

# WinMark Pro Application Note

## Integrating the WinMark Pro ActiveX Control into a Visual Basic Application

This Application Note provides introductory information about using the WinMark ActiveX control in a Visual Basic application. Use the information in this document as a supplement to:

- the *ActiveMark™ Technology* section of the *WinMark Pro User's Guide*
- the ActiveX help file available by selecting 'WinMark Pro ActiveX Control Help' from WinMark's 'Tools' menu
- the information contained in the [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 through the document.

This Note **is not** intended to serve as an introduction to the Visual Basic programming environment. For detailed information on programming in Visual Basic, please refer to the documentation and/or help files supplied with your software. Also refer to the many websites and tutorials available for help with learning and using VB.

This Note covers the following topics:

[Definitions](#) – explanation of terms related to designing with ActiveX controls

[Getting Started](#) – installation and registration of the Synrad WinMark Pro ActiveX control

[A Simple Marking Application](#) – step by step development of an introductory marking application in Visual Basic

[Manipulating The Properties Of The Text Object](#) – adds user text entry to the VB application

[Loading a Mark File into the VB Application](#) – adds mark file loading to the VB application

[Loading Multiple Mark Files into the VB Application](#) – loads one of several files from a text box control

[Using the DrawingIndex Property to Manage Multiple Mark Files](#) – adds the ability to load files once, then index to the desired file

[Handling Digital I/O within the VB Application Code](#) – adds Input/Output automation to the VB application

[Detecting Error Conditions in the MarkDrawing Method](#) – adds error detection to the VB application

[Creating Custom Polyline Objects](#) – uses the AddPolyline method to draw an ellipse object

## Definitions

The following terms are used throughout this Application Note and are meaningful within Visual Basic as well as in WinMark. 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 form in Visual Basic. 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 Visual Basic is supplied with a set of built-in controls like the command button, timer, etc., additional controls are available as stand-alone files with the file extension ".OCX". These controls use *ActiveX* technology to allow their use in a Visual Basic project so that they look and feel like the built-in controls.

### WMP

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

## Getting Started

This section will explain how to add the WinMark ActiveX control to the Visual Basic Toolbox.

1. Install WinMark on the computer that you will use to design the Visual Basic (VB) application.
2. Add the WinMark ActiveX control to the VB toolbox:
  - a. Open Visual Basic and start a new project.
  - b. Click on the 'Projects' menu.
  - c. Select 'Components'.
  - d. Scroll through the list of controls and find 'Synrad WinMark Pro ActiveX'.
  - e. If the WinMark ActiveX control is not listed, click on the 'browse' button, and locate the file SYNMHATX.OCX in the WinMark folder.
  - f. Click the checkbox next to the 'Synrad WinMark Pro ActiveX' control to select it.
  - g. Click on the 'Apply' button.
3. The WinMark icon should now be displayed in the Toolbox window. If you can't see the Toolbox, try clicking on the Toolbox icon on the VB toolbar.

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

## A Simple Marking Application

This section shows you how to create a basic application that will load a text object into the WinMark ActiveX control, then laser the object (if you have a marking system connected).

1. Open a new project and verify that the WinMark ActiveX control is loaded in the VB Toolbox.
2. Place the WinMark ActiveX control on the Visual Basic form by clicking on the WinMark ActiveX tool, then 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 VB routine.
3. Select the WinMark ActiveX control and click on the Properties button on the VB toolbar. Set the control's properties as follows:

Name: mh

4. Place a command button on the form. Set the command button's properties as follows:

Name: cmdMark

Caption: Mark

5. Double-click on a blank portion of the form to bring up the code window.
6. In the 'Private Sub Form\_Load()' event code section, enter the following:

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

7. Click on the form to bring it to the forefront, then double-click on the command button.

8. In the 'Private Sub cmdMark\_Click()' event code section, enter the following:

```
mh.MarkDrawing
```

9. Press the Visual Basic 'Start' button, and you will still see the text object "My Text Object" displayed on the drawing canvas in the ActiveX preview window.

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

That's your first VB marking application! Save the form and project for use in the subsequent sections of this Application Note.

## Manipulating the Properties of the Text Object

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

In the course of completing this section, you will expand the Basic Application developed previously to allow manipulation of the Text1 object's caption and size properties.

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 values of any of the objects in a mark file by using the appropriate 'get property' method:

- GetBoolProp* gets 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* gets any property that has an integer value (such as the Drawing's MarkCount property)
- GetFloatProp* gets any property that has a floating-point value (such as the Drawing's FieldWidth property)
- GetStringProp\** gets any property that has a string value (such as the TextCaption property of a text object)

\*The SetStringProp and GetStringProp methods have a nice built-in feature – they will 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, the 'Text1' object created earlier may be set to not mark by executing the following code:

```
mh.SetBoolProp "Text1", "MarkObject", 0
```

The SetStringProp could also be used for setting the Boolean property:

```
mh.SetStringProp "Text1", "MarkObject", "0"
```

Note that the property name strings recognized by the ActiveX control do not contain spaces, as opposed to the human-readable property names that are listed in the WinMark interface. To obtain a list of the 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 method to write the drawing and object property names and values to a text (.TXT) file from your VB application.

That being said, let's get going on the project modifications.

1. Open the VB project saved in the previous section.
2. Add a Text Box to the form. 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. Press the Visual Basic 'Start' button, and you will still see the modified Basic Marking form. 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. Press the VB 'End' button, and bring up the code design window.
6. Add a command button to the form. Set its properties as follows:  
Name: cmdBigger  
Caption: Text Larger
7. Add a command button to the form. Set its properties as follows:  
Name: cmdSmaller  
Caption: Text Smaller
8. Resize the buttons as required so that the operator can recognize the captions.
9. 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
```
10. 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
```
11. Save the project, then press the VB 'Start' button.

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 the text height also increases the text character 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 VB Application

Allowing the operator to enter and resize text is interesting and makes for a good introduction to the joys of designing a VB marking application, but the end result is not very practical in the industrial sense. It would be a painful process indeed to create a mark with twenty five objects of various types, setting the location, power, velocity and other parameters all in your VB code.

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

This approach becomes even more important if the VB 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.

In the course of completing this section, you will learn how to use the LoadDrawing method to load a WinMark file into your ActiveX application.

1. Open WinMark and create a mark file. For the sake of this exercise, add a 1D barcode, a 2D barcode, and a Text object to the file. Verify that the objects are named "Barcode1", "2Dbarcode1", and "Text1". Save the file as "ActiveX Test.MKH" in the folder that contains the VB tutorial project files.
2. Open the project saved in the last section. Modify the 'Private Sub Form\_Load()' event code section, commenting out the mh.AddText line and adding the mh.LoadDrawing line:

```
'mh.AddText "Text1", 0, 0, "My Text Object"  
mh.LoadDrawing App.Path & "\\ActiveX Test.MKH"  
mh.Redraw
```

3. Press the Visual Basic 'Start' button, and you will still see the file created in WinMark shown in the ActiveX preview window. 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. Press the VB 'End' button and 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.

Now you can load a mark file into your ActiveX application and modify one or more object properties.

## Loading Multiple Mark Files into the VB Application

Creating a mark file in WinMark then loading it into your ActiveX application has its benefits:

- you can use WinMark's drawing tools to precisely size and locate your mark objects as needed
- you can avoid LOTS of property setting code by setting the basic object properties in WinMark

But what if you need to load more than one file?

This section will explore the loading of multiple mark files into your ActiveX 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 Test3.mkh'.

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

Name:     lstFile

3. Modify the 'Private Sub Form\_Load()' event code section, commenting out the mh.LoadDrawing line and adding the lstFile configuration lines:

```
'mh.AddText "Text1", 0, 0, "My Text"  
'mh.LoadDrawing App.Path & "\ActiveX Test.MKH"  
lstFile.AddItem "ActiveX Test"  
lstFile.AddItem "ActiveX Test2"  
lstFile.AddItem "ActiveX Test3"  
lstFile.ListIndex = 0  
mh.Redraw
```

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 App.Path & "\" & lstFile & ".mkh"  
mh.Redraw
```

5. Run the application and notice 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 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 software 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 set the DrawingIndex, load a file, change the DrawingIndex, load a different file, etc. Once all the files are loaded, you switch over to the desired file by calling out its DrawingIndex value. This places no burden on the GDI resources and can speed the process up considerably when using large intricate files. Continue on to understand how this process works.

## Using the DrawingIndex Property to Manage Multiple Mark Files

If you need to load more than one mark file into your ActiveX application, you need to become familiar with the DrawingIndex property.

The DrawingIndex property is roughly analogous to the numbers on a set of post office boxes. You can assign a DrawingIndex value of 0 and load a mark file that is assigned to that index value. You can then assign a DrawingIndex value of 1 and load a different mark file to that index, and so on. When you want to mark one of the files that have already been loaded, you just call up the DrawingIndex value for the file, then execute the MarkDrawing method.

1. To modify your application to use this preferred method, quit the application (if necessary) and open the form's code page.
2. Modify the 'Private Sub Form\_Load()' event code section by adding a Dimension statement and the For...Next loop:

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

3. Note the use of the With...End With tool to ease the typing burden on the lstFile configuration code and to make the setup code easier to read (compare this to the code entered in the previous section).
4. Modify the 'Private Sub lstFile\_Click()' event code section, commenting out the LoadDrawing command and adding the DrawingIndex reference:

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

5. 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 long enough, you may have noticed that that ESC key must be pressed several times before it is recognized. This is due to the fact that, while the vector and laser control data is being generated during lasing, even the keyboard activity is ignored by the priority asserted by the WinMark marking engine. In this same way, when the ActiveX MarkDrawing method is called, the marking engine takes control of the 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 I/O automation, and the required input condition is never met, control will never return to the VB application and the system will appear to lock up.

For this reason, it is best to move all Input/Output automation from the mark file to the VB code. By moving the automation to your VB app, your code can be written to accommodate stuck or missing input conditions.

That said, let's move on. While you could use a VB timer to periodically test for a specific input condition, you will get a quicker response to an input change using a Do loop. The following code will loop until Input bit 0 goes active:

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

### Notes:

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

This code will loop as long as the input status requirement evaluates as True. The code execution will move on past the loop as soon as the input test evaluates False. If you want to wait until IN0 is cleared, you just have to remove the NOT condition from the statement.

You can use more than one input bit as the loop condition. For instance, to wait until Input bit 0 is inactive and bit 3 is active:

```
Do While ((mh.GetDigitalBit(0)) Or (Not (mh.GetDigitalBit(3))))
    DoEvents
Loop
```

Note the use of the logical OR, since the loop condition must be False to proceed through the code.

Setting or clearing an output bit is fairly simple:

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



The following code is executed when the cmdMark command button is pressed and will generate one mark loop.

```
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
```



## 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. Keep in mind that the error condition is in reference to the method execution, rather than the correct behavior of the marking system.

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 higher, a GetMarkErrorCode method added to the ActiveX control provides more detail for mark error conditions. This method may be called 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
```

Try running this code, commanding a mark while 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
```