WinMark Pro Application Note

Integrating the WinMark Pro ActiveX Control into an Microsoft® Excel® Document

This Application Note provides introductory information about using the WinMark ActiveX control in a Microsoft Excel document. Use the information in this note as a supplement to:

- the *ActiveMark™ Technology* section of the *WinMark Pro User's Guide*
- WinMark Pro's ActiveX help file (available by selecting *WinMark Pro ActiveX Control Help* from WinMark's *Tools* menu)
- the information contained in our Laser Marking FAQ on the WinMark Pro website

The information contained in this Application Note is laid out as a tutorial. Topics are organized so that the concepts are very basic at first, increasing in complexity as you progress.

This Note covers the following topics:

Definitions

The following terms will be used throughout this Application Note, and are meaningful within Excel as well as in WinMark Pro. To fully understand the use and implementation of the WinMark ActiveX control, please review these definitions carefully.

Control
A *Control* is an object that is placed on a worksheet in Excel. Command buttons, text boxes, and labels are all examples of *Controls*.

Property
A Control's *Property* determines something about its appearance or behavior. For instance, ForeColor and BackColor are typical *Properties* that define the color attributes of many controls.

Method
Most controls also have *Methods*, which are actions that can be taken on the Control. For instance, most controls have a Refresh *Method* that is used to update the display of the control's contents.

ActiveX
While Excel contains a set of built-in VBA controls like the command button, text box, list box, etc., additional controls are available as stand-alone files with the file extension ".ocx". These controls use *ActiveX* technology to allow their use in an Excel document so that they look and feel like the built-in controls.

WMP
WinMark Pro, the custom Synrad laser marking software suite that provides the Drawing Editor interface, the Launcher interface, as well as the *SynMhAtx.ocx* ActiveX control.

VBA
Visual Basic for Applications, a subset of the Visual Basic software design tool that is built into the Microsoft Office Suite. VBA allows customization of Office applications to a degree that extends beyond Macro functions, etc.

Workbook, Worksheet
Microsoft Excel organizes XLS files as *Workbooks* filled with individual *Worksheets*. The VBA code you will develop through the course of this App Note will be configured to act globally on the entire Workbook as well as on the individual sheets.

Getting Started

This section will explain how to add the WinMark ActiveX control to the Microsoft Excel application.

1 Install WinMark on the computer that you will use to design the application.

2 Add the WinMark ActiveX control to the Excel worksheet:

a Open Excel.

b Right-click in the toolbar and select *Control Toolbox*. You should see the VBA control toolbox shown in Figure 1:

Figure 1 Excel's VBA Control Toolbox dialog

Note the *Design Mode* icon in the upper left of the Toolbox (the Protractor and Pencil symbol). You can toggle Design Mode on and off to control the behavior of your application. When this toggle button is activated (depressed as shown), you can edit the size and function of the controls used in your Excel application. When the button is released to exit Design Mode, WinMark Pro's ActiveX control displays the *Drawing Canvas*, and controls within the application are active.

By default, all design instructions throughout this App Note will assume that the Design Mode button is activated.

c Click on the *More Controls* icon to the lower right of the Toolbox (the Wrench and Hammer symbol).

d Scroll through the list of controls and find *Synrad WinMark Pro Control*.

That's it. You're now ready to build a Synrad WinMark Pro VBA marking interface.

A Simple Marking Application

This section shows you how to create a basic application that loads a text object into the WinMark ActiveX control and then lases the object (if you have a marking system connected).

1 Open a new workbook and verify that Sheet1 is selected and that the *Control Toolbox* is visible.

2 Scroll through the list of controls and click on *Synrad WinMark Pro Control*.

3 Place the WinMark ActiveX control on the worksheet by clicking and dragging the mouse cursor to size the border of the ActiveX control. Size the control appropriately, since the control will be displayed as a preview window while the marking file is manipulated within the VBA code.

Note: Once the control has been placed on the worksheet, you can resize it as required while in design mode.

4 Select the WinMark ActiveX control and click on the *Properties* button on the *Control Toolbox*. Set the WMP ActiveX control's properties as follows:

Name: **mh**

5 Place a command button on the worksheet. Set the command button's properties as follows:

Name: **cmdMark**

Caption: **Mark**

6 Click on the Control Toolbox's *View Code* button to bring up the VBA code window.

7 Select 'Project Explorer' from the *View* menu. The project explorer window will open and look something like Figure 2:

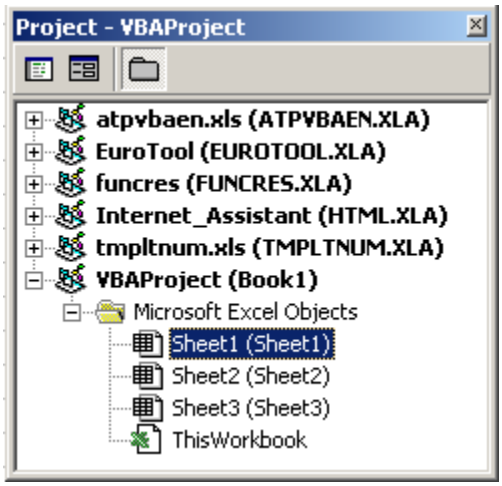

Figure 2 Excel's Project Explorer window

8 Double click on *ThisWorkbook* to bring up the workbook code window.

9 In the Object dropdown box (the upper left corner of the code window), select *Workbook*. This inserts a code header and footer for the `Workbook_Open()` event, which will execute every time the workbook file is opened (Figure 3).

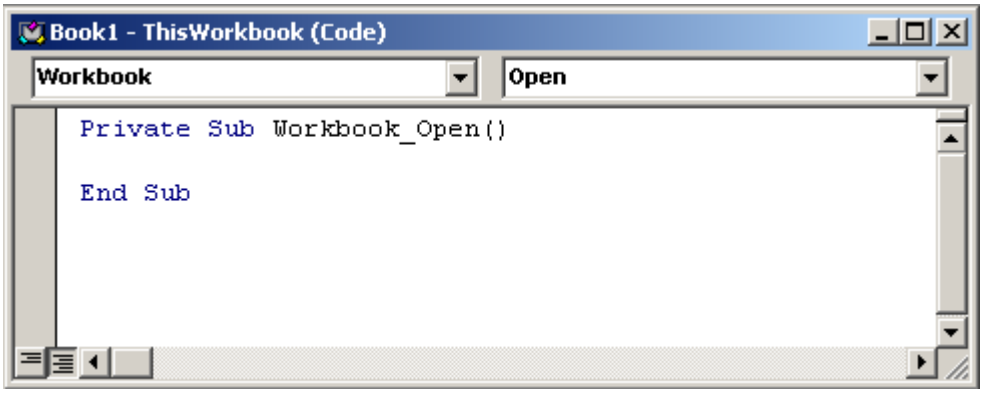

Figure 3 Code Header and Footer

10 In the `Private Sub Workbook_Open()` event code section, enter the following:

```
Sheet1.mh.AddText "Text1", 0, 0, "My Text Object"
Sheet1.mh.Redraw
```

Note that the `Sheet1` *object identifier must be added to the* `mh.AddText` *and* `mh.Redraw` *commands, since they are being called from code outside of Sheet1.*

11 Go back to the Worksheet view and double-click on the command button to bring up the `cmdMark_Click` event code section.

12 In the `Private Sub cmdMark_Click()` event code section, enter the following:

```
mh.MarkDrawing
```

13 Save the file, then close and reopen it.

Depending on your Macro Virus Protection settings in Excel, you might see the ubiquitous Excel Macro warning window when the file starts up. This warning is generated because you are opening a file that has VBA executable code built into it. Press *Enable Macros* to allow the code you've written to execute. If this warning box is too much of a nuisance, you can modify your security settings to prevent the alerts.

14 Once the file has opened, you will see the text object "My Text Object" displayed on the drawing canvas in the ActiveX preview window.

15 If the development computer is connected to a marking system, you can now press the *Mark* button on the application worksheet, and see the words "My Text Object" printed by the laser.

That's your first Excel marking application! Save the file for use in the subsequent sections of this Application Note.

Manipulating the Properties of the Text Object

In the course of completing this section, you will expand the basic application to allow manipulation of the Text object's caption, size, marking parameters, and other properties.

This section illustrates the use of the WinMark Pro ActiveX control's property manipulation methods.

You can modify any of the properties of any of the objects in a mark file by using the appropriate 'Set Property' method:

SetBoolProp	sets any property that has a Boolean value, such as the *MarkObject* property (on/off or yes/no: 1 = On or Yes, 0 = Off or No)
SetIntProp	sets any property that has an integer value (such as the *Drawing's MarkCount* property)
SetFloatProp	sets any property that has a floating-point value (such as a text object's *TextHeight* property)
SetStringProp	sets any property that has a string value (such as the *TextCaption* property of a text object)

In the same way, you can extract the property value of any object in a mark file by using the appropriate 'Get Property' method:

GetBoolProp	sets any property that has a Boolean value (on/off or yes/no – 1 = on or yes, 0 = off or no; such as the *MarkObject* property)
GetIntProp	sets any property that has an integer value (such as the *Drawing's MarkCount* property)
GetFloatProp	sets any property that has a floating-point value (such as the *Drawing's FieldWidth* property)
GetStringProp	sets any property that has a string value (such as the *TextCaption* property of a text object)

Note: The *SetStringProp* and *GetStringProp* methods have a nice built-in feature – they automatically convert any values to the appropriate data type (Boolean, Integer or Floating Point) when used on non-string object properties. You can conceivably use these methods on all property types without worrying about the required data type (although this might lead to some confusing code).

For instance, any object may be set to not mark by executing the following code:

```
mh.SetBoolProp "Object1", "MarkObject", 0
```

The *SetStringProp* can also set this Boolean property:

```
mh.SetStringProp "Object1", "MarkObject", "0"
```

Property name strings recognized by the ActiveX control *do not* contain spaces, as opposed to the human-readable property names listed in the WinMark Pro interface. To obtain a list of objects in a mark file, their accepted property names, and their current property values, select *Print All Properties* from the *File* menu in WinMark. You can also use the *WritePropListToTextFile* ActiveX method to write the drawing and object property names and values to a text (.txt) file from your VBA application.

1 Open the Excel application saved in the previous section. Open the *Controls Toolbox* if it's not visible, select Sheet1 as the current worksheet, and click the *Design Mode* button to edit the application.

2 Add a Text Box to the worksheet. Set the text box's properties as follows:

Name: **txtCptn**

Text: **Enter caption here**

3 Double click on the text box to bring up the code design window. In the `Private Sub txtCptn_Change()` event code section, enter the following:

```
mh.SetStringProp "Text1", "TextCaption", txtCptn.Text
mh.Redraw
```

4 Exit design mode (by pressing the *Design Mode* button to toggle it out), and you will see the modified Basic Marking worksheet. Highlight the text in the text box and type in new text. Notice that the *Text1* object in the WinMark ActiveX control's preview window is updated to show the new text as each character is typed in.

5 Add a command button to the worksheet. Set its properties as follows:

Name: **cmdBigger**

Caption: **Text Larger**

6 Add a command button to the worksheet. Set its properties as follows:

Name: **cmdSmaller**

Caption: **Text Smaller**

7 Resize the buttons as required so that the operator can recognize the captions.

8 Double click on the *cmdBigger* command button to bring up the code design window. In the `Private Sub cmdBigger_Click()` event code section, enter the following:

```
Dim fltHeight As Double
   fltHeight = mh.GetFloatProp("Text1", "TextHeight")
   mh.SetFloatProp "Text1", "TextHeight", (fltHeight * 1.5)
   mh.Redraw
```

9 Double click on the *cmdSmaller* command button to bring up the code design window. In the `Private Sub cmdSmaller_Click()` event code section, enter the following:

```
Dim fltHeight As Double
   fltHeight = mh.GetFloatProp("Text1", "TextHeight")
   mh.SetFloatProp "Text1", "TextHeight", (fltHeight/1.5)
   mh.Redraw
```

10 Save the file and exit *Design Mode*. Press the *Text Larger* and *Text Smaller* command buttons and notice the way the text size changes. The nature of the resizing code is proportional – increasing *Text Height* also increases the *Text Width* to maintain the aspect ratio of the text.

The operator now can modify the text caption and resize the text object.

Loading a Mark File into the Application

Allowing an operator to enter and resize text is interesting and makes for a good introduction to the joys of designing a VBA marking application in Microsoft Excel, but the result is not very practical in the industrial sense. It would be a painful process indeed to create a mark containing twenty-five objects of various types and setting their location, *Power*, *Velocity* and other mark parameters using only VBA code.

A better option is to design and test the mark file in the WinMark Pro Drawing Editor, save and load the file into your application, and then modify only the objects that change from one mark to the next (like a serial number or part number value).

This approach becomes even more efficient if the application will be used on a production line where 2 or more (even 150 or more) different part configurations will be marked at various times, where the parts may change every shift or even every mark.

1 First, open WinMark and create a mark file.

   a For the sake of this exercise, add a 1D barcode, a 2D barcode, and a text object to the file.

   b Verify that the objects are named “Barcode1”, “2D Barcode1”, and “Text1”.

   c Save the file as “ActiveX Test.mkh” in the same folder that contains your Excel tutorial file.

2 Open the file saved in the last tutorial section. Modify the `Private Sub Workbook_Open()` event code section, commenting out the *Sheet1.mh.AddText* line and adding the *Sheet1.mh.LoadDrawing* line:

```
‘Sheet1.mh.AddText “Text1”, 0, 0, “My Text Object”
Sheet1.mh.LoadDrawing (ThisWorkbook.Path & "\ActiveX test.mkh")
Sheet1.mh.Redraw
```

   Note: The advantage of using the `ThisWorkbook.Path` property is that it allows greater freedom in the installation of the Excel application and the associated mark files. The files do not need to be on a particular drive, they just all need to be in the same folder.

3 Save the file, then close and reopen it.

   The file created in WinMark is shown in the ActiveX preview window (if it isn’t, the mark file is either not in the same folder as the Excel file, or the filename differs from what was entered above).

   Notice that you can enter data in the *txtCptn* Text Box and see it applied to the text object. When modifying an object’s properties, it does not matter whether the object was created within the ActiveX application, or is a member of a pre-existing mark file that has been read into the ActiveX control.

4 Click the *Design Mode* button and double click the *txtCptn* Text Box to bring up the code for the `txtCptn_Change()` event. Add two more *SetStringProp* commands so that the event contains the following lines of code:

```
mh.SetStringProp “Barcode1”, "BarcodeNumber", txtCptn.Text
mh.SetStringProp “2D Barcode1”, "2DBarcodeText", txtCptn.Text
mh.SetStringProp “Text1”, "TextCaption", txtCptn.Text
mh.Redraw
```

5 Start the application and notice that when you enter a new value in the *txtCptn* Text Box all three objects change values. Already you may realize how you could apply a single variable value to multiple objects in a mark file.

Loading Multiple Mark Files into the VB Application

1 Create two more mark files.

The files should be copies of the *ActiveX Test.mkh* file, but move the objects around so that you can recognize one file from another in the preview window. Name the files *ActiveX Test2.mkh* and *ActiveX Test2.mkh*.

2 Add a list box to the worksheet. Set its properties as follows:

Name: **lstFile**

3 Modify the `Private Sub Workbook_Open()` event code section, commenting out the `mh.LoadDrawing` line and adding the *lstFile* configuration lines:

```
'Sheet1.mh.AddText "Text1", 0, 0, "My Text"
'Sheet1.mh.LoadDrawing (ThisWorkbook.Path & "\ActiveX test.mkh")
Sheet1.lstFile.AddItem "ActiveX Test"
Sheet1.lstFile.AddItem "ActiveX Test2"
Sheet1.lstFile.AddItem "ActiveX Test3"
Sheet1.lstFile.ListIndex = 0
```

4 Double click on the list box to bring up the code design window. In the `Private Sub lstFile_Click()` event code section, enter the following:

```
mh.LoadDrawing ThisWorkbook.Path & "\" & lstFile & ".mkh"
mh.Redraw
```

5 Save the file, close it, and reopen it.

You should see that the original *ActiveX Test.mkh* file loads into the control at startup. Click on the other files and verify that the preview window shows the correct file.

This is all very nice, but the ActiveX control must load another mark file every time the list box selection changes, even if the new selection has already been loaded once. A drawback to this method is that loading and reloading the files, over time, can gobble up all of the PC's GDI resources. Then the Operating System crashes and burns, which is never a good thing.

A better way to do this is to use the *mh.DrawingIndex* property, which allows you to load all of the files of interest, then call up the desired file by simply calling its index. This places no burden on the GDI resources and can speed the process up considerably when using large intricate files. Continue on to the next section of this tutorial to understand how this process works.

Using the DrawingIndex to Manage Multiple Mark Files in the Application

1 To modify our application to use this preferred method, go back to *Design Mode* and open the workbook's code page.

2 Modify the `Private Sub Workbook_Open()` event code section by adding a Dimension statement and the For…Next loop:

```
Private Sub Workbook_Open()
Dim I As Integer
   'mh.AddText "Text1", 0, 0, "My Text"
   'mh.LoadDrawing ThisWorkbook.Path & "\ActiveX Test.MKH"
   With Sheet1.lstFile
      .AddItem "ActiveX Test"
      .AddItem "ActiveX Test2"
      .AddItem "ActiveX Test3"
   End With
   For I = 0 To 2
      Sheet1.mh.DrawingIndex = I
      Sheet1.lstFile.ListIndex = I
      Sheet1.mh.LoadDrawing ThisWorkbook.Path & "\" & lstFile & ".mkh"
   Next I
   Sheet1.lstFile.ListIndex = 0
   Sheet1.mh.Redraw
End Sub
```

Note the use of the *With…End With* code to ease the typing burden on the *Sheet1.lstFile* configuration code and to make the setup code easier to read.

3 Modify the `Private Sub lstFile_Click()` event code section, commenting out the *LoadDrawing* command and adding the *DrawingIndex* reference:

```
'mh.LoadDrawing ThisWorkbook.Path & "\" & lstFile & ".mkh"
mh.DrawingIndex = lstFile.ListIndex
mh.Redraw
```

4 Run the code and notice that, to the user, the code functions the same.

Although this improved method requires a bit more code up front, the long-term stability of your application will benefit greatly from its use.

Handling Digital I/O within the VB Application Code

If you've used WinMark Pro long enough, you may have noticed that sometimes the ESC key must be pressed several times before it is recognized. This is because keyboard activity is given a lower priority than that asserted by the WinMark marking engine while vector and laser control data is being generated during the mark process (you really don't want the Operating System wandering off to perform some other task while the laser is on, do you?).

In this same way, when the ActiveX *MarkDrawing* method is called, the marking engine takes control of the computer's system resources. Control is not passed back to the calling routine until the mark is complete. If the mark file being called by the *MarkDrawing* method contains Input/Output (I/O) automation, and the required input condition is never met, control will never return to the application and the system will appear to lock up. For this reason, it is best to move all I/O automation from the mark file to the VBA code. By so doing, your code can be written to accommodate stuck or missing input conditions.

To wait until Input Bit 0 (IN0) is set, or active, use the *GetDigitalBit* method in a *Do* loop:

```
Do While Not (mh.GetDigitalBit(0))
    DoEvents
Loop
```

Note: If you want to wait until IN0 is cleared, you just have to remove the *NOT* condition from the statement.

The *DoEvents* statement allows the application to respond to events that might be going on in the PC (ESC key press, COM port activity, etc.) while waiting for the required input condition.

Setting or clearing an output bit is done using the *SetDigitalBit* method:

```
mh.SetDigitalBit 4, True
```

These commands may be used in the *cmdMark_Click* procedure, so that when the button is pressed, the IN0 input must go active, then inactive, to generate one mark loop. This code may then be put within an overall loop to generate multiple marks:

```
Private Sub cmdMark_Click()
   'change the mark button caption to indicate that marking is in progress
   cmdMark.Caption = "Please Wait"

   'set Output4 to indicate that the marking system is ready
   mh.SetDigitalBit 4, True

   'wait for GO command on input 0 (input set = mark)
   Do While Not (mh.GetDigitalBit(0))
      DoEvents
   Loop
```

```
'*********************************************************************
'** Place code here to manipulate objects... update SN values, etc.
'*********************************************************************
   'clear Output4 to indicate that the laser is on
   mh.SetDigitalBit 4, False

   'update the display and mark
   mh.Redraw
   mh.MarkDrawing

   'marking is done... clear output 4
   mh.SetDigitalBit 4, False

   'verify that the input bit is cleared to avoid looping twice
   ' on one GO command
   Do While (mh.GetDigitalBit(0))
      DoEvents
   Loop

   'update Mark button caption and go back to ready state
   cmdMark.Caption = "Mark"

End Sub
```

Note: Your application must be connected to a marking head and have IN0 automation connected to fully execute this mark loop.

Detecting Error Conditions in the MarkDrawing Method

The *MarkDrawing* method returns a Boolean (True or False) value that is True if the method returns without errors, False if errors were detected. One possible way to use the Boolean value is as follows:

```
Dim bError as Boolean, strTemp as String
   strTemp = “There was an error detected in the MarkDrawing method”
   bError = mh.MarkDrawing
   If Not bError then MsgBox strTemp
```

In WinMark Pro builds 3426 and above, the ActiveX control includes a *GetMarkErrorCode* method to provide information about mark errors. Call this method if the *MarkDrawing* method returns a False result:

```
Dim Err as String
   If not (mh.MarkDrawing) Then
   Select Case (mh.GetMarkErrorCode)
      Case 0
         Err = "User Abort"
      Case 1
         Err = "No Response From Head"
      Case 2
         Err = "Line Speed Too Fast To Finish"
      Case 3
         Err = "Fiber Link Overrun"
      Case 4
         Err = "Line Speed Too Fast - Missed Start"
      Case 5
         Err = "Encoder Continuity Error"
      Case 6
         Err = "Marking Head Not Ready"
      Case 7
         Err = "Bad End Of Mark Response"
      Case 8
         Err = "Marking Head Not Powered Up"
      Case 9
         Err = "Invalid Drawing"
      Case 10
         Err = "No Marking Head Response"
      Case 11
         Err = "Cannot Clear Test Mark Mode"
      Case 12
         Err = "Cannot Set Test Mark Mode"
      Case 13
         Err = "Test Mark Transmission Error"
      Case 14
         Err = "Test Mark Memory Full"
      Case Else
         Err = "Unknown"
   End Select
   MsgBox "The following error was detected:" & vbCr & vbCr & Err
   End If
```

To test this code, try running it while commanding a mark when the marking head is powered off.

Creating custom Polyline objects

Use the *AddPolyline* method to create custom polyline objects from an array of X-Y points. For instance, data from a vision system may be used to cut custom shapes on the fly. The following example creates an ellipse:

```
Private Type PLARRAY
   X As Single
   Y As Single
End Type
Dim Polyline() As PLARRAY
Dim T As Single 'angle in degrees
Dim E As Single 'eccentricity of ellipse 0 -> 1
Dim N As Integer 'number of co-ordinate pairs in array
Dim I As Integer

'Draw an ellipse
'formula: for t = 0 to 360 degrees,
'     0 < e < 1, x = COS(t),
'     y = ((1 - e^2)^.5)SIN(t)

   N = 72
   E = 0.5
   ReDim Polyline(N)
   For I = 0 To N
      T = (I * 2 / N) * (3.14159265358979) 'convert degrees to radians
      Polyline(I).X = txtScale.Text * Cos(T)
      Polyline(I).Y = txtScale.Text * (((1 - E ^ 2) ^ 0.5) *
      Sin(T))
   Next I
   mh.AddPolyLine "Polyline1", N + 1, Polyline(0).X, False
   mh.Redraw
```

Well, that about covers the basics of ActiveX marking application development in Microsoft Excel. For further examples, please check out the ActiveX samples available for download from our site at http://www.winmark.com/ActiveX_Sample_Files/activex_sample_files.htm