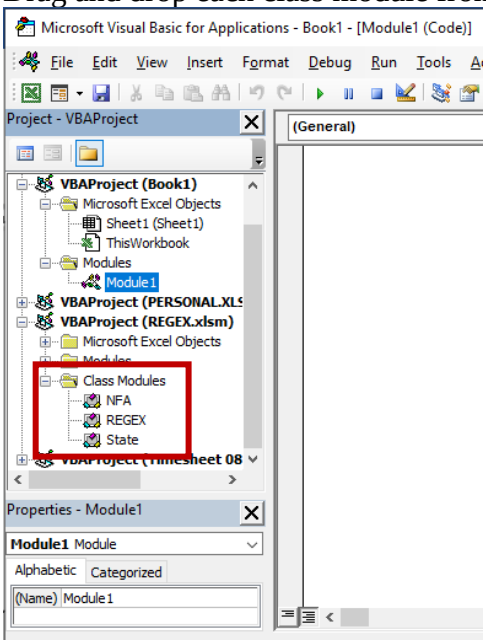


Module has been added under Modules folder for project.

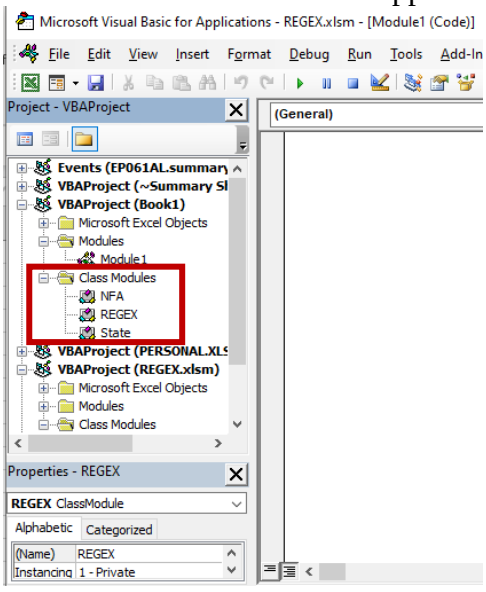


6. Copy all Independent REGEX classes modules to project (drag and drop)
    - NFA
    - REGEX
    - State

Drag and drop each class module from “REGEX.xlsm” to desired project.



Class Modules folder should appear in desired project, containing the class modules.



7. Write code that uses the class modules (see “Code Description”).
8. To keep changes, save as .xlsm (macro-enabled workbook) or .xlam (add-in)

Code Description

This section describes how to include regex in module code.

To create regex:
Dim myREGEX As New REGEX

- myREGEX = variable name for new regex

To set up regex:
blSuccess = myREGEX.Setup (strRegex)

- blSuccess = Boolean: true if regex was built successfully, false otherwise
- strRegex = Regular expression as string (e.g. "(a|b)(c|d)")

Recommendation: wherever possible, create the regex once and use it to check against multiple search strings

To check whether search string matches regex:
blResult = myREGEX.Match(strSearch, blCaseSensitive, blWholeStrOnly)

- blResult = Boolean: true if regex found in search string under given conditions, false otherwise
- strSearch: String being searched for instance of regex
- blCaseSensitive: Boolean (optional; default = true): if true, search string must match case of regex (for literal alphabetical characters), if false, not case sensitive
- blWholeStrOnly: Boolean (optional; default = false): if true, regex must match with the entire search string to be considered a match; if false, regex may match with a part of the search string

Demo Code

```
'---------------------------------------------------------------------------------------------------
'This code may be copied to a module and run to demonstrate Independent Regex
'To function, must have these class modules in same project:
   'NFA
   'REGEX
   'State
'---------------------------------------------------------------------------------------------------
Option Explicit
Private Sub DEMO()
'Code-based demonstration:  user can change the input variables, run subroutine, and get
Independent Regex results in immediate window
'---------------------------------------------------------------------------------------------------
Dim blCaseSensitive As Boolean, blResult As Boolean, blSuccess As Boolean, blWholeStrOnly
As Boolean
Dim myREGEX As New REGEX
Dim strRegex As String, strSearch As String, strSearch2 As String
'---------------------------------------------------------------------------------------------------
'Input variables
'
'Regex string
strRegex = "(a|b)(c|d)"
'Strings being searched for regex
strSearch = "ac"
strSearch2 = "aq"
'Case sensitive? (optional; default = true)
blCaseSensitive = True
'Matches for whole word only? (optional; default = false)
blWholeStrOnly = False
'---------------------------------------------------------------------------------------------------
'Setup (can then be used for multiple matches)
blSuccess = myREGEX.Setup(strRegex)
If blSuccess = False Then
   Debug.Print "Warning:  setup failed"
End If
'
'Checking match with search string
blResult = myREGEX.Match(strSearch, blCaseSensitive, blWholeStrOnly)
Debug.Print "Result 1 = " & CStr(blResult)
'
'Using same regex on different search string
blResult = myREGEX.Match(strSearch2, blCaseSensitive, blWholeStrOnly)
Debug.Print "Result 2 = " & CStr(blResult)
'
End Sub
```