SEM-VE

Application Programming Interface

v 1.0

Prepared by
Fibics Incorporated
556 Booth St. Suite 200
Ottawa, Ontario
Canada K1A 0G1
1-613-860-0861
info@fibics.com

For
Carl ZEISS SMT
Version 2010-April-26

This document describes the SEM-VE Application Programming Interface which allows external programs to communicate with the SEM-VE software.

Note: To make effective use of this interface, the user is required to have a firm knowledge of Microsoft Windows™ programming and the Microsoft Component Object Model (COM). Further information is available at http://www.microsoft.com/com **and in numerous third party publications.**

1. Introduction

This document is intended for third party developers wishing to write applications that communicate with the SEM-VE application to acquire high resolution images. The Application Programming Interface (API) consists of an Type Library, registered as "*NPVE3Z Library*" that provides access to the SEM-VE application.

As a COM Type Library the API can be accessed by various programming languages such as Visual Basic, Visual C++ and various scripting languages (VBScript, JavaScript).

The API exposes a COM IDispatch interface with the following properties and methods. When the COM object is created programmatically, it will automatically start the SEM-VE application if it is not already running. The functions and properties can then be called by the third party application. Upon destruction of the object, the SEM-VE application will be closed if it was started by the third party application.

2. Installing the NPVE3Z Library

The NPVE3Z Library is part of the SEM-VE or NPVE application. Before using the API, make sure the SEM-VE application is properly installed and that the system is calibrated. The NPVE3Z Library is automatically registered when the SEM-VE or NPVE applications is started.

3. Using the API

1. *Using the Type Library with Visual Studio 2008 in Visual C#*

Create a new Windows Forms project.

To add a reference to a type library:

From the Project menu, select **References**.
Select the **COM** tab.
Select the "***NPVE3Z Library***" type library from the list
Click **OK**.

In your application, define a variable of type NPVE3Z.IFibicsSEMVE

```
NPVE3Z.IFibicsSEMVE veobj;
```

and at some point during the initialization of the application, assign it as

```
      veobj = new NPVE3Z.FibicsSEMVE();
```

You can then use this object to call any of the API functions described in the following section. The following code excerpt shows a simple example of how to create the object and call the AquireImage function:

```
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    NPVE3Z.IFibicsSEMVE veobj;

    public Form1()
    {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
      try
      {
        veobj = new NPVE3Z.FibicsSEMVE();
        button2.Enabled = true;
        button1.Enabled = false;
      }
      catch (Exception ex)
      {
        MessageBox.Show("Fibics SEM-VE Object creation failed: " +
                        ex.Message);
      }
    }

    private void button2_Click(object sender, EventArgs e)
    {
      try
      {
        veobj.AcquireImage(Convert.ToInt32(textBoxWidth.Text),
                           Convert.ToInt32(textBoxHeight.Text),
                           Convert.ToSingle(textBoxDwell.Text),
                           textBoxFileName.Text);
        while (veobj.Busy)
          System.Threading.Thread.Sleep(200);
        MessageBox.Show("Image complete");
      }
      catch (Exception ex)
      {
        MessageBox.Show("Image acquisition failed: " + ex.Message);
      }
    }
  }
}
```

2. *Using the Type Library with Delphi (BDS 2006)*

Create a new VCL Forms Application.

From the **Component** menu, select **Import Component…**
Select **Import a Type Library**
Select *NPVE3Z Library* from the list
In the **Palette Page** list, type "Fibics", this will cause the **Generate Component Wrappers** to be checked
Select **Create Unit** and press **Finish**.

When this is done, you can select the FibicsSEMVE object from the **Tool Palette** and drop it onto your main form or create an IFibicsSEMVE object programmatically and call the API functions it exposes, as illustrated in the following short example.

```
var
  SEMVE: IFibicsSEMVE;

procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    SEMVE := CoFibicsSEMVE.Create();
    Button2.Enabled := True;
  except
    on e: Exception do
      ShowMessage('Fibics SEM-VE Object creation failed: '+e.Message);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  try
    if (SEMVE <> nil) then
      SEMVE.AcquireImage(StrToInt(WidthEdit.Text),
                         StrToInt(HeightEdit.Text),
                         StrToFloat(DwellEdit.Text),
                         PChar(FileEdit.Text));
    while (SEMVE.Busy) do
    begin
      Application.ProcessMessages();
      Sleep(200);
    end;
    ShowMessage('Image Complete');
  except
    on e: Exception do
      ShowMessage('Image acquisition failed: '+e.Message);
  end;
```

```
end;
```

4. The API Interface

Properties

```
float FOV;
```

read returns the current FOV (field of view) in microns.
write sets the current FOV in microns.

```
bool Busy;
```

read indicates if the scan generator is imaging (TRUE) or Idle (FALSE)

Methods

```
void AcquireImage([in] long W, H; [in] float Dwell; [in]
LPSTR Filename);
```

`W` and `H` are the width and height of the image in pixels. Image pixels are square, so this also defines the aspect ratio of the image. For example, 16384 x 16384 results in a square image, but 4000 x 32768 results in a thin vertical image with an aspect ratio of approximately 1:8. The pixel size will be `FOV`/max(`W`, `H`).

`Dwell` is the dwell time for each pixel, in microseconds.

`Filename` is the local filename where the image will be stored. The image is stored as a TIFF file with two pages: the first page contains the high resolution image and the second contains a 2048 x 2048 binned low resolution image.

Start acquiring a high resolution image with the given parameters and stores the resulting image in the given location. If `AcquireImage` is called when an image is already being acquired, the current image will end and a new one will be acquired with the new parameters. In general, Busy should be polled to determine when the image is complete as the function returns once the imaging starts.

```
void Cancel();
```

Stops scanning immediately. The resulting image is still written to the file but will be completed with blank data.

Appendix A. NPVE Library Definition in IDL Format

```
[
  uuid(D8FCB34B-2A2F-47E6-8FAE-D2B366076FC7),
  version(1.0),
  helpstring("NPVE3Z Library")
]
library NPVE3Z
{
  importlib("stdole2.tlb");
  [
    uuid(4D3D2A4C-CB90-4A5F-B92A-5A23C24B8E0D),
    version(1.0),
    helpstring("Dispatch interface for FibicsSEMVE Object"),
    dual,
    oleautomation
  ]
   interface IFibicsSEMVE: IDispatch
  {
    [    propget,     id(0x000000C9)    ]
    HRESULT _stdcall FOV([out, retval] float * Value );
    [    propput,     id(0x000000C9)    ]
    HRESULT _stdcall FOV([in] float Value );
    [    id(0x000000CA)    ]
    HRESULT _stdcall AcquireImage([in] long W, [in] long H, [in] float
Dwell, [in] LPSTR Filename );
    [    propget,     id(0x000000CB)    ]
    HRESULT _stdcall Busy([out, retval] VARIANT_BOOL * Value );
    [    id(0x000000CC)    ]
    HRESULT _stdcall Cancel( void );
  };

  [
    uuid(405076F3-8B40-45EB-9291-C2692AE983D8),
    version(1.0),
    helpstring("Events interface for FibicsSEMVE Object")
  ]
   dispinterface IFibicsSEMVEEvents
  {
    properties:
    methods:
  };

  [
    uuid(76EAEE96-7FCC-4888-928C-F2CC11D84FE9),
    version(1.0),
    helpstring("FibicsSEMVE Object")
  ]
  coclass FibicsSEMVE
  {
    [default] interface IFibicsSEMVE;
    [default, source] dispinterface IFibicsSEMVEEvents;
  };
};
```