PharmaSUG 2017 - Paper QT14

Visual Basic for Applications: A Solution for Handling Non-ASCII Removal and Replacement

Eric Crockett, Chiltern International

ABSTRACT

CDISC (or Clinical Data Interchange Standards Consortium) submission data and documentation should not include non-ASCII or non-printable characters. However, the source information used to support the data and documentation often (and unavoidably) includes such characters. For the Trial Design domains such as TI or TS, it is common for companies to leverage SAS® macros to address this issue. When the source information is present in the mapping specification the best approach may be to programmatically address the issue within the specification. An example of how special characters can be removed and replaced at the specification level using a Microsoft Office Visual Basic for Applications (VBA) macro will be shared and the practical advantages discussed.

INTRODUCTION

Non-ASCII characters are prohibited in submission data and must be removed or replaced. One common source for prohibited characters is Excel data or metadata. Companies commonly use SAS® to remove or replace non-ASCII characters, but when the issue originates in Excel it is often best to correct it at the source using a VBA macro.

Some might already be familiar with the clean(*) function to remove most non-printable characters. The clean function will simply remove most unwanted characters, however it will not replace characters with a logical replacement (e.g. a space for a carriage return). Just removing offending characters quickly leads to issues (e.g. removing a carriage return often makes two words into one).

With a VBA macro, removal and replacement choices can be managed at a granular level. Examples of appropriate replacements include: ™ replaced with ^TM or © replaced with (C). The replace function, which is only available in VBA, is used to accomplish these tasks. With proper planning, replacements are easily recognizable and conform to submission data standards.

METHODOLOGY

The primary goal is to end up with a set of cells that contain only printable characters. The approach is to cycle through each user-selected cell and perform a series of replacements on each cell. The series of replacements are split into two types: single-digit character replacements and replacements involving more than one character.

CELL SELECTION & LOOPS

User-specified cells are more desirable and increase the functionality of the macro. Users may only want to modify a subset of the cells and fewer processed cells results in faster processing time. The following code assigns a dynamically defined cell reference for the selected cells to be modified. All the user must do is select the cells over which to run the macro.

```
Set Cells = Range(Selection.Address)
```

No application works as expected if the user unintentionally makes a mistake. Many types of user errors can be addressed with a prompt. For example, when a user selects a huge number of cells to cycle through, processing time dramatically increases. For this reason it has been considered worthwhile to include a prompt to determine whether or the number of cells selected was intentional. The following code forces the user to confirm the number of cells selected with a simple yes or no response.

```
Dim m As Long
m = Cells.Count
If m >= 100 Then
    If MsgBox("You have selected " & m & " cells to modify. Are you
      sure?", vbYesNo) = vbNo Then Exit Sub
End If
```

Using a for-loop in VBA involves declaring two variables of the same type (in this case it is “As Range”) which is simply defining that the variable will hold set of cells. The first variable is assigned the user-selected cells; the other is used only as the reference cell in the for-loop, so it is not actually initialized to any value as its values change as the macro loops. As the macro loops through each of the user-selected cells, the program performs the series of replacements defined in the VBA macro. Since the replacements are restricted to what is defined in the code, this macro is a “living macro” in that it is intended to be expanded and improved with time.

```
Dim SingleCell, Cells As Range

For Each SingleCell In Cells
     <...replacement code goes here...>
Next SingleCell
```

THE REPLACE FUNCTION

The replace function is the real driving force in the apparatus. There are 3 primary inputs required for the replace function: a string, the target, and the replacement. The following is a one-line example replacing the trademark character ™ with a logical replacement such as ^TM.

```
SingleCell.Value = Replace(SingleCell, "™", "^TM")
```

The next step is to define two string variables for single-character replacements. For this task, the use of a for-loop is the best fit because it does not involve multiple lines of code for each single-character replacement. In the example below, the first string contains the target characters and the second string contains the corresponding replacement. The characters are displayed directly above one other so the strings can easily be compared. VBA allows most non-printable characters to be a part of the programming language whereas SAS® does not. This feature facilitates the maintenance and readability of the program.

```
SeltReplace = "ÉÈèéëÏïÖöÑñüµ––÷×ß‘’`´"
Replacement = "EEeeeIiOoNnuu--/xB''''"
```

The next step is to define the characters that will be replaced with more than one character. Replacement of multiple characters is generally more than one non-printable character requires individual replace function calls (see the trademark example above).

GENERAL COMMENTS REGARDING VBA MACROS

There are some general comments regarding VBA macros that are of concern as they are likely sources of frustration and anguish. These include processing speeds and some drawbacks to VBA.

PROCESSING SPEED

When implementing a VBA macro it important to minimize the use of system resources. By default, VBA macros run in real-time. So VBA macros visually update the spreadsheet while the program is running and unless otherwise specified, you could actually watch the macro work. While that may seem like a nice to spend a few minutes, this greatly decreases performance, particularly when a macro has a large number of tasks. One way to decrease the processing time is to turn off this default effect with the following code.

```
Application.ScreenUpdating = False
      <...all code goes here...>
Application.ScreenUpdating = True
```

MACROS CANNOT BE UNDONE

One of the largest drawbacks to VBA macros is that they cannot be undone. In fact, all editing history is erased so it is important to save your files prior to running any VBA macros.

SECURITY

Only use VBA macros that you trust and know are safe. Malicious code exists and VBA is particularly powerful in that it can search through file systems and can even pull your windows user account name and more.

CONCLUSION

It has been shown how Excel VBA can be used to resolve data submission standards by removing and replacing non-ASCII characters with a logical ASCII-based substitution. Excel VBA macros are a powerful tool that provide general support in the process of implementing CDISC standards.

ACKNOWLEDGMENTS

A special thanks to Steven Kirby, who helped make this paper possible through his support of my Excel VBA endeavors.

RECOMMENDED READING

- *WiseOwlTutorials – YouTube.com*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Eric Crockett
Chiltern International
eric.crockett@chiltern.com

APPENDIX

CODE

```
Sub RemoveAndReplaceAllNonASCII()
    Application.ScreenUpdating = False

    Dim SeltReplace, Replacement, Char, Rep, TheCell As String
    Dim i, j, k As Integer
    Dim SingleCell, Cells As Range
    Dim m As Long
    SeltReplace = "ÉÈèéëÏïÖöÑñüµ—–÷×ß‘’`´"
    Replacement = "EEeeeIiOoNnuu--/xB''''"
    k = Len(SeltReplace)

    Set Cells = Range(Selection.Address)
    m = Cells.Count
    If m >= 100 Then
        If MsgBox("You have selected " & m & " cells to modify. Are you
          sure?", vbYesNo) = vbNo Then Exit Sub
    End If
    For Each SingleCell In Cells

            For i = 1 To k

                Rep = Mid(Replacement, i, 1)
                Char = Mid(SeltReplace, i, 1)
                SingleCell.Value = Replace(SingleCell, Char, Rep)

            Next i

            SingleCell.Value = Replace(SingleCell, "™", "^TM")
            SingleCell.Value = Replace(SingleCell, "©", "(C)")
            SingleCell.Value = Replace(SingleCell, "®", "(R)")
            SingleCell.Value = Replace(SingleCell, "²", "^2")
            SingleCell.Value = Replace(SingleCell, "³", "^3")
            SingleCell.Value = Replace(SingleCell, ChrW(&H2264), "<=")
            SingleCell.Value = Replace(SingleCell, ChrW(&H2265), ">=")
            SingleCell.Value = Replace(SingleCell, ChrW(945), "alpha")
            SingleCell.Value = Replace(SingleCell, "•", "- ")
            SingleCell.Value = Replace(SingleCell, "Æ", "AE")
            SingleCell.Value = Replace(SingleCell, "æ", "ae")
            SingleCell.Value = Replace(SingleCell, "…", "...")
            SingleCell.Value = Replace(SingleCell.Value, vbLf, " ")
            SingleCell.Value = Replace(SingleCell, "m2", "m^2")
            SingleCell.Value = Replace(SingleCell, "m3", "m^3")
            SingleCell.Value = Replace(SingleCell, "″", Chr(34))
            SingleCell.Value = Replace(SingleCell, ChrW(&H9702), "-")
            SingleCell.Value = Replace(SingleCell, "‶", Chr(34))
    Next SingleCell

    Application.ScreenUpdating = True
End Sub
```