THERMOMICROSCOPES

SPMTOOLS ANALYSIS
TECHNICAL REFERENCE GUIDE

ThermoMicroscopes
1171 Borregas Avenue
Sunnyvale, CA 94089
Tel: (408) 747-1600
Fax: (408) 747-1601

February 2001

85-10332 Rev. A

Binary License Agreement

You, the Licensee, assume responsibility for the selection of the program to achieve your intended results, and for the installation, use, and results obtained from the program.

IF YOU USE, COPY, MODIFY, OR TRANSFER THE PROGRAM, OR ANY COPY, MODIFICATION, OR MERGED PORTION, WHOLE OR PART EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

LICENSE

You may:

- Use the program on a single machine and copy the program into any machine-readable or printed form for backup or support of your use of the program on the single machine.
- Modify the program and/or merge it into another program for your use on the single machine. Any portion of the program merged into another program will continue to be subject to the terms of this Agreement. You must reproduce and include the copyright notice on any copy, modification, or portion merged into another program.
- Transfer the program and license to another party within your organization if the party agrees to accept the terms and conditions of this Agreement. If you transfer the program, you must at the same time either transfer all copies, whether in machine readable form or printed form, to the same party or destroy any copies not transferred; this includes all modifications and portions of the program merged into other programs. You may not transfer the program to a party outside of your organization without the express written permission of ThermoMicroscopes.

TERMS

The license is effective on the date you take delivery of the software as purchased from ThermoMicroscopes, and remains in effect until terminated as indicated above or until you terminate it. If the license is terminated for any reason, you agree to destroy or return the program together with all copies, modifications, and merged portions in any form.

COPYRIGHT NOTICE—COVERS ALL ATTACHED DOCUMENTATION

ThermoMicroscopes Corporation retains all ownership rights to this documentation and all revisions of the SPMLab computer program and other related software options.

Reproduction of any portion of this publication or any image depicted in this publication (except for archival purposes or for the specific use of the program in conjunction with ThermoMicroscopes equipment) without prior written authorization of ThermoMicroscopes is prohibited by law and constitutes a punishable violation of the law.

THERMOMICROSCOPES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THERMOMICROSCOPES BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE, LOSS OF DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND EVEN IN THE EVENT OF SUCH DAMAGES ARISING FROM ANY DEFECT OR ERROR IN THIS PUBLICATION OR IN THE SPMLab SOFTWARE.

SPMTools Data Acquisition, SPMTools Analysis, SPMLab, and ThermoMicroscopes are trademarks of ThermoMicroscopes Corporation.

THERMOMICROSCOPES

You can find ThermoMicroscopes on the World Wide Web at http://www.thermomicro.com.

CORPORATE HEADQUARTERS

ThermoMicroscopes
1171 Borregas Avenue
Sunnyvale, CA 94089
USA

Tel: 408.747.1600
Fax: 408.747.1601
E-mail: info@thermomicro.com

THERMOMICROSCOPES EUROPE

ThermoMicroscopes Europe
48, rte des Acacias
CH-1227 Acacias (Geneva)
Switzerland

Tel: (+41) 22 300 44 11
Fax: (+41) 22 300 44 15
E-Mail: marc@thermomicro.com

THERMOMICROSCOPES UK

ThermoMicroscopes UK
4A Minton Place
Victoria Road
Bicester OX6 7QB, United Kingdom

Tel: (+44) 01869 252544
Fax: (+44) 01869 327744
E-mail: drewmurray@dial.pipex.com

THERMOMICROSCOPES PRODUCT WARRANTY COVERAGE

ThermoMicroscopes warrants that products manufactured by ThermoMicroscopes will be free of defects in materials and workmanship for one year from the date of shipment. The product warranty provides for all parts (excluding consumables, maintenance items, and PCs), labor, and software upgrades.

Instruments, parts, and accessories not manufactured by ThermoMicroscopes may be warranted by ThermoMicroscopes for the specific items and periods expressed in writing on published price lists or quotes. However, all such warranties extended by ThermoMicroscopes are limited in accordance with all the terms, conditions, and other provisions stated in this warranty. ThermoMicroscopes makes no warranty whatsoever concerning products or accessories not of its manufacture except as noted above.

Customers outside the United States and Canada should contact their local ThermoMicroscopes representative for warranty information appropriate to their locale.

CUSTOMER RESPONSIBILITIES

1. Use ThermoMicroscopes replacement parts.
2. Use ThermoMicroscopes or ThermoMicroscopes-approved consumables, such as lamps, cantilevers, filters, etc.
3. Provide adequate and safe working space around the products for servicing by ThermoMicroscopes personnel.

REPAIRS AND REPLACEMENTS

ThermoMicroscopes will, at its option, either repair or replace defective instruments or parts. Repair or replacement of products or parts under warranty does not extend the original warranty period. With the exception of consumable and maintenance items, replacement parts or products used on instruments out of warranty are themselves warranted to be free of defects in materials and workmanship for 90 days. Any product, part, or assembly returned to ThermoMicroscopes for examination or repair must have prior approval from ThermoMicroscopes and be identified by a Return Materials Authorization (RMA) number obtained before returning the product. The product, part, or assembly must be sent freight prepaid to the factory by the Customer. Return transportation to the customer will be at ThermoMicroscopes' expense if the product, part, or assembly is defective and under warranty.

WARRANTY LIMITATIONS

This warranty does not cover: *a*) Parts and accessories which are expendable or consumable in the normal operation of the products. *b*) Any loss, damage, and/or product malfunction resulting from shipping or storage, accident, abuse, alteration, misuse, or use of user-supplied software, hardware, replacement parts, or consumables other than those specified by ThermoMicroscopes. *c*) Products which are not properly installed. *d*) Products which are not operated within the specified environmental conditions. *e*) Products which have been modified or altered without written authorization from ThermoMicroscopes. *f*) Products which have had the serial number altered or removed. *g*) Improper or inadequate care, maintenance, adjustment, or calibration by the user.

SERVICE

To contact the service department directly with service-related issues or questions, you can reach service personnel Monday through Friday, 8:00 am-5:00 pm, Pacific Standard Time (voice mail is available 24 hours).

Within the USA and Canada: 800-727-5782

Outside the USA and Canada: 408-744-3001

Fax: 408-747-1601

e-mail: support@thermomicro.com

When contacting the service department, it is helpful if you have the model and purchase order numbers available.

Table of Contents

SPMTools Analysis Installation vii

Chapter 1 — SPMTools Analysis Introduction 1

Technical Reference Guide 1

SPMTools Analysis General Overview 1

SPMTools Analysis Product Architecture 3

Chapter 2 — SPMTools Analysis Architecture 5

Microsoft Visual Basic 5

ThermoMicroscopes Supplied Software 5

Chapter 3 — Tutorial 7

Project Creation 7

Designing the User Interface (UI) 8

Writing the BASIC Programs 8

Running & Debugging the Application 8

“Shrink Wrapping” the Application 8

Tutorial Examples 8

Creating a Simple Leveling Program 10

Example 1: Basic Processing and Analysis 12

Example 2: Manual Leveling 13

Example 3: Line Profile Display 16

Example 4: Drawing Objects & Controlling Contrast 20

Example 5: Printing Images 22

Chapter 4 — Software Component References 23

TOPOIMG.OCX Reference 23

Member Variables 23

Methods 25

OCX events....26
TOPOGRAPH.OCX Reference....27
Member variables....27
Methods....27
OCX events....30
TOOL32.DLL Reference....31
TOPOPRO.DLL Reference....33

SPMTOOLS ANALYSIS INSTALLATION

1. Load Microsoft Visual Basic 6.0 onto the computer following the Microsoft installation instructions.

2. If you are installing SPMTools Analysis on a computer that controls an Explorer instrument, make sure that SPMTools Data Acquisition is already installed.

3. Insert Diskette 1 into the 3.5-inch floppy drive (drive a:) of your computer.

4. From the Windows **START** button, select **RUN**, and type: **`a:setup.exe`**. Follow the instructions in the wizard to complete the installation.

CHAPTER 1 — SPMTOOLS ANALYSIS INTRODUCTION

TECHNICAL REFERENCE GUIDE

The SPMTools Analysis Technical Reference Guide describes the steps involved in developing image analysis applications using the Microsoft Visual Basic 6.0 software development environment and a set of ThermoMicroscopes standard software components.

It is assumed that the user of SPMTools Analysis is familiar with Microsoft Visual Basic 6.0 and has some experience programming in the BASIC language. Refer to the Microsoft Visual Basic 6.0 Programmer's Guide for questions not addressed in this manual.

SPMTOOLS ANALYSIS GENERAL OVERVIEW

SPMTools Analysis provides a great deal of flexibility for developing image processing tools based on the image processing module of ThermoMicroscopes SPMLab. SPMTools Analysis is used to write program code in Microsoft Visual Basic 6.0 to access image processing and visualization functions.

The SPMTools Analysis product was developed based on the following design criteria:

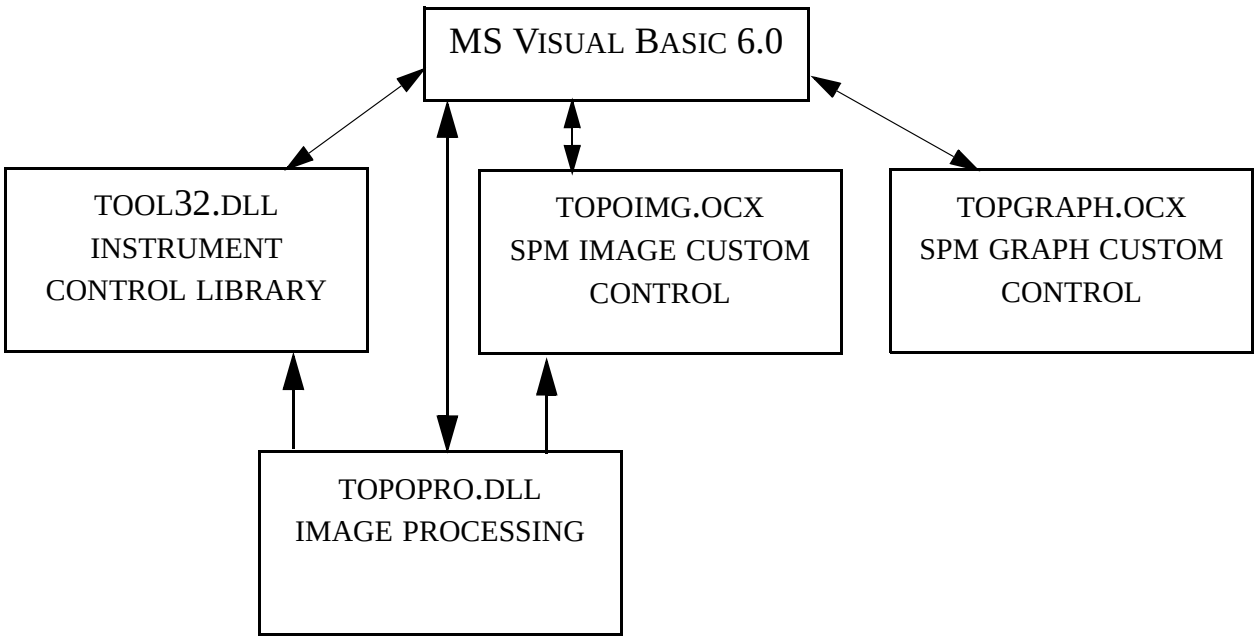
- It uses a standard software development environment.
- It provides open software architecture to allow design of custom image pro-

cessing algorithms.

- It provides a high level language with a graphical user interface to allow the use of standard image processing algorithms and the display of processed data.
- It uses SPMLab software components.

The five examples provided with the software package serve only to illustrate how to use SPMTools Analysis and do not have the full functionality of SPMLab. Through the examples, the user is allowed to select only the most basic parameters.

SPMTOOLS ANALYSIS PRODUCT ARCHITECTURE



CHAPTER 2 — SPMTOOLS ANALYSIS ARCHITECTURE

MICROSOFT VISUAL BASIC

The main development environment for SPMTools Analysis is Microsoft Visual Basic 6.0. The professional edition of Visual Basic 6.0 includes all the programming and user interface development tools that the SPMTools Analysis user needs.

As the SPMTools Analysis product evolves, more utility will be added to the product in the forms of Visual Basic sample source code, application development DLLs, and Windows “mini applications.”

THERMOMICROSCOPES SUPPLIED SOFTWARE

TOOL32.DLL TOOL32.DLL is a 32 bit DLL that provides higher level acquisition, display, and file handling functions.

TOPOIMG.OCX TOPOIMG.OCX is an OLE control that allows control and visualization of SPM images.

TOPOPRO.DLL TOPOPRO.DLL is a 32-bit DLL that contains standard image proecessing dialog boxes and higher level image processing algorithms.

TOPOGRAPH.OCX TOPOGRAPH.OCX is an OLE control that allows control and visualization of 2D SPM data.

CHAPTER 3 — TUTORIAL

Chapter 3 describes the functionality of SPMTools Analysis by briefly mentioning the steps necessary to construct a running Visual Basic application and by giving examples to demonstrate how the constructed applications can be utilized by the user.

As previously mentioned, it is assumed that the user of SPMTools Analysis has a working knowledge of Visual Basic 6.0. If necessary, refer to the Microsoft Visual Basic Programmer's Guide before beginning this tutorial for questions concerning the process of creating a Visual Basic Application.

PROJECT CREATION

The first step in developing a Microsoft Windows application using Visual Basic is to create a new project. Once the project is created, the system keeps track of all incremental changes made to the application source files and ensures the integrity of the software application. Any number of projects can be created and developed in parallel since all of the housekeeping tasks are performed by the system. Note that the Visual Basic program must start from the SPMTools Analysis directory.

To create a new SPMTools Analysis project, the user simply creates a new project and adds the TOPOWN32.BAS, TOPOPRO.BAS, and TOPOIMG.OCX (see Chapter 4) files to the project. The user can then begin designing, writing, and running the application.

DESIGNING THE USER INTERFACE (UI)

After creating the project, the user designs the User Interface (UI) of the application by inserting available UI components and controls (such as drop-down menus, command and option buttons, and list and text boxes) into the appropriate Visual Basic form(s). The user can easily rearrange the UI at any point during development by reentering the design mode.

WRITING THE BASIC PROGRAMS

A BASIC subroutine template is created by the system for each UI control inserted into the Visual Basic form(s). The user writes BASIC code into the subroutine templates to direct the interactions of the application and the underlying logic and flow of the program. DLLs supplied with this package can be called from the BASIC subroutines, which provides access to all of the tools available within SPMTools Analysis.

RUNNING & DEBUGGING THE APPLICATION

The user can run the application at any time by clicking on the Run button ▶ on the Visual Basic toolbar or by selecting **RUN→START**. If there are any errors in the code, the system tries to guide the user to the source of the error.

In most cases, the user engages in a debugging session by setting breakpoints in the code and monitoring the values of different variables as the application runs in a simulated fashion. The user can make any necessary changes directly to the code from within the same editing window where debugging is performed.

"SHRINK WRAPPING" THE APPLICATION

When the user decides that the application is ready to stand alone, an executable Microsoft Windows program is generated in the form of a .EXE file. The "shrink wrapped" application runs without any dependence on the Visual Basic environment, which drastically reduces the system overhead.

TUTORIAL EXAMPLES

The SPMTools Analysis tutorial examples are found in the Visual Basic project TOOLPRO_EX.VBP, which can be found in the SPMTools Analysis program directory. Once the project is opened and loaded, the examples are given as the files EXAMPLE1.FRM, EXAMPLE2.FRM, EXAMPLE3.FRM, etc. After double clicking on and opening the example of interest, the form will be in the design mode, and the user can begin programming.

To perform a data analysis routine using the SPMTools Analysis examples, the user clicks the Run button ▶ on the toolbar, and a window containing five command buttons, one for each example, will open, as shown in Figure 3.1. Click on the example command buttons to perform data analysis using the user interface of each example.

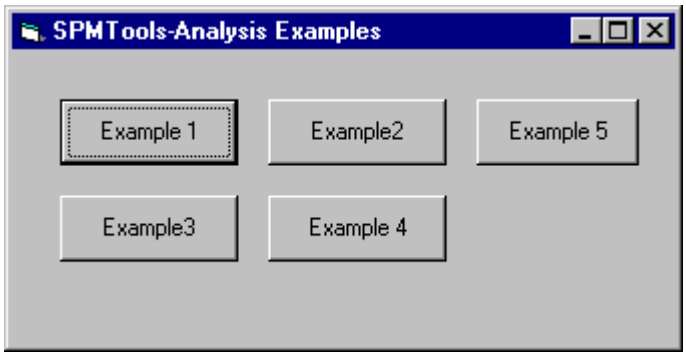


FIGURE 3.1

Throughout the tutorial, BASIC code is used to explain events that occur in SPMTools Analysis. Since the code in this manual is used for discussion purposes only, the actual code found in the SPMTools Analysis application may contain more lines than are shown.

Below are the topics discussed in the tutorial. The example number in which the topic is covered is given in parentheses (though some topics are not specifically covered in the examples).

- Standard SPMLab 5.01 dialogs (1)
- Image levelling (2)
- Image buffer (1)
- Image processing (2)
- Scale operation (4)
- Graph display (3)

CREATING A SIMPLE LEVELING PROGRAM

This first tutorial takes the user through the complete process of creating a stand-alone application and does not use any of the examples. The simple leveling program created in this tutorial will have an image display and four buttons: Load Image, Level, Save and Undo.

BEGIN THE PROJECT

1. Make sure the Visual Basic program starts from the SPMTools Analysis directory.
2. Launch the Visual Basic application.
3. To open a new project, select **FILE→NEW PROJECT** if the New Project dialog is not already open. Click on the New tab, select Standard.EXE, and click **OPEN**. A new project window opens.
4. Load the required OCX files by selecting **PROJECT→COMPONENTS**. Click on the Control tab, locate "TopoGraph OLE Control module" and "Topoimg OLE Custom Control module" in the list, and check the box beside each. If using SPMTools Analysis for the first time, these two files will not appear in the list. To add them, click the **BROWSE** button and navigate to the folder containing the SPMTools Analysis program files.
5. Select **PROJECT→ADD MODULE**, click the **EXISTING** tab, and select Topopro.bas (located in the folder containing the SPMTools Analysis program files).
6. Select **PROJECT→ADD MODULE**, click the **EXISTING** tab, and select Topown32.bas.
7. Confirm that the required modules are loaded by selecting **VIEW→PROJECT EXPLORER** and viewing the contents of the Modules folder.

ADD FUNCTIONS

1. Click the Command button in the VB Toolbox, and size and place the first button in the form.
2. Right-click on the new button, select Properties, and enter the caption: "Level."
3. Click on the Image button to add an image display to the form.
4. Click the Command button to add another button to the form, and call it "Load Image."

   The form should now look like the one shown in Figure 3.2.

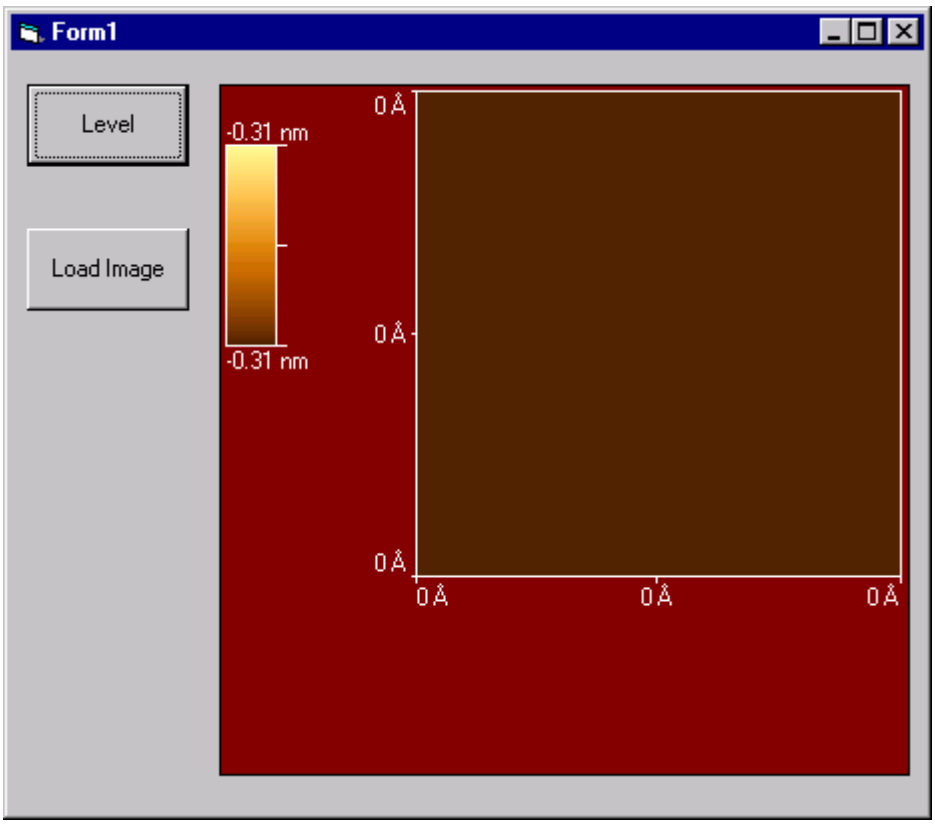


FIGURE 3.2

5. Double-click on the **LOAD IMAGE** button to access the code in the text editor.
6. Type: **topoimg1.fileload="exampleimg.zfr"** (to reference the sample image provided).
7. Double-click on the **LEVEL** button, and type:
   call hLevelingDlg(hWnd,topoimg1.hWnd)

SAVE AND TEST PROGRAM

1. Select **FILE→SAVE FORM**, and save the form as MyFirstForm.
2. Select **FILE→SAVE PROJECT**, and save the project as MyFirstProject.
3. Click the Run button ▶ or select **RUN→START** to run the newly created program.
4. Click the **LOAD IMAGE** button to load the supplied image file.
5. Click the **LEVEL** button to open the image leveling dialog box.
6. Choose a leveling function and apply it.

The above steps outline the basic procedure for applying image processing. In Step 7, under "Add a Function," any of the image processing functions can be called.

ADD AN "UNDO" BUTTON

1. Use the Command button to add a new button called "Undo."
2. Double-click on the **UNDO** button, and type:
   bSelectPreviousDataBuffer(topoimg1.hWnd)
3. Save the form and project, and run the program again. After performing leveling, the **UNDO** button will restore the previous image in the buffer to the display.

EXAMPLE 1: BASIC PROCESSING AND ANALYSIS

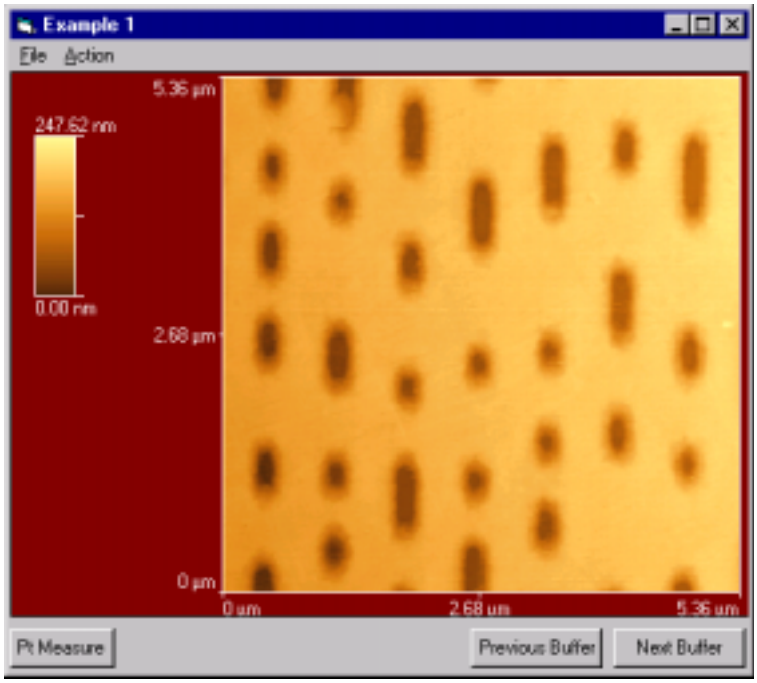


FIGURE 3.3

Figure 3.3 shows the Visual Basic form for Example 1 (EXAMPLE1.FRM) with an image loaded. The form is used to apply the main imaging processing algorithms in SPMLab 5.01 and is similar to the leveling program created in the previous section.

EXAMPLE 2: MANUAL LEVELING

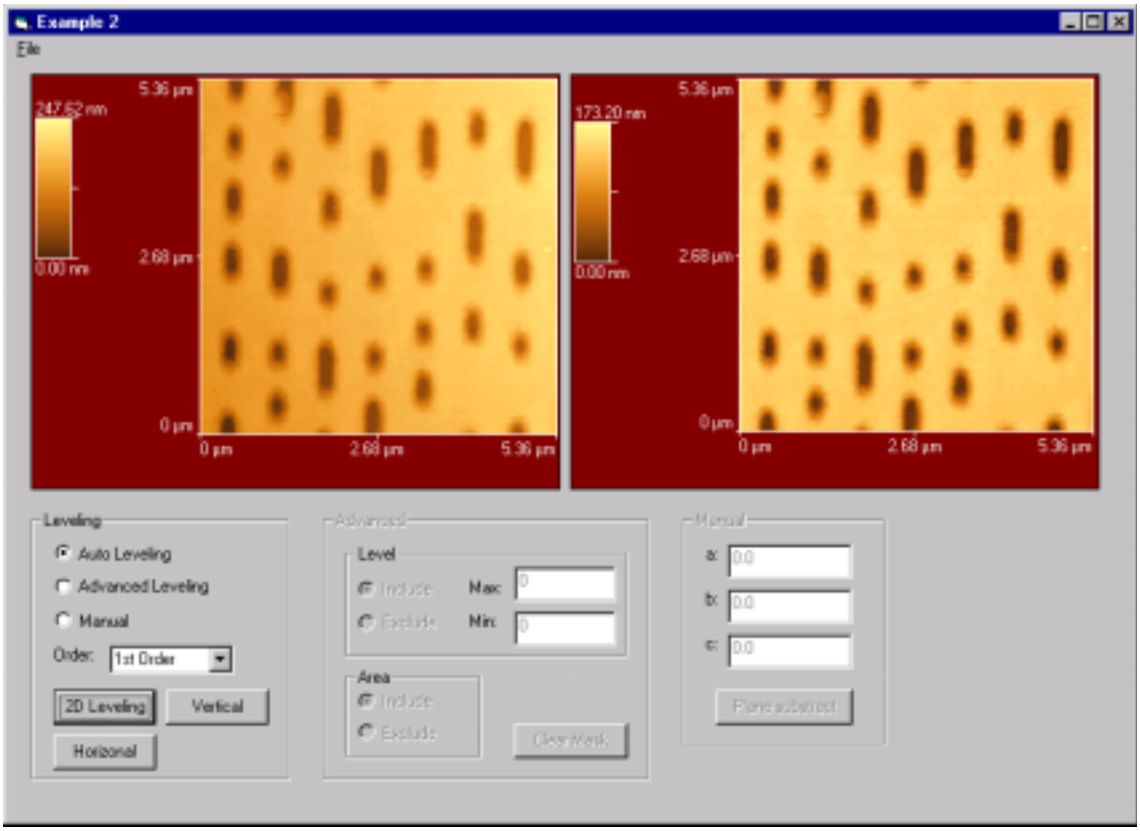


FIGURE 3.4

Figure 3.4 shows the Visual Basic form for Example 2 (EXAMPLE2.FRM) with an image loaded (in the left display) and the result of applying a leveling routine (in the right display). The form is used to apply standard image leveling as well as custom leveling algorithms.

Below is the code for applying basic leveling (using the **2D LEVELING** button):

```
Private Sub Cmd2DLevel_Click()
Dim bOK As Boolean
Dim hDest As Long
' If hDest = 0, Processed image overwrite the orginal image
    If bSameOCX Then
        hDest = 0
    Else
        hDest = Topoimg2.hWnd
    End If
    bOK = bImageLeveling(hDest, Topoimg1.hWnd, LEVELING_2D,
iOrder, Not (bAuto))
```

```
End Sub
```

`iOrder` is the order of leveling; it is defined by the **ORDER** combo box. `bAuto` is defined by the radio buttons (**AUTO LEVELING**, **ADVANCED LEVELING** or **MANUAL**).
`bAuto`= True = simple 2D leveling
`bAuto`= False = masked leveling

The mouse mode is determined by the user input: Auto, Advanced or Manual leveling (page 24). The following code shows how the cursor displays differently and how the function is called to enable area selection for masking in Advanced leveling.

```
Private Sub OptAdvance_Click()
    bAuto = False
    Call EnableAdvLevel(True)’ enable advanced leveling
    Call EnableManualLevel(False)’ disable manual leveling
    Call EnableLevelingCommand(False)’ disable leveling but-
ton; no leveling until a mask is selected
    Topoimg1.MouseMode = 3 ’ enable area selection
End Sub

Private Sub OptAuto_Click()
    bAuto = True
    Call EnableAdvLevel(False) ’ disable advanced leveling
    Call EnableManualLevel(False) ’ disable manual leveling
    Call EnableLevelingCommand(True) ’ enable leveling but-
tons
    Topoimg1.MouseMode = 0 ’ disable area selection
End Sub

Private Sub OptManual_Click()
    Call EnableAdvLevel(False)
    Call EnableManualLevel(True)
    Topoimg1.MouseMode = 0 ’ disable area selection
End Sub
```

The following code shows the response to the mouse button when the mouse cursor is inside the left-hand image display and mouse mode <>0.

```
Private Sub Topoimg1_GetPointPos(ByVal iCursor As Long,
ByVal PosLeft As Long, ByVal PosTop As Long, ByVal PosRight
As Long, ByVal PosBottom As Long)
Dim res As Integer
   If (iCursor = IMG_RIGHT_MOUSE_BUTTON) Then
        res = iGetHwndImageResolution(Topoimg1.hWnd)
       Call vSetMaskBlock(PosLeft, PosTop, PosRight, PosBot-
tom, res, bIncludeArea)
        Call vShowMaskOverLay(Topoimg1.hWnd)
        Call EnableLevelingCommand(True)
    End If
```

```
End Sub
```

The following code for the Plane Subtraction button demonstrates how point-by-point data processing can be done. The first-order leveling shown below serves as the template for all manual leveling routines. For example, the code could be changed to do 2nd- and 3rd-order leveling, or to skip lines.

```
Private Sub CmdFit_Click()
Dim i As Integer, j As Integer, k As Long
Dim lData(1024) As Long, iRes As Long
Dim a As Double, b As Double, c As Double
Dim iMin As Long, iMax As Long
'maually substrate a plane Z = ax + by + c
    a = Val(CoeffA.Text)
    b = Val(CoeffB.Text)
    c = Val(CoeffC.Text)
    ' Copy from topoimg1 to topoimg2 so that all the header
info get transfered
    Call vCopyImage(Topoimg2.hWnd, Topoimg1.hWnd)
    iRes = iGetHwndImageResolution(Topoimg1.hWnd)
    For i = 0 To iRes - 1
        k = bGetImageData(Topoimg1.hWnd, i, 0, 1, iRes,
lData(0))
        For j = 0 To iRes - 1
            lData(j) = lData(j) - a * j - b * i
        Next j
        k = bSetOcxImageData(Topoimg2.hWnd, i, 0, 1, iRes,
lData(0), False)
    Next i
    Call vGetDACMinMax(Topoimg2.hWnd, iMin, iMax)
    Call vNewDACMinMaxImage(Topoimg2.hWnd, 0, iMin, iMax)
    Call vRedisplayImage(Topoimg2.hWnd)
End Sub
```

EXAMPLE 3: LINE PROFILE DISPLAY

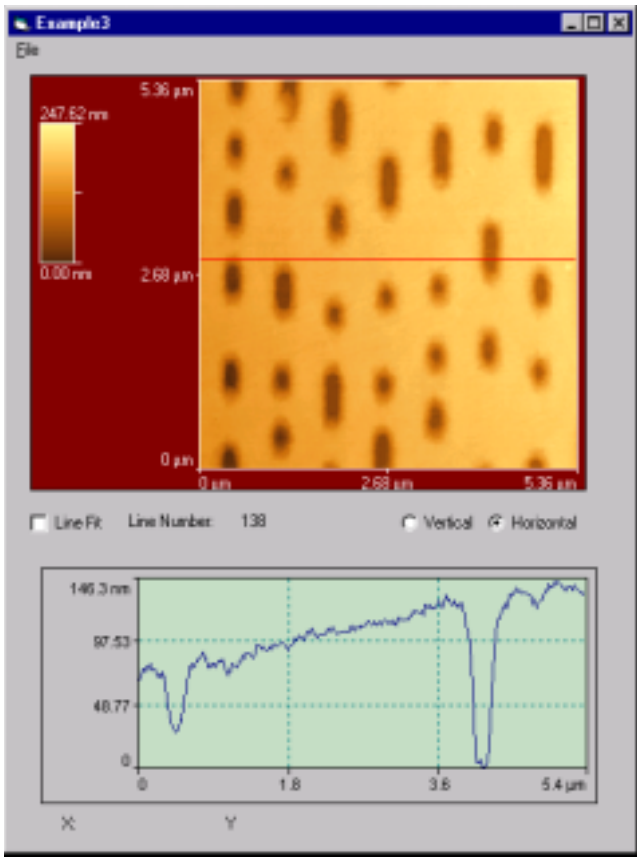


FIGURE 3.5

Figure 3.5 shows the Visual Basic form for Example 3 (EXAMPLE3.FRM) with an image loaded and a line profile displayed. The form is used to do simple line profile display.

The following code shows how a line is drawn on the image and how the line profile is then drawn on the graph.

```
Private Sub Topoimg1_GetPointPos(ByVal iCursor As Long,
ByVal PosLeft As Long, ByVal PosTop As Long, ByVal PosRight
As Long, ByVal PosBottom As Long)
Dim iRes As Long, k As Long, i As Integer
Dim iYmin As Long
Dim fData(1024) As Double
    iRes = iGetHwndImageResolution(Topoimg1.hWnd)
    Call Topoimg1.ClearAllOverlay
    If (OptVertical.Value = True) Then
        LineNum.Caption = Str(PosLeft)
```

```
        k = iGetImageLineData(Topoimg1.hWnd, PosLeft,
fData(0), True, ChkLineFit.Value)
        iYmin = Int(fData(0))
        For i = 0 To iRes - 1
            lData(i) = fData(i)
            If (lData(i) < iYmin) Then
                iYmin = lData(i)
            End If
        Next i
        'or the following line if Line Fit is not required
        ' k = bGetImageData(Topoimg1.hWnd, 0, PosLeft, iRes,
1, lData(0))
       Call Topoimg1.AddOverlay(OVERLAY_LINE, 0, RGB(255, 0,
0), PosLeft, 0, PosLeft, iRes - 1)
    Else
        LineNum.Caption = Str(PosTop)
        k = iGetImageLineData(Topoimg1.hWnd, PosTop,
fData(0), False, ChkLineFit.Value)
        iYmin = Int(fData(0))
        For i = 0 To iRes - 1
            lData(i) = fData(i)
            If (lData(i) < iYmin) Then
                iYmin = lData(i)
            End If
        Next i
        ' Or the following line if Line Fit is not required
        '  k = bGetImageData(Topoimg1.hWnd, PosTop, 0, 1,
iRes, lData(0))
       Call Topoimg1.AddOverlay(OVERLAY_LINE, 0, RGB(255, 0,
0), 0, PosTop, iRes - 1, PosBottom)
    End If
    For i = 0 To iRes - 1
      fLineData(i * 2) = i * fxyscale ' x data
      fLineData(i * 2 + 1) = (lData(i) - iYmin) * fDACWorld
+ fDACZero ' y data
    Next i
    bLineDefined = True
    If (bUserDraw) Then
       Call vDrawLine ' user must implement own drawing rou-
tine
    Else
        Call TopoGraph1.SetGraphData(iRes, fLineData(0)) '
draw data using Topograph.ocx
    End If
End Sub
```

The following code shows the response to a mouse click on the line profile.

```
Private Sub TopoGraph1_ClickedOnGraph(ByVal wAction As Long,
ByVal iCursorMode As Long, ByVal wNumPts As Long, ByVal
fXValue As Double, ByVal fYValue As Double)
```

```
Dim iPt As Integer, iRes As Long, iSize As Long
Dim fPt As Double, fFraction As Double
Dim fZValue As Double
    iRes = iGetHwndImageResolution(Topoimg1.hWnd)
    iSize = iRes / 50  ' cross mark, 1/50 of the screen size
    XValue.Caption = Format(fXValue, "#0.00")
    'YValue.Caption = Format(fYValue, "#0.00")
    fPt = fXValue / fxyscale
    iPt = Int(fPt)
    fFraction = fPt - iPt
    If (iPt = iRes - 1) Then
        fZValue = fLineData((iRes - 1) * 2 + 1)
    Else
        ' do interportation
        fZValue = (fLineData((iPt + 1) * 2 + 1) - fLine-
Data(iPt * 2 + 1)) * fFraction + fLineData(iPt * 2 + 1)
    End If
    YValue.Caption = Format(fZValue, "#0.00")
    Label5.Caption = Str(iCursorMode)
    If (wAction = LEFTBTNDOWN) Then
        Call TopoGraph1.SetGraphMarker(fXValue, fZValue, 1,
8, RGB(255, 0, 0), 0) ' add marker on the image
        If (OptVertical.Value = True) Then
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), Val(LineNum.Caption) -
iSize, iPt - iSize, Val(LineNum.Caption) + iSize, iPt +
iSize)
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), Val(LineNum.Caption) +
iSize, iPt - iSize, Val(LineNum.Caption) - iSize, iPt +
iSize)
        Else
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), iPt - iSize, Val(Line-
Num.Caption) - iSize, iPt + iSize, Val(LineNum.Caption) +
iSize)
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), iPt - iSize, Val(Line-
Num.Caption) + iSize, iPt + iSize, Val(LineNum.Caption) -
iSize)
        End If
    End If
    If (wAction = RIGHTBTNDOWN) Then
        Call TopoGraph1.ClearMarker(0)
      Call Topoimg1.ClearAllOverlay
        If (OptVertical.Value = True) Then
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0),    Val(LineNum.Caption), 0,
Val(LineNum.Caption), iRes - 1)
        Else
```

```
            Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), 0, Val(LineNum.Caption),
iRes - 1, Val(LineNum.Caption))
        End If
    End If
End Sub
```

EXAMPLE 4: DRAWING OBJECTS & CONTROLLING CONTRAST

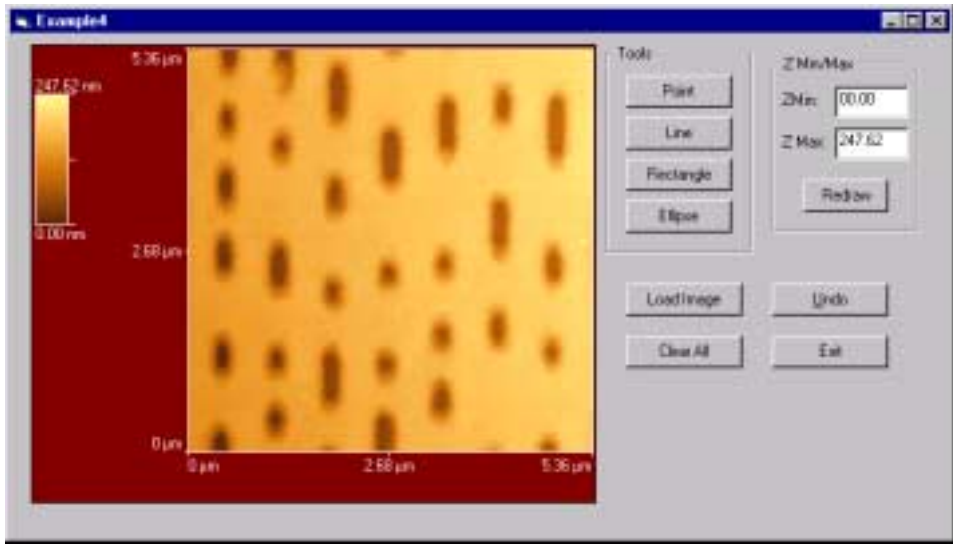


FIGURE 3.6

Figure 3.6 shows the Visual Basic form for Example 4 (EXAMPLE4.FRM) with an image loaded. The form is used to draw shapes on the image display and control the contrast.

The following code shows how a shape is drawn to overlay the image display. A mouse click on the **POINT**, **LINE**, **RECTANGLE** or **ELLIPSE** buttons determines both the mouse mode and the overlay mode. The mouse mode determines the kind of region that is defined when the mouse moves. The overlay mode is used to draw the feature once the region is defined. This can be done in one of two ways: Exclusive OR or Normal. The overlay mode understands lines, rectangles and ellipses, but not a single point, so when the **POINT** function is selected, a point is represented by two overlapping lines ✚. The overlay mode does not distinguish between a square and a rectangle, nor between a circle and an ellipse.

```
Private Sub Topoimg1_GetPointPos(ByVal iCursor As Long,
ByVal PosLeft As Long, ByVal PosTop As Long, ByVal PosRight
As Long, ByVal PosBottom As Long)
    If (iCursor = IMGN_POINT_CLICK) Then
        Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), PosLeft - 5, PosTop, PosLeft
+ 5, PosTop)
        Call Topoimg1.AddOverlay(OVERLAY_LINE,
OVERLAY_NORMAL, RGB(255, 0, 0), PosLeft, PosTop - 5, Pos-
Left, PosTop + 5)
    Else
        If (iCursor = IMGN_AREA_DONE) Then
```

```
            Call Topoimg1.AddOverlay(iOverlayMode,
OVERLAY_NORMAL, RGB(255, 0, 0), PosLeft, PosTop, PosRight,
PosBottom)
        End If
    End If
End Sub
```

The following code shows how an image is redrawn with a different image contrast, based on values input in the **ZMIN/MAX** boxes.

```
Private Sub CmdReDraw_Click()
Dim iNewDACmin As Long, iNewDACmax As Long
    fZMin = Val(NewZMin.Text)
    fZMax = Val(NewZMax.Text)
    If (fZMax > fZMin) Then
        iNewDACmin = (fZMin + fZBase + fDACZero) / fDACScale
' convert from real world unit to DAC value
        iNewDACmax = (fZMax + fZBase + fDACZero) / fDACScale
' convert from real world unit to DAC value
       Call vNewDACMinMaxImage(0, Topoimg1.hWnd, iNewDACmin,
iNewDACmax)
    End If
End Sub
```

EXAMPLE 5: PRINTING IMAGES

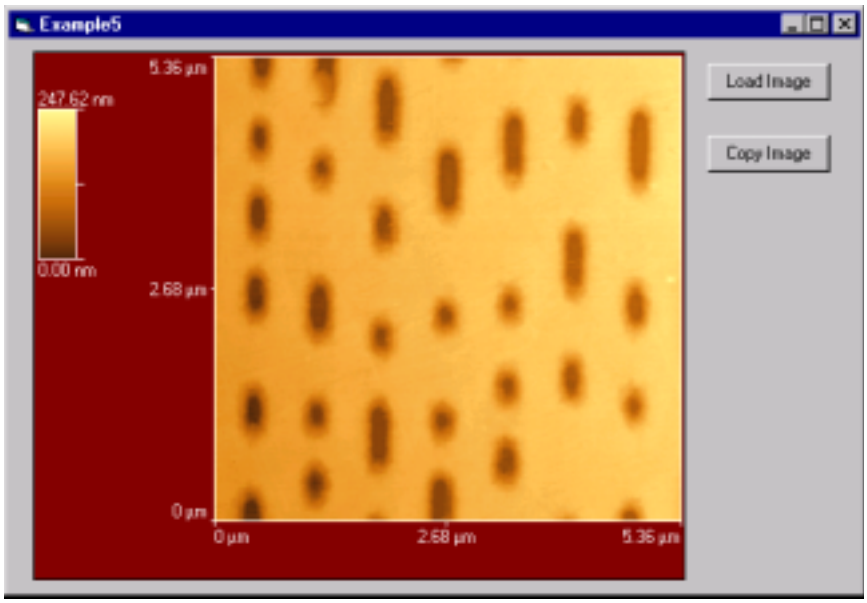


FIGURE 3.7

Figure 3.7 shows the Visual Basic form for Example 5 (EXAMPLE5.FRM) with an image loaded. The form is used to enable the printing of images by using standard Windows copy and paste methods to print from other applications. In this example, the Load Image is the only SPMTools Analysis function.

CHAPTER 4 — SOFTWARE COMPONENT REFERENCES

TOPOIMG.OCX REFERENCE

The OCX control provides an interface for collecting and displaying ThermoMicroscopes SPM images and the convenience to save and restore images from files.

MEMBER VARIABLES

AcquireEnable (BOOL)
The AcquireEnable property determines whether the image is an acquisition image (TRUE) or not (FALSE). This property is for data acquisition only and should be set to FALSE when using SPMTools Analysis.

DataType (long)
The DataType property is the data type of the image.

- Topography
- Internal Sensor
- Z Modulation
- Lateral Force
- External Input 1
- External Input 2
- FastTrack
- Linearized Z
- User defined channel 0-3

DataDirection (long)
The DataDirection property is the direction (Forward or Reverse) of the image scan and is for data acquisition only.

DataMode (long)
The DataMode property is the image acquire mode and is for data acquisition only.

2D

3D-FIS

3D-CITS

3D-MFM

3D-DITS

3D-EFM

Labels (long)
The Labels property defines the XY labels that will be present on the image.

None

X and Y

X only

Y only

FileLoad (string)
The FileLoad property defines the filename from which the program is to be loaded.

FileSave (string)
The FileSave property defines the filename under which the program is to be saved. Once the FileSave property is set, the image data is automatically stored into the file.

MouseMode (long)
When the MouseMode value is <>0, the following shapes can be defined.

MOUSEMODE VALUE	SHAPE
1	POINT
2	LINE
3	RECTANGLE
4	SQUARE
5	HORIZONTAL LINE
6	VERTICAL LINE

MOUSEMODE VALUE	SHAPE
7	ELLIPSE
8	CIRCLE

METHODS

void AddOverlay(long iType, long iDrawMode, OLE_COLOR DrawColor, long Left, long Top, long Right, long Bottom)

Parameters:

iType: Overlay type

OVERLAY_LINE = 0 = Line

OVERLAY_RECT = 1 = Rectangle

OVERLAY_ELLIPSE = 2 = Ellipse

OVERLAY_TEXT = 3 = Text

OVERLAY_TEXT_SCALED = 4 = Scalable text

iDrawMode: Draw Mode

OVERLAY_NORMAL = 0 = Normal draw mode

OVERLAY_XOR = 1 = XOR draw mode

DrawColor: Draw Color

Left: Left position of the overlay

Top: Top position of the overlay

Right: Right position of the overlay

Bottom: Bottom position of the overlay

Return:

Description:

Add an overlay to the OCX image.

void ClearAllOverlay(void)

Parameter: None.

Return: None.

Description: Clear all overlays on the OCX image

void vClearLastOverlay()

Parameter: None.

Return: None.

Description: Clear the last overlay on the OCX image

OCX EVENTS

GetPointPos(int iCursor, int PosLeft, int PosTop, int PosRight, int PosBottom)

Parameter:

iCursor: Cursor function

IMGN_POINT_MOVE = 1025

IMGN_POINT_CLICK = 1026

IMGN_AREA_SIZING = 1027

IMGN_AREA_POSITION = 1028

IMGN_AREA_DONE = 1029

PosLeft: Object Left position

PosTop: Object Top position

PosRight: Object Right position

PosBottom: Object Bottom position

Description:

if the mouse cursor is in the following modes:

MOUSE_MODE_POINT,

MOUSE_MODE_VERTICAL_LINE,

MOUSE_MODE_HORZ_LINE

one will receives the event when

a) the mouse cursor is moving (iCursor = IMGN_POINT_MOVE)

b) the left hand mouse button is pressed (iCursor = IMGN_POINT_CLICK)

if the mouse cursor is in the following modes:

MOUSE_MODE_RECT

MOUSE_MODE_SQUARE

MOUSE_MODE_ELLIPSE

MOUSE_MODE_CIRCLE

one will receives the GetPointPos event when

a) the mouse cursor is moving and an area is not defined (iCursor = IMGN_POINT_MOVE)

b) the mouse cursor is moving while the left hand button is pressed (iCursor = IMG_AREA_SIZING)

c) the mouse cursor is moving and an area is defined (iCursor = IMG_AREA_POSITION)

d) Right hand mouse button is pressed and an area is defined (iCursor = IMG_AREA_DONE)

TOPOGRAPH.OCX REFERENCE

MEMBER VARIABLES

char ***Caption**
Graph caption

int **CurrentSet**
Current set number

int **CursorMode**
Cusror mode

CM_HORI = 100 = Horizontal line mode
CM_VERT= 101= Vertical line mode
CM_VECTOR = 102 = Vector line mode

int **NumPts**
Number of points in the graph

int **NumSet**
Number of sets of graph

char ***XText**
Label for x-axis

int **XYSetting**
flag storing XY setting.

LOG10_SCALE_X = 0x2 = Log scale on X
LOG10_SCALE_Y = 0x4 = Log scale on Y
AUTO_SCALE_X = 0x8 = Do auto scale on X
AUTO_SCALE_Y = 0x10 = Do auto scale on Y

char ***YText**
Label for y-axis

METHODS

void **AutoScaleXY(void)**
Parameters: None
Return: None
Description: Carry out auto scaling and redisplay data

void **ClearMarker(int iMarker)**

Parameters:

iMarker:Marker number

Return:

Description:

Clear the numbered marker

void **SetDataFromTo(long FromPt, long ToPt, double FAR* pData, BOOL bOnlyDraw)**

Parameters:

FromPt:	Starting point
ToPt:	Ending point
pData:	data array
bOnlyDraw:	Reserved, not used

Return:

Description:

Update certain portion of the graph from FromPt to ToPt.

Data is stored as a one dimension array, even elements store the X positions, odd elements store the Y positions.

BOOL **SetGraphData(long NumPoints, double FAR* XYData)**

Parameters:

NumPoints:	Number of points
XYData:	Data array

Return:

True if successful, otherwise False

Description:

Set the graph data. Data is stored as a one dimension array, even elements store the X positions, odd elements store the Y positions.

void **SetGraphLabel(short iNumX, short iNumY, LPCTSTR szXUnit, LPCTSTR szYUnit, LPCTSTR szFormat)**

Parameters:

iNumX:	Number of labels on X axis
iNumY:	Number of labels on Y axis
szXUnit:	X unit string
szYUnit:	Y unit string
szFormat:	unit format

Return:

Description:

Set the graph label

void **SetGraphMarker(double x, double y, short style, short size, OLE_COLOR clr, short num)**

Parameters:

x:	Marker X position
y:	Marker Y position
style:	Marker style
	TRIANGLE_MARKER = 1 = Triangular marker
	SQUARE_marker = 2 = Square marker
	INDEX_marker = 3 = Index marker
clr:	Marker color
num:	Marker number

Return:

Description:

Draw marker on graph.

void SetGraphPt(long PtNum, double XValue, double YValue, BOOL bRedraw)

Parameters:

PtNum:	Point number
XValue:	X value
YValue:	Y value
bRedraw:	True:Redraw graph
False:	Don't redraw graph

Return:

Description:

Modify individual point on the graph

void SetMargin(long left, long top, long right, long bottom)

Parameters:

left:	left margin
top:	top margin
right:	right margin
bottom:	bottom margin

Return:

Description:

Set the graph margin

void SetMode(long iSetNum, long iMode)

Parameters:

iSetNum:	Set number
iMode:	DOT_MODE = 106 = Dot mode
	SHAPE_MODE = 107 = Shape mode
	LINE_MODE = 108 = Line mode

Return:

Description:
Set the graph drawing mode.

void SetShape(long iSetNum, long iMode)
Parameters:

iSetNum:	Set number	
iMode:	SM_UP_TRIANGLE	= 109 = use up triangle
	SM_DN_TRIANGLE	= 110 = use down triangle
	SM_CIRCLE	= 111 = use circle
	SM_SQUARE	= 112 = use square
	SM_DIAMOND	= 113 = use diamond shape

Return:
Description:
Set the shape to use if draw mode is set to shape mode

void SetXYGrid(short XGrid, short YGrid)
Parameters:

XGrid:	Number of grid lines on X direction
YGrid:	Number of grid lines on Y direction

Return:
Description:
Setup the number of grid line on X and Y direction

void SetXYRange(double XMin, double XMax, double YMin, double YMax)
Parameters:

XMin:	Minimum value for X-axis
XMax:	Maximum value for X-axis
YMin:	Minimum value for Y-axis
YMax:	Maximum value for Y-axis

Description:
Manually setup X and Y scale

OCX EVENTS

ClickedOnGraph(int wAction, int iCursorMode, int wNumPts, double fXValue, double fYValue)
Parameters:

wAction:	LEFTBTNDOWN	= 10 = Left mouse button down
	LEFTBTNUP	= 11 = Left mouse button up
	MOUSEMOVE	= 12 = Mouse move
	RIGHTBTNDOWN	= 13 = Right mouse button down
iCursorMode	Cursor Mode, see comment on CursorMode	

wNumPts: Point number
fXValue: Cursor X position
fYValue: Cursor Y position

Description:

when there is a mouse activity while the cursor is in the graph area. The above event will be activated. The mouse activity is given in the parameter wAction.

If the cursor mode is set to either Vertical mode or Horizontal mode, wNumPts will return 1, and when the left hand mouse button is pressed for the first time (or after the right hand mouse button is pressed) , the will then change into either a vertical line or horizontal line and the mouse cursor is locked inside the graph area. Moving the mouse or pressing the left hand button will activate the ClickedOnGraph event. Right hand mouse click will unlock the mouse cursor, change back to a crossbar and activate the ClickedOnGraph event.

When the cursor mode is set to Vector mode. Use the left hand mouse button to define the first point of the vector. Moving the mouse while holding the left hand button will activate the ClickedOnGraph event and a straight line will be draw between the first point and the new cursor position. Releasing the left hand mouse button will reset the mouse cursor to a cross bar and active the ClickedOnGraph event with wNumPts = 2, indicating that this is the second point of the vector.

TOOL32.DLL REFERENCE

Following are those functions in Tool32.DLL that are relevant to SPMTools Analysis.

BOOL bGetImageData(hWnd, row, col, nRows, nCols, Data)

Parameters:

hWnd: handle of the OCX control
row: starting row of image area
col: starting column of image area
nRows: number of rows image area extends to the right
nCols: number of columns image area extends down
Data: data array

Returns:

TRUE: operation successful
FALSE: operation failed

Description:

This procedure returns an area of the image into the data array (Data) as long

integer (32 bit) values. The area returned is the rectangular are defined as starting at row,col and extending nRows to the right and nCols down.

BOOL bSetImageData(hWnd, row, col, nRows, nCols, Data, BOOL bUpdateColorBar):

Parameters:

hWnd: handle of the OCX control

row: starting row of image area

col: starting column of image area

nRows: number of rows image area extends to the right

nCols: number of columns image area extends down

Data: data array

bUpdateColorBar

TRUE: colorbar update successful

FALSE: colorbar update failed

Returns:

TRUE: operation successful

FALSE: operation failed

Description:

This procedure copies the long integer data into the image and displays it on the screen. The destination area is the rectangular area defined as starting at row,col and extending nRows to the right and nCols down.

int iGetVertLineData (HWND hWnd, int iLine, double *pfData, BOOL bFitting)

Parameters:

hWnd: handle of the OCX control

iLine: line Number

pfData: data array

bFitting:

TRUE: performs a first order line fitting

FALSE: no line fitting

Return:

number of data pixels in the line

Description:

This procedure returns a vertical line (line number: iLine) from the image into the data array pfData as floating point value.

BOOL bGetLineData(HWND hWnd, int iLine, double *pfDAta, BOOL bFitting)

Parameters:

hWnd: handle of the OCX control

iLine: line Number

pfData: data Array

bFitting:

TRUE: performs a first order line fitting

FALSE: no line fitting

Returns:

TRUE: operation successful

FALSE: operation failed

Description:

This procedure returns a hortizontal line (line number: iLine) from the image into the data array pfData as floating point value.

int iGetHwndImageResolution(HWND hWnd)

Parameter:

hWnd: handle of the OCX control

Return:

pixel resolution of the image in the OCX control

Description:

This procedure returns the pixel resolution of the image in the OCX control.

void vClearImageData(HWND hWnd)

Parameter:

hWnd: handle of the OCX control

Description:

This procedure clears up the image data and image display.

BOOL bImageToBitmap(HWND hWnd, int bitsize, LPSTR szName)

Parameters:

hWnd: handle of the OCX control

bitsize: pixel dimension

szName: filename

Returns:

TRUE: operation successful

FALSE: operation failed

Description:

This procedure saves the image to a bitmap. It is not necessary for the bitmap size to be the same as the pixel dimension.

TOPOPRO.DLL REFERENCE

STANDARD SPMLAB 5.01 DIALOGS

The following functions call the standard image processing dialogs in SPMLab5.01. Each function takes two parameters. The first parameter is the handle to the calling dialog; the second is the handle of the image ocx. The return value is the handle of the image processing dialog.

Each of these dialogs is explained in greater detail in *SPMLab Version 5.01 Software Reference Manual*. A page number reference to the relevant section of the SPMLab manual is given in parentheses at the end of each of the following descriptions.

DISPLAY

The following dialogs allow you to modify basic visual display characteristics of your image whithout modifying the data.

hShadingDlg:
Opens the Shading dialog, which allows you to shade images, creating a simulated light source which can dramatically enhance image features by creating highlights and shadows (pg. 5-24).

hColorHistogramDlg:
Opens the Histogram dialog, which allows you to use the data's histogram to define the specific portion (or portions) of the Z range where the majority of the color palette will be distributed (pg. 5-30).

hView3DDlg:
Opens the Graphic dialog, which allows the active image to be rotated and displayed from any angle, based on a 3D mesh grid interface (pg. 5-34).

IMAGE PROCESSING

The following dialogs allow you to do image processing on your data.

hLevelingDlg:
Opens the Leveling dialog, which allows you to automatically or manually define a level plane for the display of your data. This allows you to compensate for sample tilt, allowing accurate measurement of Z height across a sample without adding any erroneous tilt information to the data (pg. 5-45).

hFilterDlg:
Opens the Filter dialog, which allows you to filter the image pixel-by-pixel using a low, median, or high filtering process. The function is useful for reducing single-pixel noise or improving feature resolution in some images (pg. 5-56).

hConvolutionDlg:
Opens the Convolution dialog, which allows you to apply smoothing, sharpening, or other filtering to the image. The operation is calculated pixel-by-pixel from either a standard kernel, a user-defined-kernel, or a kernel library (pg. 5-54).

hRotateImageDlg:
Opens the Rotation dialog, which allows you to automatically rotate the displayed image in 90° increments or to manually rotate the image (pg. 5-62).

DATA ANALYSIS

The following dialogs allows you to do data analysis.

hLineToolDlg:
Opens the Line Measurement dialog, which allows a cross-sectional measurement along a user-defined line on the selected image. X, Y, and Z data can be obtained for any position along the line profile, as well as point-to-point distance and angle information (pg. 5-80).

hLineAnalysisDlg:
Opens the Line Analysis dialog, which provides a variety of tools for measuring the characteristics of user-selected line profiles on the active image (pg. 5-86).

hCriticalDlg:
Opens the Critical Dimension dialog, which provides measurement tools to analyze average dimensions of various feature patterns. This can be useful in calculating the average width between a segment of parallel lines, as opposed to measuring the width only at the intersecting cross section of a single line (with the line analysis tools) (pg. 5-111).

hAreaRoughnessDlg:
Opens the Area Standard Rounghness dialog, which allows a roughness analysis function to be applied to the image on the basis of the entire area or a user-defined area of a scan (pg. 5-94).

h2DAreaDlg:
Opens the Area Measurement dialog, which which allows a measurement function to be applied to the image on the basis of the entire area or a user-defined area of a scan (pg. 5-96).

IMAGE LEVELING

One can set up a mask to exclude or include a specific z range or certain region of the image for plane leveling.

The following function sets up the mask. There are two types of mask: Level Mask and Block Mask. Level Mask excludes or includes a specific Z range; block mask is used to exclude or include a specified region.

void vShowMaskOverLay(HANDLE hwnd)

Parameters:

hwnd: handle of the image ocx

Returns:

None

Description:

Overlay the mask on top of the image. The included area is highlighted in green.

BOOL bLoadDLLImageMask(char *szFile, int res)

Parameters:

szFile: File Name

res: Image Resolution

Returns:

True is operation is successful otherwise False

Description:

Load the mask from a file to memory

void vInitializeDLLLevelMask (int res)

Parameters:

res: Image resolution

Returns:

None

Description:

The level mask needs to be initialized it before it can be used. This function set up the memory required by level mask and set all pixel to 0

void vInitializeDLLBlockMask(int res)

Parameters:

Res: Image resolution

Returns:

None

Description:

The level mask needs to be initialized it before it can be used. This function set up the memory required by level mask and set all pixel to 0

void vSetMaskBlock(int PosLeft, int PosTop, int PosRight, int PosBottom, int res, BOOL bInclude)

Parameters:

PosLeft,

PosTop,

PosRight,

PosBottom: Specify the region in pixel

Res: Image resolution

bInclude: Include or exclude the region specified

Returns:

None

Description:

Include or exclude the specified region

void vSetMaskLevel (HWND hwnd, double fMin, double fMax, BOOL bInclude)

Parameters:

hwnd: Handle of the image OCX

fMin: Lower threshold value in current unit

fMax: Upper threshold value in current unit

bInclude: Include or exclude the range specified

Returns:

None

Description:

Include or exclude the specified range.

BOOL bImageLeveling (HWND hWndDest, HWND hWndOrg , int mode, int order, BOOL bMasked)

Parameters:

hWndDest: Handle of the destination image OCX. If it is 0, then the resulting image is stored in the original image OCX

hWndOrg: Handle of the original image OCX.

mode: Leveling mode

LEVELING_2D: 2D leveling

LEVELING_HORIZONTAL: Horizontal leveling

LEVELING_VERTICAL: Vertical leveling

order: Leveling order.

For 2-D leveling it must be smaller or equal to 3

For horizontal or vertical leveling, it must be smaller or equal to 6

bMasked: Masked leveling or not. If selected masked leveling, one

first initialize the leveling mask and the block mask prior to this operation.

Returns:
True if operation is successful, otherwise FALSE

Description:
Function to do image leveling.

IMAGE BUFFER

Every image OCX has its own data buffer storing previously processed image. The buffer is six pages long. The original image is always stored in the first pages, and the rest of the buffer is organized in first in first out manner.

BOOL bSelectPreviousDataBuffer (HWND hwnd)

Parameters:
hwnd: Handle of the image OCX.

Return:
FALSE if the current image is the first element of the buffer, otherwise TRUE

Description:
This function set the previous buffer as the current image.

BOOL bSelectNextDataBuffer (HWND hwnd)

Parameters:
hwnd: Handle of the image OCX.

Return:
FALSE if the current image is the last element of the buffer, otherwise TRUE

Description:
This function set the next buffer as the current image.

BOOL bLoadBinImage (HWND hwnd, HWND hImgWnd, char *szName)

Parameters:
hwnd: Handle of the calling dialog
hImgWnd: Handle of the image OCX
szName: File name

Returns:
True if successful, otherwise False

Description:
This function load a binary file into the image OCX

void vCopyImage (HWND hWndDest , HWND ByVal hWndSrc)

Parameters:

hWndDest: Handle of the destination image OCX

hWndScr: Handle of the source image OCX

Returns:

None

Description:

This function copy the image from one OCX to another.

IMAGE PROCESSING

BOOL bDoConvolution (HWND hWndDest, HWND hWndOrg, int iMatrix, short *iKernel)

Parameters:

hWndDest: Handle of the destination image OCX. If it is 0, then the resulting image is stored in the source image OCX

hWndOrg: Handle of the source image OCX

iMatrix: Dimension of the matrix. One of the following: 3, 5 or 7

iKernel: Pointer to the kernel array

Returns:

True if successful, otherwise False

Description:

Performs convolution filter

BOOL bDoFilter (HWND hWndDest, HWND hWndOrg, int iMatrix, int iMode)

Parameters:

hWndDest: Handle of the destination image OCX. If it is 0, then the resulting image is stored in the source image OCX

hWndOrg: Handle of the source image OCX

iMatrix: Dimension of the matrix. One of the following: 3, 5 or 7

iMode:Filter mode

Returns:

True if successful, otherwise False

Description:

Performs filter operation

BOOL bDoRotation (HWND hWndDest, HWND hWndOrg, double fRot)

Parameters:

hWndDest: Handle of the destination image OCX. If it is 0, then the resulting image is stored in the source image OCX

hWndOrg: Handle of the source image OCX

fRot: Rotation in radian.

Returns:

True if successful, otherwise False

Description:

Perform image rotation

void vNormalizeImage (HWND hWndDest, HWND hWndOrg)

Parameters:

hWndDest: Handle of the destination image OCX. If it is 0, then the resulting image is stored in the source image OCX

hWndOrg: Handle of the source image OCX

Returns:

None

Description:

Normalize the data to full dynamic range

SCALE OPERATION

void vGetZAxisScale (HWND hwnd, int *iUnitType, double *fDACtoWorld, double, *fDACtoWorldZero)

Parameters:

hwnd: Handle of the image OCX

iUnitType: Unit type

ZUNIT_MIRCON = 0 = Micron

ZUNIT_NM = 1 = nM

ZUNIT_ANG = 2 = Angstorm

ZUNIT_NA = 3 = nA

ZUNIT_V = 4 = Volts

ZUNIT_NA_V = 5 = nA/V

fDACtoWorld: Scale factor from DAC unit to real world unit

fDACtoWorldZero: Zero offset

Returns:

Description:

Get the Z Axis information from the image OCX.

Real world unit = DACValue * fDACtoWorld + fDACtoWorldZero

void vGetZunitStr (int iUnitType, char *szZUnit)

Parameters:

iUnitType: Unit Type: See pg. function above

szZUnit: Z unit in text format.

Returns:

Description:

Get the Z unit string in text format.

void vGetPixelSize (HWND hwnd, double *fPixelSize, int *iUnitType)

Parameters:

hwnd:	Handle of the image OCX
fPixelSize:	Pixel Size in world unit
iUnitType:	Unit type
	XYUNIT_UM = 0 = micron
	XYUNIT_NM = 1 = nm
	XYUNIT_ANG = 2 = Å

Return:

Description:

Get the X-Y axis information from the image OCX.

void vGetXYunitStr (int iUnitType As Long, char *szXYUnit)

Parameters:

iUnitType:	Unit Type: See pg. function above
szXYUnit:	XY unit in text format.

Returns:

Description:

Get the XY unit string in text format.

void vGetDACMinMax (HWND hwnd, int *iMin, int *iMax)

Parameters:

hwnd:	Handle of the image OCX
iMin:	Min DAC value

Return:

in parameter

Description:

Search the image and return the max and min value

void vNewDACMinMaxImage (HWND hWndDest, HWND hWndSrc, int iMin, int iMax)

Parameters:

hWndDest:	Handle of the destination image OCX. If it is 0, then the resulting image is stored in the source image OCX
hWndSrc:	Handle of the source image OCX
iMin:	Min DAC value for picture display
iMax:	Max DAC value for picture display

Return:

Description:

By default, the computer search the image and use the min and the max DAC

value as the lowest point and the highest point for image display. Use this function to change the lowest and highest point manually.

BOOL bRescaleXY (HWND hwnd, int iUnit, double fRange)

Parameters:

hwnd: Handle of the destination image OCX

iUnit: Unit type

XYUNIT_UM = 0 = micron

XYUNIT_NM = 1 = nm

XYUNIT_ANG = 2 = Å

fRange: Image size

Return:

True if successful, otherwise False

Description:

Manually, overwrite the XY scale information

BOOL bRescaleZ (HWND hwnd, int iUnit, double fDACtoWorld, double fDACtoWorldZero)

Parameters:

hwnd: Handle of the destination image OCX

iUnit: Unit type

ZUNIT_MIRCON = 0 = Micron

ZUNIT_NM = 1 = nM

ZUNIT_ANG = 2 = Angstorm

ZUNIT_NA = 3 = nA

ZUNIT_V = 4 = Volts

ZUNIT_NA_V = 5 = nA/V

fDACtoWorld: Scale factor from DAC unit to real world unit

fDACtoWorldZero:Zero offset

Return:

True if successful, otherwise FALSE

Description:

Manually, overwrite the Z scale information