

Modelling In Excel, How To Teach an Old Dog, New Tricks

Paul Shannon, RTI Health Solutions, Manchester, UK

ABSTRACT

Microsoft Excel, for many years has been used as a generic tool in many industries for a wide range of analyses and modelling. Thanks, in part to the scope of Visual Basic for Applications (VBA); the programming language behind Excel's macro functionality. However, VBA has seen very limited development over the past decade. Consequently, this has resulted in model developers and users looking away from Excel in order to build a real-world, Graphical User Interface (GUI) for such models. Fortunately though, an old dog can learn new tricks.

It is now possible to build a GUI within Excel, based on web languages, giving the look and feel of GUI's a 21st century application, without using external software to achieve this.

INTRODUCTION

This paper looks at a new idea for extending the scope of VBA by adding the use of web technologies within VBA. Both the benefits and challenges faced in this topic are covered. The development of this paper (and its subject matter) has come from a health economic perspective. As a result, the examples in this paper also come from a similar angle. This does not imply that this health economics is the only relevant scope of the enclosed topic.

Since the introduction of VBA (and Excel 4 macros; VBA's predecessor), as a macro programming language, users have been able to customise Excel workbooks to make them look, feel and behave more like a custom-built application. Creating a purpose built window or 'userform' for a workbook suddenly became easier, allowing developers to display message boxes or dialogs containing project specific information. Using the same methods, it is even possible to completely replace the Excel model with a GUI; built within a userform containing standard windows 'objects', such as textboxes, dropdown selections (comboboxes), tick boxes etc.

However, issues with complexity and logistics can quickly arise. VBA is based on the Visual Basic (version 6) programming language and was first introduced into MS office in 1997. Since then, development of VBA has been somewhat conspicuous by its absence. The standard objects used in VBA such as textboxes, command buttons and tick boxes etc; come from a library called the "Microsoft Forms Library" (MS Forms), which itself is very dated. The properties and events available to the objects in this library are limited. As a result, the task of creating a GUI solely using VBA and MS Forms quickly becomes very labour intensive and time consuming.

With VBA not being developed through newer versions of Excel, model developers have started to look for alternative methods of developing a GUI. In particular, the rapid spread of web-orientated technologies throughout the turn of the century meant end users were looking for 'the web look' in GUIs. E.g.: items which can highlight when the user hovers the mouse over, thus providing, at least, an intelligent-looking solution through an advanced visual feedback to the end user. Achieving this using VBA can prove very time consuming and have limited scope. Consequently, stand-alone programs have often been developed to run an Excel model. This has meant an increase in the burden of work on professional programmers in order to build a modern, professional-looking model.

WHY DO EXISTING SOLUTIONS FALL SHORT?

The standard objects, such as userforms, textboxes, command buttons etc. seen in many every day applications are provided by the Microsoft Windows Common Controls library. The corresponding objects in VBA are powered by the MS Forms library. Unfortunately, whilst these objects may appear identical in a visual capacity, the functionality (from a programmer's perspective) of MS Forms objects is more limited. Comparing the MS Forms library to the Windows Common Control library reveals that MS Forms has fewer events available and has less powerful properties. Furthermore, VBA does not support object arrays, this makes programming the same commands into multiple objects, very long-winded. One example of VBA's shortcomings is highlighted in the table below. This table lists the events available in an MS Forms textbox and an HTML textbox. In total, there are 15 events available in MS Forms, whilst 60 are available in HTML. This lack of events in MS Forms when compared to other programming environments, results in even simple features such as mouseover events are difficult to program.

FIGURE 1: TEXTBOX EVENTS AVAILABLE IN VBA (MS FORMS) AND HTML

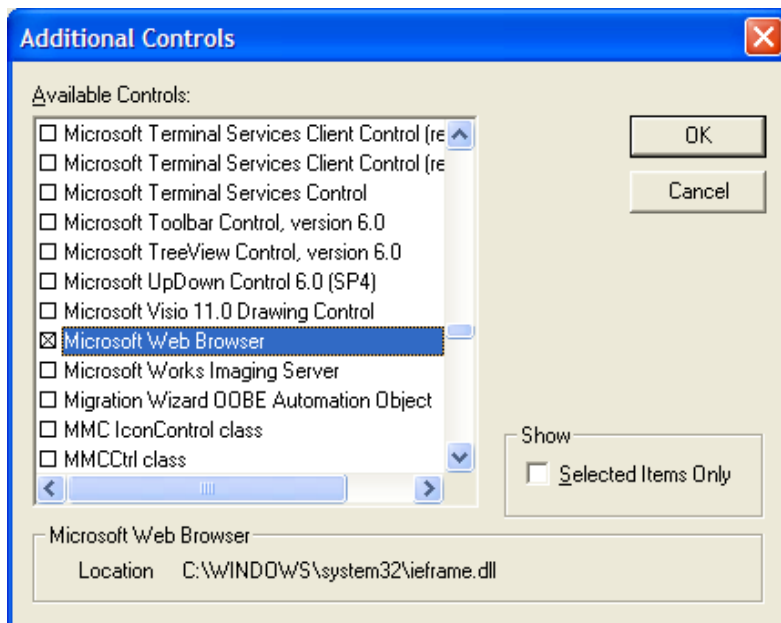
MS Forms Textbox	HTML Textbox			
AfterUpdate	onActivate	onDataBinding	onFocusOut	onMouseWheel
BeforeDragOver	onAfterUpdate	onDbClick	onHelp	onMove
BeforeDropOrPaste	onBeforeActivate	onDeactivate	onInit	onMoveEnd
BeforeUpdate	onBeforeCopy	onDisposal	onKeyDown	onMoveStart
Change	onBeforeCut	onDrag	onKeyPress	onPaste
DbClick	onBeforeDeactivate	onDragEnter	onKeyUp	onPreRender
Enter	onBeforeEditFocus	onDragEnd	onLoad	onPropertyChange
Error	onBeforeEditPaste	onDragLeave	onLoseCapture	onReadyStateChange
Exit	onBeforeUpdate	onDragOver	onMouseDown	onResize
KeyDown	onBlur	onDragStart	onMouseEnter	onResizeEnd
KeyPress	onChange	onDrop	onMouseLeave	onResizeStart
KeyUp	onClick	onErrorUpdate	onMouseMove	onSelect
MouseDown	onContextMenu	onFilterChange	onMouseOut	onSelectStart
MouseMove	onCopy	onFocus	onMouseOver	onServerChange
MouseUp	onCut	onFocusIn	onMouseUp	onUnload

As a result, this makes the use of MS Forms objects in VBA for complex projects such as building a GUI more difficult. In many scenarios, anecdotal evidence of using these objects can often result in reaching the limit of their capability, at least in the capacity of the type of use which they were designed for. The frustrations caused by these limitations, coupled with the lengthy and repetitive nature of designing and in particular, programming a GUI using this technique, resulted in the search for a new approach.

OLD DOGS CAN LEARN NEW TRICKS

Somewhat ironically, having highlighted flaws in VBA, it is VBA's own flexibility which allows for a new methodology to be used. Userforms in Excel can contain other objects as well as those provided by default in the MS Forms library. In fact, most 'ActiveX' or 'COM' objects can be added to a userform as well (however, don't try to add any of the Windows common controls library, VBA won't let you). One such object, is the "Microsoft Web Browser" object which comes as standard with Microsoft Windows. (You'll find the file for this in your Windows System folder, called "ieframe.dll"). In principle this is akin to adding an Internet Explorer window into your GUI so you can display web pages. By adding a web browser control to a userform, *in theory*, any functionality possible on a web page is now possible in an Excel GUI. To add the relevant object, go to the Tools | Additional controls menu in VBA editor, and select the "Microsoft Web Browser" object as shown in Figure 2.

FIGURE 2: ADDING THE WEB BROWSER OBJECT TO VBA

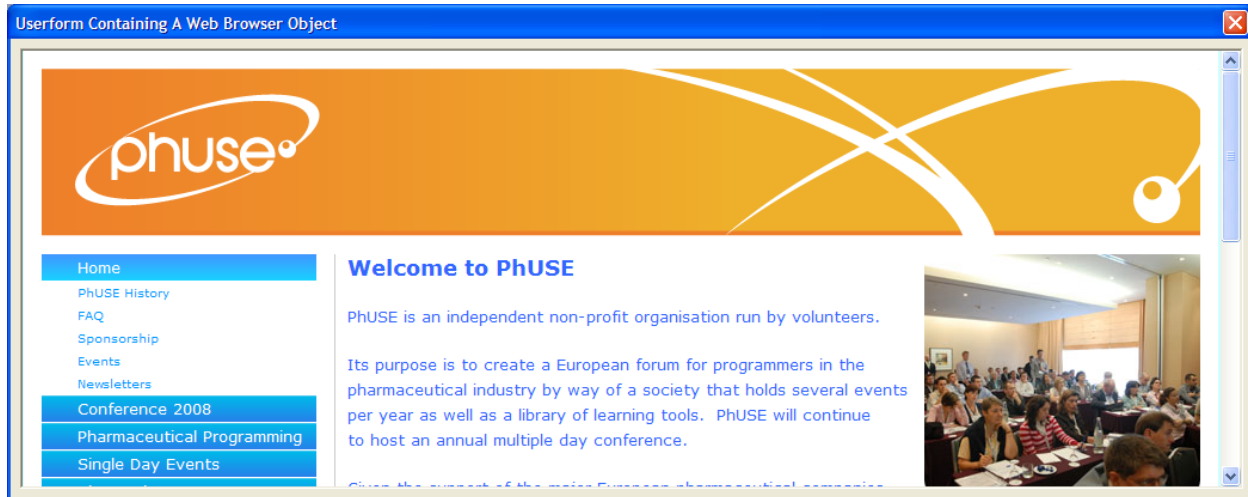


PhUSE 2008

ANYTHING INTERNET EXPLORER CAN DO...

The net effect of using a web browser object in VBA is that you can have an instance of Microsoft Internet Explorer embedded within your userform. This means the developer now has a whole new set of tools available. Anything which can be achieved in Internet Explorer, can now, in theory be achieved in VBA. As an example, figure 3 shows a basic VBA userform with a web browser object where the browser has navigated to the PhUSE website <http://www.phuse.info> (as accessed on 18 AUG 2008).

FIGURE 3: A USERFORM CONTAINING A WEB BROWSER OBJECT



The corollary of this is that, as a developer, the need now is to think about creating web pages, containing the graphics, text, inputs and outputs required for a GUI rather than merely building objects in VBA. Web-pages however, can contain many different languages; Hyper Text Markup Language (HTML) JavaScript and VBScript to name a few. Therefore, it appears a new skill set is needed to develop such pages, and the level of dependency on the programmer is apparently increasing. Furthermore, VBA is by no means now a redundant programming language, far from it, in-fact. Whilst web languages such as HTML and JavaScript can be used to generate web pages which will be shown in the Web Browser Control, this Web Browser control is still an object, placed on a userform, running essentially as an Excel macro. Therefore, there is still a quantity of VBA code used to make the GUI work.

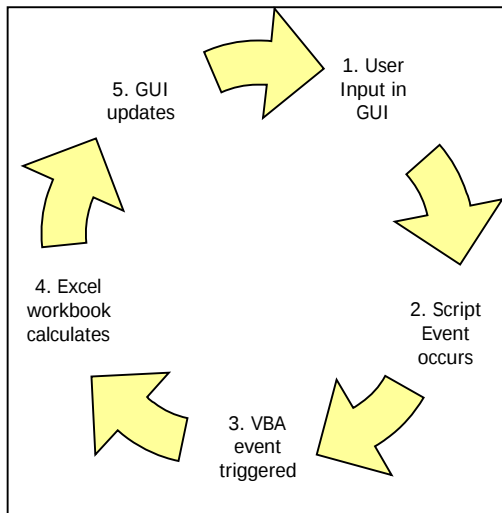
CREATING HTML FROM AN EXCEL WORKBOOK

It may seem, therefore, that whilst the graphical appearance of our GUI can now be improved, the complexity in achieving this is increased, and the earlier promise of a reduced burden of time on the programmer appears to be a long way away. The ideal solution requires standardised communication methods between the HTML pages in the web browser, and the underlying Excel model.

Consider a web page, shown as a page of a GUI; which contains a series of (e.g.) textboxes, which refer to a series of cells in the underlying Excel model. When the contents of a textbox is changed by the user, this change needs to be passed back to Excel, which needs to update the corresponding cell, recalculate the workbook as necessary, then feed the newly calculated cell values (and any other relevant changes, e.g.: chart updates, conditional formats etc.) back to the GUI (as shown in figure 4). Therefore, a series of standards must be used to ensure reliable method of communication between the web pages in a GUI and the underlying Excel workbook.

The diagram below (Figure 4) demonstrates the flow of two-way communication between an HTML page in a GUI and the underlying Excel workbook. The major challenge with this solution was to ensure a reliable method of two-way communication could be established between the GUI and Excel. VBA is still used to ensure this communication can take place effectively.

FIGURE 4: FLOW OF COMMUNICATION BETWEEN A GUI AND THE UNDERLYING EXCEL WORKBOOK



Any web browser object, in VBA has a series of events associated rather like any of the MS Forms objects. One such event is the StatusTextChange event. This event is triggered whenever the text in the status bar is changed. Even though the status bar is not visible in the Web Browser Control, it still exists, and can have its properties changed too. A simple JavaScript command can be used to change the status bar text. For example, to change the status bar text to "Hello World", you can use the following JavaScript:

```
top.status = "Hello World";
```

This would then automatically trigger the StatusTextChange event in VBA. The new status text is passed into the event as the Text argument. We can use this to pass commands from the HTML pages, through to the Excel document. The example below shows a standard JavaScript function to change the Web Browser's status bar text, when the value of an input tag is changed.

```
function inputChange(o) {
    var obj = document.getElementById(o);
    var val = new String();

    if (obj.type=='checkbox') {item=obj.checked;}
    else {item=obj.value;}

    top.status = "inputChange? " + obj.id + "? " + item;
}
```

As Excel refers to each cell within a given workbook in the form of 'sheet name'!\$R\$C, where R refers to the row letter and C refers to the column, a standard practice of applying this same naming method to HTML objects in the GUI would ensure that the identifier for each object as required for web pages would a) remain unique and b) always be comparable to a corresponding location in the Excel workbook. This now gives the information needed to enable a reliable communication between the HTML pages and Excel, ensuring no chance of mis-communication. I.e.: The AddressLocal property in Excel can correspond directly to the object ID property in an HTML document. Consequently, whenever an object is changed in the GUI, an event is triggered in the Web Browser Control; this triggers an event sub-routine in VBA. It is in this macro where Excel is updated and calculated.

The complexity of programming an HTML based GUI may seem greater than merely using VBA alone as now two programming languages have to be written to complete the job. Fortunately, by implementing a series of aforementioned programming standards, one pre-written macro can be called and used to update any cell when a corresponding item is changed in the GUI. Therefore it is less likely that further VBA will need to be written in any great detail. Taking this idea a step further, a series of pre-written VBA macros can be developed to cope with the majority of situations regularly seen in this type of solution; thus creating an 'engine' written in VBA to manage HTML events in the GUI. Now the emphasis of the development on a per-project scale is merely to develop the web pages based on set standards to show the GUI.

AUTOMATED CODE GENERATION

Previously, a programmer may be needed to build a GUI, but a non-programmer could at least build some elements of a GUI, should they have experience of writing or even just recording macros in Excel. Now, at this point, the development lies predominantly in the hands of a web developer. Fortunately, we can use Excel to create much of the HTML and JavaScript code automatically, using simple VBA functions. Figure 5 shows two simple

PhUSE 2008

VBA functions, used together to generate an HTML paragraph (<p> tag). Any Excel user can call this function, in a workbook. The returning value will be HTML code.

FIGURE 5: EXAMPLES OF EXCEL FUNCTIONS TO GENERATE HTML CODE

```
Public Function AlignH(r As Range) As String
    Select Case r.HorizontalAlignment
        Case -4108: Align = "center"
        Case -4152: Align = "right"
        Case Else: Align = "left"
    End Select
End Function

Public Function ParagraphTAG(r As Range) As String
    'Write opening tag with CSS style property
    ParagraphTAG = "<p align=" & AlignH(r) & _
        " style='font-family:" & r.Font.Name & ";" & _
        " font-size:" & r.Font.Size & "pt;'>"

    'Write InnerHTML and closing tag.
    ParagraphTAG = ParagraphTAG & r.Text & "</p>"
End Function
```

USAGE:

```
=ParagraphTag(sheet1!$B$2)
```

RETURNS:

```
<p align=center style='font-family:Arial; font-size:10pt;'>Drug Costs:</p>
```

This example works adequately for writing static text as HTML code. However, an Excel model will inevitably have a series of inputs which need to be edited by the user and outputs which change dynamically as a result. In HTML, there are two types of Inputs, the <input> tag and the <select> tag. The latter generates an HTML combo box, whilst the former can take the form of a textbox, a checkbox, a radio button etc. We can read various properties in the Excel object library to automatically determine exactly what function in the model, each cell is performing (e.g.: input, output, static text).

```
Public Function GetValidation(r As Range) As Long
    On Error GoTo ErrHandler:
    GetValidation = r.Validation.Type
ErrHandler:
    GetValidation = 0
End Function

Public Function InputTAG(r As Range) As String
    'If cell value is boolean, assume a tickbox should be used.
    If VBA.UCase(r.Text) = "TRUE" Or VBA.UCase(r.Text) = "FALSE" Then
        InputTAG = "<input type=checkbox id='" & _
            r.AddressLocal & "' value=" & _
            r.Value & _
            " onclick=javascript:inputChange('" & r.AddressLocal & "');
    />"

    'If validation type is 3, then a list is present, so use a <select> tag.
    ElseIf GetValidation(r) = 3 Then
        InputTAG = SelectTAG(r)

    ElseIf r.Locked = False Then
        InputTAG = "<input type=text id='" & r.AddressLocal & _
            "' value=" & r.Text & _
            " onblur=javascript:inputChange('" & r.AddressLocal & "'); />"

    'If the cell is not an input, but has a formula, consider it an output,
    'Therefore, it needs to be in it's own tag and assigned its own id.
    ElseIf r.HasFormula = True Then
        InputTAG = "<font id='" & r.AddressLocal & "'>" & r.Text & "</font>"
    End If
End Function
```

PhUSE 2008

```

Else
    InputTAG = r.Text
End If
InputTAG = InputTAG & VBA.Chr(10)
End Function

Public Function SelectTAG(r As Range) As String
Dim rList As Range, i as integer

If r.Validation.Type <> 3 Then Exit Function      'Include some error trapping
On Error GoTo ErrHandler:

'Open the <select> tag and get the source range for the list of responses
SelectTAG = "<select id='" & r.AddressLocal & _
    "' onchange=javascript:cmbChange('" & r.AddressLocal & "');>"
Set rList = Range(Mid(r.Validation.Formula1, 2))

'Loop through each item and write an <option> tag
For i = 1 To rList.Rows.Count
    SelectTAG = SelectTAG & "<option " & _
        IIf(r.Text = rList.Cells(i, 1).Text, "selected ", "") & ">" & _
        rList.Cells(i, 1).Text & "</option>"
Next i
'Close the tag
ErrHandler:
    SelectTAG = SelectTAG & "</select>"
End Function

```

USAGE:

=InputTAG(Sheet1!\$E\$4)

RETURNS:

<input type=text id='Sheet1!\$E\$4' value=£321.69
onblur=javascript:inputChange('Sheet1!\$E\$4') />

We now have functions which can write formatted paragraphs, or create input cells dynamically based solely on settings within Excel. For example, we can decide whether or not to create an <input> tag based on the value of the 'locked' property, for a given cell. Similarly, we can tell whether to create an <input> tag or a <select> tag based on the data validation type for a given cell. This gives just a taster as to how we can create comparable HTML code based on the settings for a particular cell or range of cells.

Taking this same principle a step further, Figure 6 shown below is a short example of an Excel model used to calculate the overall cost to a health service of administering a particular drug. In this example, the table calculates the prescription cost for a course of treatment for two drugs; Drug A and drug B.

FIGURE 6: COST ANALYSIS EXAMPLE IN EXCEL

	A	B	C	D	E	F
1						
2						
3						
4						
6						
8						
10						
12						

Drug Costs:		
	Drug A	Drug B
Drug Cost per dose	£591.36	£174.12
Doses per day	3	4
Days of Treatment	7	28
Total Cost	£12,418.56	£19,501.44
Drug A costs £7,082.88 less than Drug B per treatment		

This table has been formatted to appear as would be expected in an Excel workbook. We can use the functions defined above to recreate this table dynamically in an HTML document. But let's look at what's involved in this process. The cell range used is from B2 to F12. This is a total of 55 cells. Each of which will need a particular

PhUSE 2008

formula to be recreated. So at present, it doesn't seem to be a massive time saver. The solution? Functions to call the functions.

The ideal solution here (as is often the case when creating HTML based on Excel), is to create an HTML <table> tag to contain the above section. The rows and cells in Excel easily correspond to <tr> and <td> tags in HTML, which can also hold the relevant formatting. We can then write the relevant <input> tags into each cell as required.

```
Public Function TableTAG(r As Range) As String
Dim i As Integer

    TableTAG = "<table cellpadding=0 cellspacing=0>"      'Open HTML tag
    On Error GoTo ErrHandler:                            'Set error handler

    'Loop through each row in selected range, calling ROW function
    For i = 1 To r.Rows.Count
        TableTAG = TableTAG & TableROW(r.Rows(i))
    Next i

ErrHandler:      'Close HTML tag
    TableTAG = TableTAG & "</table>"
End Function

Public Function TableROW(r As Range) As String
Dim i As Integer

    TableROW = "<tr>" & VBA.Chr(10)                      'Open table ROW <tr> tag
    On Error GoTo ErrHandler:                            'Set error handler:

    'Loop through each cell in the specified row
    For i = 1 To r.Columns.Count
        TableROW = TableROW & TableCell(r.Cells(1, i))
    Next i

ErrHandler:      'Close tag
    TableROW = TableROW & "</tr>" & VBA.Chr(10) 'Add a line break in code
End Function

Public Function TableCell(r As Range) As String
Dim rMerge As Range

    On Error GoTo NoMerge:      'Control for merged cells, we only want to
    Set rMerge = r.MergeArea    'use the first cell in a merged area
    If rMerge.Cells(1, 1).AddressLocal <> r.AddressLocal Then Exit Function
    GoTo OpenCell:

NoMerge:
    Set rMerge = r

OpenCell:
    'Open the <td> tag. Write dimensions into tag
    TableCell = "<td class='" & r.Style.Name & "' colspan=" & rMerge.Columns.Count
    _
    & " rowspan=" & rMerge.Rows.Count & " width=" & rMerge.Width & "pt height=" &
    _
    rMerge.Height & "pt>"

    TableCell = TableCell & InputTAG(r) 'Now call InputTAG

ErrHandler:      'Close tag
    TableCell = TableCell & "</td>" & VBA.Chr(10) 'Add a line break in code
End Function
```

As Excel already contains the desired formatting, we can use the functions above to generate the code for an HTML

PhUSE 2008

table similar to that shown in Figure 6. The formula required can be as simple as:

USAGE:

```
=TableTAG(sheet1!B2:F12)
```

Not only can this be used to quickly generate the HTML to build a GUI, but it can ensure that the code written will conform to the standards required by the GUI's engine. More importantly, however, this allows for a massive time saving on the part of the programmer. Now, anyone with the required knowledge of code generating functions can build at the very least, a basic GUI.

Another example of automated code generation includes the ability to allow Excel functionality to be replicated in the GUI. Such an example could be that data validation in Excel can be automatically read by the same function above, to ensure for example, that only positive values are entered in the cells. This can be carried through to the GUI in the form of customised error messages should the validation rules be breached.

To give an idea of how this can save time, the resulting code from the above example would be:

```
<table cellpadding=0 cellspacing=0><tr>
<td class='Title' colspan=1 rowspan=1 width=122.25pt height=12.75pt>Drug
Costs:</td>
<td class='Normal' colspan=4 rowspan=1 width=132pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=12.75pt></td>
<td class='Title' colspan=1 rowspan=1 width=62.25pt height=12.75pt>Drug A</td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
<td class='Title' colspan=1 rowspan=1 width=62.25pt height=12.75pt>Drug B</td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=12.75pt>Drug Cost per
dose</td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
<input type=text id'$C$4' value=£591.36 onblur=javascript:inputChange('$C$4');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
<input type=text id'$E$4' value=£174.12 onblur=javascript:inputChange('$E$4');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=12.75pt>Doses per
day</td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
<input type=text id'$C$6' value=3 onblur=javascript:inputChange('$C$6');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
<input type=text id'$E$6' value=4 onblur=javascript:inputChange('$E$6');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
</tr>
</tr>
```

PhUSE 2008

```
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=12.75pt>Days of
Treatment </td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
  <input type=text id='$C$8' value=7 onblur=javascript:inputChange('$C$8');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
<td class='Input' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
  <input type=text id='$E$8' value=28 onblur=javascript:inputChange('$E$8');
/></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=12.75pt>Total
Cost</td>
<td class='Output' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
  <font id='$C$10'>£12,418.56</font></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
<td class='Output' colspan=1 rowspan=1 width=62.25pt height=12.75pt>
  <font id='$E$10'>£19,501.44</font></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=12.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=1 rowspan=1 width=122.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=62.25pt height=3.75pt></td>
<td class='Normal' colspan=1 rowspan=1 width=3.75pt height=3.75pt></td>
</tr>
<tr>
<td class='Normal' colspan=5 rowspan=1 width=254.25pt height=19.5pt>
  <font id='$B$12'>Drug A costs £7,082.88 less than Drug B per
treatment</font></td>
</tr>
</table>
```

SUMMARY

This paper only scratches the surface of how the Microsoft Web Browser control can be used in Excel. Other aspects of Excel which may be considered for automatic HTML generation might be:

- Styles – perhaps as a Cascading Style Sheet
- Charts – Either as a static image, or as an interactive object
- Hyperlinks

It is also worth noting that using VBA to write syntax for another language is not limited to HTML. In fact, exactly the same technique can be applied to generate XML or even RTF syntax, should the need arise in any given solution.

CONCLUSION

Microsoft's own macro language in Excel (VBA) is rather out-dated for the purpose of building a modern GUI. However, at the same time, the flexibility of VBA allows it to be combined with the 21st century technologies of web development. This hybrid of different languages has resulted in a technological creep which provides a whole new scope to solve a long existing problem.

A Web Browser Control can be used in VBA providing HTML capability to replace the need for using the standard MS Forms library objects. This enables custom-built windows and dialogs in Excel to have a modern feel as well as providing a much improved level of interaction with the end user. Furthermore, HTML code can be automatically generated using VBA. This provides a level of automation in the development of a GUI.

Whilst increasing the perceived complexity in the underlying technology, the ease and scope within the reach of programmers and non-programmers alike, when developing a GUI under real-world time constraints is now greater

PhUSE 2008

than ever. This methodology, however, does not aim to replace the need for a programmer, but to reduce the burden of complete dependency on programmers as well as repetition of common tasks for such a project by sharing the development process and reducing the timelines required for such a project.

REFERENCES

www.phuse.info, accessed 18th August 2008

Pure Javascript, Wyke R. A., Gilliam J. D., Ting C.; Sams Publishing; 1999

Microsoft Office Excel 2003 Power Programming With VBA; Walkenbach J.; Wiley Publishing Inc, 2004.

CONTACT INFORMATION

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

Paul Shannon
RTI Health Solutions
Williams House
Manchester Science Park
Lloyd St North
Manchester
M15 6SE
Tel: +44 (0) 161 232 4926
Fax: +44 (0) 161 232 3409
Email: pshannon@rti.org
Web: www.rtihs.org