Script Programmers Manual

Table of Contents

1 Frontpage

for Nanosurf Scripting Interface

v3.10.1.x

BT01681, v3.10.1.x

2 Introduction

This manual is meant as a reference for the COM Automation interface of the Nanosurf software. This manual consists of two parts. The first part contains an explanation of the concepts behind the interface. This part should be read entirely to understand the concept of COM Automation, scripting and how it is implemented in the software. The second part is a object reference of all classes with their method and properties published by Nanosurf. In this part it is recommended to read the entry page for each class to get an overview what these classes functionality is. Afterwards it can be read method or property wise, when a exact understanding of specific functions is needed.

This manual does not describe general usage of the microscopy. Please read for general understanding the Nanosurf Operating Instruction Manual and the Nanosurf Software Reference Manual.

2.1 Motivation

Microscopy is a wonderful technology with a large amount of possibilities for data analysis. The Nanosurf control software tries to offer a graphical user interface to the most general tasks used by operators in a daily manner. Nevertheless, there are many thinkable tasks specific for a single application used only by a small group of users. To integrate these functions into the core of the software would blow it up and the simplicity would fade away. Other groups of users are very advanced and like to write custom analysis or automation sequences. They need a way to do their new experiments. Third, some would like to integrate or combine the microscopy with other equipment like motorized sample stages, manufacturing equipment, scratch testers or others. They also need a possibility to let the different instruments work smoothly together and act as one new machine.

Therefore Nanosurf has developed an scripting interface and new menu items to the control software to help all group of users. The users which are interested to automate daily tasks are able to write a script once which defines the custom task. The script is called comfortably by a click of the mouse from the pull down menu. The advanced users would like the integrated script editor to program new measurement modes or create their own analysis algorithms. Integrators possibly will use the external script interface to write complete new interfaces or control the microscopy out of another software like LabView and Python.

2.2 What you can do

The script objects give you access to online microscope controls as scan range or feedback set point. Other objects serve for post processing of data and control the visualization of them. A script may extract measured data, create new data and store it in a image document. Most of the user interface data entry fields of the panels are accessible as object properties.

You may call methods from other objects like windows operating system objects, Internet Explorer, Microsoft Word and many other vendors applications, ...

Many possible applications for scripting are:

- Automation of repetitive tasks like
  - Scripts which Approach to sample, take an image, and store it
  - Scripts which loads special parameter sets and start a process for quality control purpose

- Write custom data analysis algorithms
  - Scripts which calculate the volume of a hole in a image or count grains
  - Scripts which calculate height histogram or subtract two images
  - Scripts which calculates calibration information from a spectroscopy measurement

- Extending the functionality
  - Scripts which measure multiple images at the same position every 30min and store them
  - Scripts which measure large high resolution images as a patch work and plot them in one resulting image
  - Scripts which provide lithography functionality and control the tip position

- Building complex new systems
  - External scripts which controls an automated XY-table and moves the microscope to different image locations
  - Scripts which control additional experiment equipment like a temperature controller or light sources

More ideas you will find in the chapter Scripting Examples

We hope we could give you some ideas what can be done with the scripting interface. Try it out and create new applications!

2.3 What you cannot do

With the scripting interface you gain access to internal functions and data of the microscopy control software "Nanosurf". This is the PC part of the microscopy control software which provides access to microscope functionality and post processing of stored image documents. For real time controlling of the microscope itself an external control electronics with its own software in a flash RAM is used. The script interface does not give you access to this firmware.

Therefore real time processing or signal modification is not possible. You cannot create new z feedback control algorithm, real time filtering of signals or create custom new operating modes.

2.4 How to proceed

Depending on your knowledge of scripting under Microsoft Windows operating system you may need to read some chapter carefully or just skip them:

- This Introduction chapter gave you an overview of the possibilities of the scripting interface
- Chapter Scripting describes the general concept of scripting technology.
- With chapter Integration you learn how to integrate the software with other application.
- The Tutorial is a step by step example of a short script.
- More examples are provided in chapter Script examples.
- Finally chapter Object Reference describes all properties and method of Nanosurf script classes.

3 Scripting

In this chapter we will look to the embedded script command interpreter. The connection with external programs is described in chapter Integration.

The term "scripting" means adding functionality to an existing application from external sources at run time. Such sources can be another running program or at run time by a embedded command interpreter the application itself.

3.1 Embedded VBScript

In the Nanosurf software a command language interpreter is built in, called "VBScript". This programming language was defined by Microsoft for the main usage of building interactive HTML web pages. It supports a subset of Visual Basic commands and features. A formerly known similar programming environment was "Visual Basic for Application", in short VBA, which was implemented in old versions of Word or Excel.

A basic hello world program example looks like this:

```
' start of script
msg = "Hello World!"
MsgBox msg
'end of script
```

Copy this example into the Script editor and click "Run" (See Script editor).

The functionality of the microscope application is grouped into object of different classes. Each class provides some properties and methods to get access to the application internals. There is a main object called "SPM.Application". This object is automatically defined if you run your script from the embedded script editor or menu item "Script".

Otherwise you have to create one with function **CreateObject("Nanosurf_C3000.Application")**.

A full description of the available classes with their methods and properties you find in chapter Object Reference.

To just simulate a click to the "Start" button in the "Imaging Window" see the following example:

```
' connect to scan object
Set objScan = SPM.Application.Scan
' call start method
objScan.Start
'disconnect from scan object
set objScan = nothing
```

Copy this example into the Script editor and click "Run" (See Script editor).

Go to More Documentation and find links to sources where VBScript is explained.

3.2 More Documentation

We cannot give you a full overview of the scripting. Also describing the full language of VBScript would go over the focus of this manual. But there are many good resources on the internet which can guide you. Here are some useful links:

VBScript tutorials and function references:

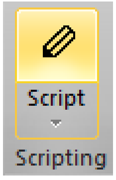
www.w3schools.com
https://www.devguru.com/content/technologies/vbscript/home.html

Scripting technology and references from Microsoft:

https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc784547(v=ws.10)

3.3 Menu Script

To work with scripts there is the ribbon group "Scripting" in the Nanosurf software.



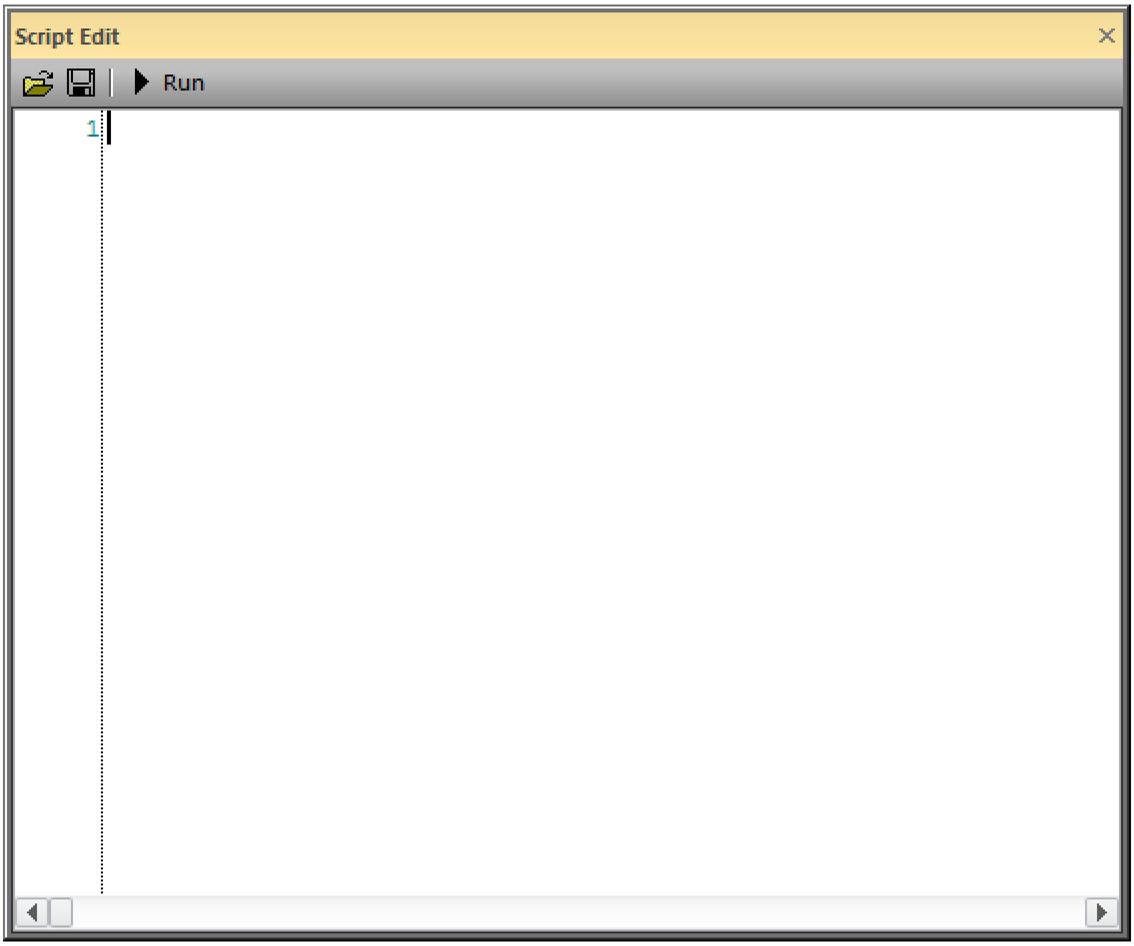
- The Nanosurf has an integrated script editor where you can develop your scripts and run them. See Script editor section for details.
- Scripts can also be written in an external standard text editor like Notepad. They have to be saved with file extension .vbs to be recognized as scripts by the application. To run such stored scripts call menu "Run form file" (See Run from file).
- If script files are placed in a special directory they appear as menu item in menu "Script" (See Scripts as menu items).

3.3.1 Script editor

The Nanosurf has an integrated script editor where you can develop your scripts, run, load and save them.

Call Menu "Script"->"Scrip Editor" and a dialog appears. This dialog is mode less and stays open while you can work with other parts of the application.

Script Editor Dialog:

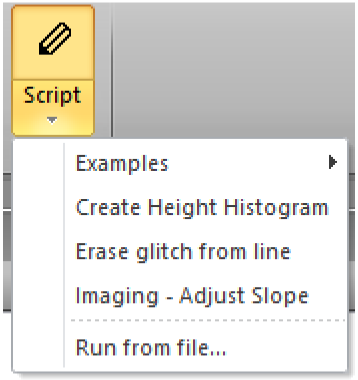


In the editor field you can write scripts and run them immediately.

To store the script permanently click "Save...", to load another script from file into the editor click "Load...".

3.3.2 Run from file

With menu item "Script"->"Run form file..." you get a quick access to stored scripts.



Select in the in the appearing file dialog the desired script file and click "Load". The script will be loaded and run directly. If an error is detected in the script a dialog will appear with a description.

3.3.3 Scripts as menu items

To get even more quick access to stored scripts it is possible to display script file names in the pull down menu "Script" as menu items.

If you click on one of these menu items the script will be loaded and executed immediately.

In the example above the file "Test Script.vbs" is displayed in the menu as an item. If you click on it file "Test script.vbs" will be loaded and executed.

If an error is detected in the script a dialog will appear with a description.

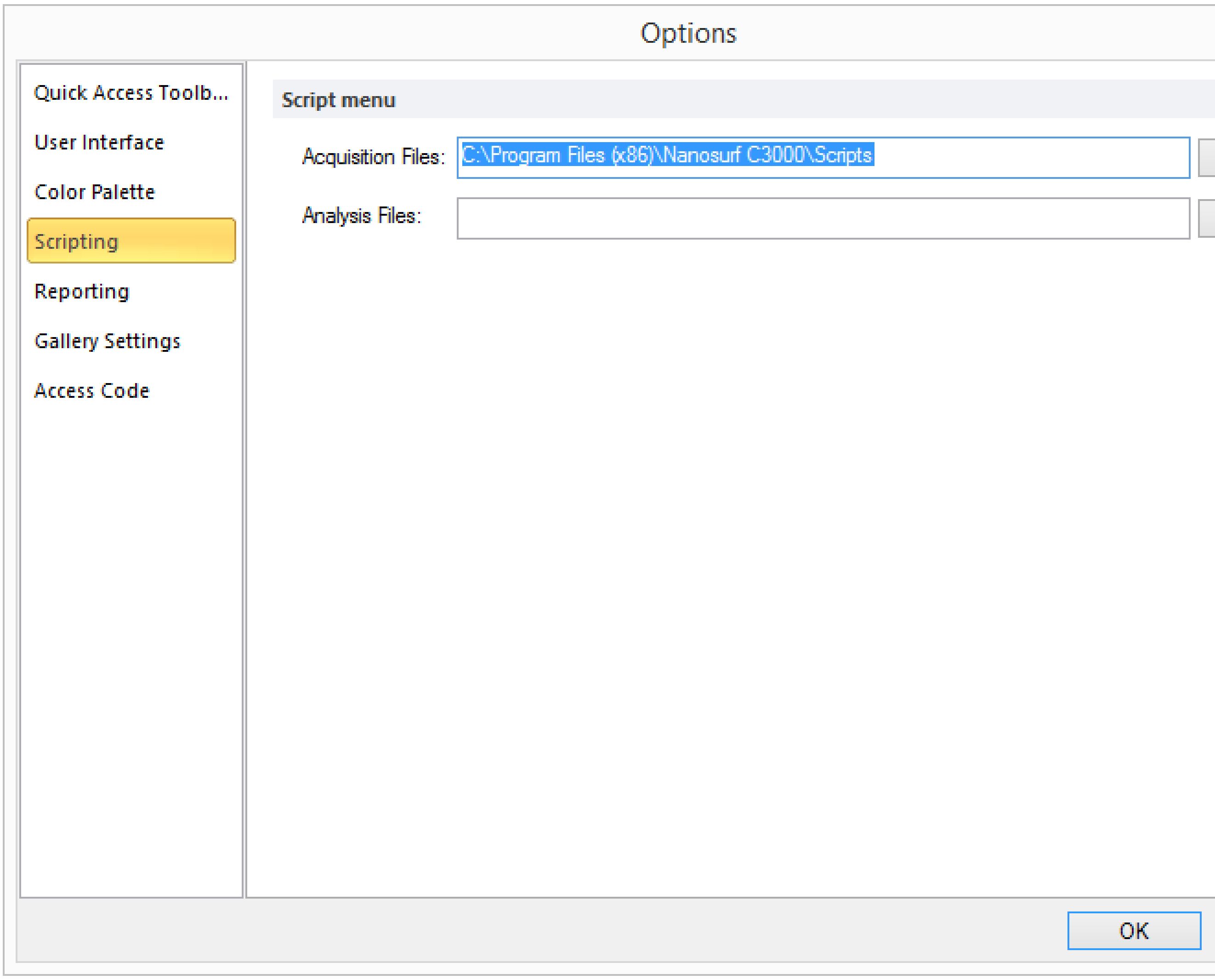
The files which are displayed in the pull down menu have to be stored in a special directory. The directory name can be defined with the Script configuration dialog.

3.3.4 Script configuration

The quick access script files which are displayed in the pull down menu as items have to be stored in a special directory.

To tell the application your script menu folder, open the configuration dialog with Ribbon

"File", Menu "Options", Item "Scripting":



Enter a valid directory name in the edit field or select one by click to "Browse".

Leave the dialog by a click to "OK".

All files with the extension ".vbs" in this selected directory are displayed now in the pull down menu "Script". See Scripts as menu items.

4 Integration

To control the Nanosurf software from an external program the application can act as a server according to the COM Automation standard defined by Microsoft. Many programming environments and software packages are able to access the application as a client through this interface standard:

Some programming environments:

Visual C++, Visual Basic, Delphi, Windows Scripting Host, LabView, ...

Other software packages:

MathLab, MathCAD, Excel, Word, Internet Explorer, ...

Most of the scripts written for embedding into the application can be called with minor or no changes with the help of the Windows Scripting Host (short WSH) which is part of the Windows operating system. If you double click to a vbs-file you start the application WScript.exe and the script is interpreted there (See Windows Scripting Host).

For some programming environment the following sections give a quick guide on how to interface to the COM Automation server.

4.1 COM Automation

The abbreviation COM stands for 'Component Object Model', which is a Microsoft standard for building interoperable software components. The COM standard describes how a program (called server) can publish its functionality to other programs (called client). The clients can then use the functions of the server using this published information. The functionality can even be used if the client and server are on different computers, connected by a network, independent of the programming language in which the programs were written.

The COM automation standard is defined using the COM standard. The COM automation standard was necessary because the basic COM standard only defines the internal principle how to access the functions of a server by a client. But the client needs prior to its own compilation the information about the servers function details in order to be able to access them. This is a problem for scripting languages like Visual Basic or other programs like LabView which should be able to access unknown servers during run time. This problem was solved by the COM Automation standard.

A COM Automation Server publishes its functionality in such a way that COM Automation Clients can ask the server during run time about its functions and access them afterwards. Microsoft defined for this purpose the Dispatch interface definition. The Dispatch information about the servers function are stored in the servers exe-file, and in a binary file with the extension '.tlb' which can be loaded by a client if early binding is necessary or to build class wrapper.

The Dispatch interface of the Nanosurf software is defined in the file "Nanosurf_C3000".tlb.

The root interface is named "Nanosurf_C3000.Application" and is the only named interface which can be created by CreateObject(). All other sub objects are created by this root object.

4.2 Windows Scripting Host

You can control all of the functionality of the Nanosurf from a windows shell script. In newer version of the Windows operating systems (starting from Windows 98/2000) Microsoft distributes the so called Windows Scripting Host (WSH). With the WSH you are able to write shell scripts in a language like Visual Basic Script (.vbs, VBScript) or JavaScript (.js). VBScript is also used in applications like Internet Explorer, Word or Excel to give to user the possibility to enhance the functionality of this software package.

You can use either the window based host WScript.exe or the command shell host CScript.exe to execute scripts.

There are many documentation about the windows scripting host as books or online. See More Documentation.

Scripts have to be stored in files. The extension of the file defines the program language the scripting host is using.

Example

1. Open a Editor (e.g Notepad.exe) and copy the following script text into it:

```
' VBScript example: Measure an image
'-----------------------------------

' connect to microscope
Dim objApp  : Set objApp = CreateObject("Nanosurf_C3000.Application")
objApp.Simulation = True
Do While objApp.IsStartingUp : Loop

'scan an image
Dim objScan : Set objScan = objApp.Scan
objScan.Lines = 16
objScan.Scantime = 0
objScan.StartFrameUp
Do While objScan.IsScanning : Loop
objScan.StartCapture

'disconnect from objects
Set objScan = Nothing
Set objApp = Nothing
```

2. Save the script to a file. Name the file "MyScript.vbs"
3. Open the File Explorer and navigate to the stored file.
4. Double click on icon "MyScript.vbs"
5. WScript.exe should be executed and run your script.

6. The Nanosurf should start and a quick dummy image should be measured

4.3 Visual C++

This section describes how to integrate the Nanosurf object interface with Visual C++ 6.

Visual C++ 6 provides a wizard to integrate the Nanosurf object interfaces in an application. The wizard generates for each COM interface a C++ wrapper class. The information about the COM interface reads the wizard from the Nanosurf_C3000.tbl file. This file is distributed with the installation of the application.

If you would like to call some methods or properties from an application follow these steps.

- Create a new dialog based project. Make sure that "Automation" in the Project Wizard Step 3 is activated.
- Start the wizard. Open the "Class Wizard" and click on "Add class...", select "From a typlibrary...". In the "File Dialog" select the Nanosurf_C3000.tbl from the C:\Program files \Nanosurf\Nanosurf\Bin directory. In the next dialog all available interfaces from Nanosurf_C3000 are displayed and selected. Click "OK" to accept the names.
- Your project should have now new classes called IProxyXXXXX visible in the class tree
- Add the variable `IProxyApplication m_objApp` to the dialog class definition and insert `#include "Nanosurf_C3000.h"` at the beginning.
- In the OnInitDialog() function connect to the microscope with the following code

```
m_objApp.CreateDispatch("Nanosurf_C3000.Application");

while (m_objApp.IsStartingUp() != FALSE) ;
```

To call any method call *obj.***Methodname***(arguments)*

To set a property call *obj.Set***Propertyname***(value)*

To read a property call *value* = *obj.Get***Propertyname***()*

To connect to a subclass of the Nanosurf define a variable of this type and attach the return value of the objApp.GetClassname() function to it. After usage of a class call `DetachDispatch()`.

Example:

```
// dialog class header
#include "Nanosurf_C3000.h"

CMyDialog {
  ....
```

```
  IProxyApplication m_objApp;
  IProxyScan m_objScan;
};

// dialog class implementation cpp-file
CMyDialog::OnInitDialog() {
  ....

  // connect to server
  m_objApp.CreateDispatch("Nanosurf_C3000.Application");
  while (m_objApp.IsStartingUp() != FALSE) ;
  m_objScan.AttachDispatch(m_objApp.GetScan());
  m_objScan.SetScantime(0.5); // [s]

  ....
}
```

4.4 Labview

Use LabView's ActiveX function blocks in the diagram of your virtual instrument to control the functionality of the Nanosurf. Four function block types are needed:

• The 'ActiveX Open'-block to start the Nanosurf Server program

• The 'ActiveX Close'-block to stop the Nanosurf Server after executing the VI.

• The 'ActiveX Method'-block to call the Nanosurf methods to send it commands.

• The 'ActiveX Property'-block to read or write the Nanosurf properties to change and/or read its configuration and status information .

Follow the procedure below on how to wire a ActiveX diagram:

- First, a connection between LabView and the Nanosurf software is established using the 'ActiveX Open' function block.
- Place this block from the palette 'Functions->Communication->ActiveX'. Now connect the block to the Nanosurf Software:
- Clicking the ActiveX Open block with the right mouse button and selecting the menu item 'Select ActiveX...->Search'.
- Click the 'Browse' button in the dialog to search for the Nanosurf's type library with the filename 'Nanosurf_C3000.tlb'. This file is located in you Nanosurf installation directory, which typically is 'C:\program files\Nanosurf\Nanosurf\Bin'. A list of creatable objects is opened after selecting this file. This list contains the name 'Nanosurf_C3000.Application' as creatable object.
- Select 'Nanosurf_C3000.Application' and click 'OK'. The object is now connected to the 'ActiveX Open' block. The outputs of this block should be connected to the corresponding inputs of the other ActiveX function blocks. The example program uses the Nanosurf_C3000 automation server properties to read or write the status and settings of the Nanosurf. In order to do this, create an 'ActiveX Property' function block and connect it to the 'ActiveX Open' block:
- Create the block analogous to the 'ActiveX Open' function block.

- Select the specific property by clicking the lower part of the 'ActiveX Property' block with the right mouse button, and select a property
- from the list in the 'Property>' submenu.
- Select whether to read or write the property using the menu item 'Change to read' or 'Change to write' in the same submenu. The current read/write status of the property is indicated by a small arrow.
- The procedure is the same for method calls: Insert the block 'ActiveX Method', wire it and select the desired method in the pop up menu. Take care to only call a method at a timed interval, or a specific event, do not call it continuously.
- To close the Nanosurf Server you place the function block 'ActiveX Close' in the diagram and wire its two inputs to the corresponding outputs of the 'ActiveX Open' block.

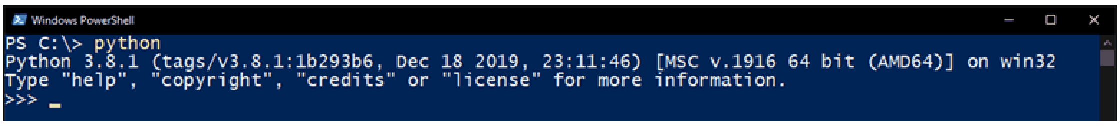
Refer to your LabView documentation and examples on ActiveX for more detailed description on how to use the ActiveX function blocks.

4.5 Python

This section describes how to use Python to control Nanosurf instruments. The Python scripts were tested with Python 3.8.

Quick installation procedure:

1. Ensure that Python is installed on the control computer. Windows 10 has Python in Windows Store, but this source should not be used for our purpose. Instead, use the latest Python release from www.python.org or Anaconda Python. Make sure, it is installed for the current user, and not for all the users (requires administrator rights). To test your Python installation, open the Windows Command Prompt or the Windows PowerShell, type *python* there and press Enter. You should see a Python prompt with the version number.



2. Install the Nanosurf Python module from PyPI, by opening the Windows Command Prompt or Windows PowerShell and executing:

```
pip install nanosurf
```

   or, if pip does not work due to network restrictions, by downloading the PyPI package, unzipping it into a folder, and from this folder executing:

```
python setup.py install
```

3. Start the Nanosurf software, make sure it is communicating with the controller

(although the basic functionality would also work in the simulation mode).

4. Check that a valid “Scripting Interface” code is entered in the Nanosurf software, under File -> Options -> Access Codes.

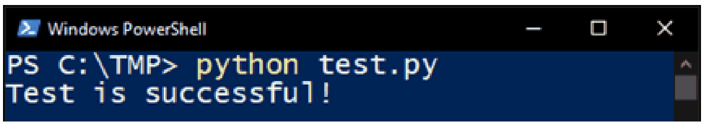
5. Python scripts can be edited with Notepad and executed in a Windows Command Prompt, but we suggest using Visual Studio Code editor (code.visualstudio.com), or any other code editor.

Example script:

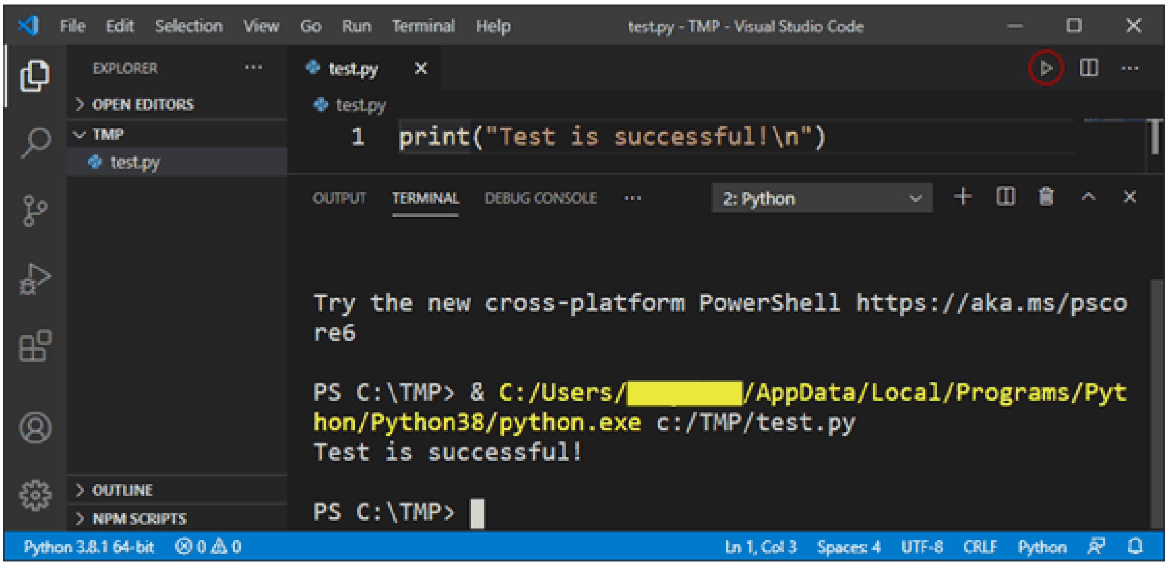
```
import nanosurf

# Create control object for the Nanosurf SPM controller.
spm = nanosurf.SPM()
application = spm.application
application.AutoExit = False

# Creating various objects for the system control
scan = application.Scan
zcontroller = application.ZController

# For example, we would like to change the Z controller settings
zcontroller.SetPoint = 70 # Set the setpoint to 70%
zcontroller.PGain = 3100 # Set P-gain to 3100
zcontroller.IGain = 3500 # Set I-gain to 3500

# Change the Scan settings
scan.ImageWidth = 10 * 1e-6 # Set width of scan to 10 um
scan.ImageHeight = 10 * 1e-6 # Set height of scan to 10 um
scan.CenterPosX = 1 * 1e-6 # X offset = 1 um
scan.CenterPosY = 5 * 1e-6 # Y offset = 5 um
scan.AutoCapture = True # Turn on end-of-frame data capture
scan.Start() # Starts scanning
```

To run the script, open your favorite terminal in the folder with the script and execute:

```
python your_script_name.py
```



Alternatively, use the terminal in the VSCode, or simply click the ▷ icon in the top right corner.



For a full list of the objects, and their methods and properties, read the Chapter 7. Object Reference.

4.6 Others

The integration procedure in third party programs are different but mostly follow a common structure.

If a program support COM Automation it either can call the server command directly during runtime with late binding like Visual Basic or can create some wrapper class or object with the help of the "Nanosurf_C3000.tlb" file:

- With most of the interpreter languages like Visual Basic, JScript, Mathlab or Python calling a COM Server object is done by defining a object variable and call a function like CreateObject(), CreateDispatch() or similar.

- Other compiled programs created with languages like Visual C++ or Delphi you have to first create a proxy class in the language itself. Most development platform help the programmer with a wizard to do this. The information for the proxy classes is extracted from the file "Nanosurf_C3000.tlb" installed with the Nanosurfapplication itself in the C:\Program files\Nanosurf\Nanosurf\Bin directory.

For more help read the documentation of your client application. if you do not found the corresponding chapter easily search for keywords like "COM Automation", "ActiveX", "OLE" or "Dispatch".

5 Tutorial

This chapter is a step by step tutorial which shows you the basic elements of a script and how to control the microscope.

After the tutorial you should be able to write your own scripts and know how to use the properties and functions of the Nanosurf software. You can then start exploring the object reference chapter to learn all the details.

5.1 Script "AutoImage"

The tutorial script "AutoImage" is a example script which shows basic operating concepts of the microscope. It performs an fully automated approach, measure a topography image, calculates the min and max values and save the image into a document file.

The script is very modular and many passages can be reused in your own scripts. It shall help you as an starting point for own script. More scripts you will find in the chapter Script examples.

The script can be executed in the simulator or on a real sample. As a sample we use the 10um calibration grid found in your Toolbox. If you use a High Resolution Scanner the scan range will be automatically reduced.

To follow the tutorial enter new script code step by step in the embedded "Script Editor" (See Script editor) or in an external editor like Notepad.

The script will be developed and discussed in 7 steps

1. Step - Start the application, create the needed objects and release them
2. Step - Prepare the measurement, set operating mode and Z-Controller settings
3. Step - Approach to surface
4. Step - Scan an image
5. Step - Withdraw from surface
6. Step - Calculate the min and max z height value and display the result
7. Step - Save the image in a document to disk

If you do not like to type in the source by your self you find the source in the directory:

C:\Program files\Nanosurf\Nanosurf\Scripts\Examples

5.2 Start the application

Step 1 Start the application

First of all we will write a program version header and force the interpreter to allow only predefined variables. This help avoiding typing error bug which are difficult to find.

```
'-----------------------------------------------------------
' Prog: AutoImage  - Fully automated measurement of a image
'-----------------------------------------------------------
' Version 1.0 Nanosurf
```

```
'-----------------------------------------------------------
Option Explicit
```

Then we need access to the methods of the application. Therefore we create a object variable with the root class "Application". If the application is not already started this will start the software. Then we wait until the application is ready and have connected to the Controller. This is done with our first usage of an internal method the application is providing to us IsStartingUp. If you would like to get a full description about this method read the description in the Object Reference Chapter section Class Application.

```
' startup application and get all needed objects
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
Do While objApp.IsStartingUp : Loop
```

Next we create object to all the sub modules we what to use. This will be the Approach class for approaching, the Scan class for imaging, the OperatingMode class for setting up the preferred mode and the z-controller class for defining setpoint etc. Our root object can give us object variable to all theses classes.

```
Dim objAppr   : Set objAppr   = objApp.Approach
Dim objScan   : Set objScan   = objApp.Scan
Dim objOpMode : Set objOpMode = objApp.OperatingMode
Dim objZCtrl  : Set objZCtrl  = objApp.ZController
```

Again if you like to know more read the section Class Application.

Now we let some space for the code from step 2 to 7.

```
' insert code for step 2 - 6 here
```

At the end of the program listing we need to tell the application that we do not need the object any longer and we free the object variable in the opposite order as we created them.

```
MsgBox "End of script"

Set objZCtrl  = Nothing
Set objOpMode = Nothing
Set objScan   = Nothing
Set objAppr   = Nothing
Set objApp    = Nothing
```

Its time to save our work. Click "Save"-Button and call the file "AutoImage Tutorial.vbs". The ".vbs" is important. This marks the file as a VBScript executable.

Now we would like to test the code we just wrote and run it.

If you wrote your script in the Script Editor Dialog please click "Run". The Position and the

Imaging Window should open and a message dialog telling "end of script". If there where mistyping errors a dialog with a error message should appear.

If you wrote your script in an external editor, double click the saved file in the explorer. The Nanosurf application should start and the starting up dialog should appear. The Position and the Imaging Window should open and a message dialog telling "end of script". If there where mistyping errors a dialog with a error message should appear.

In case of an error message return to the source code navigate to the reported text line and correct the error. Save the file and run it again. Repeat this until no error occurs anymore.

You are prepared now for Step 2 - Preparing the measurement

5.3 Preparing measurement

Step 2 Preparing the measurement

We write now the code for setting up everything right to be able to approach afterwards.

We will now take use of the created objects from Step 1 and define our desired operating mode condition and z-controller settings useful for measuring on the 10um calibration grid. To do this we will write values to some properties of the class OperatingMode and ZController. Detailed explanation read in the appropriate section in chapter Object Reference.

```
'-----------------------------------------------------------
' Step2: Preparing the measurement
'-----------------------------------------------------------

objOpMode.OperatingMode = 3  ' Dynamic mode
objOpMode.Cantilever = 1     ' NCLR
objOpMode.VibratingAmpl = 0.1 'V
objOpMode.AutoVibratingFreq = True
objZCtrl.SetPoint = 50 '%
objZCtrl.PGain = 10000
objZCtrl.IGain = 1500
```

That's for now. Save your work again. Run it.

Still no action is done but you should see in the Operating Mode Panel and the Z-Controller Panel that the mode and the settings have been changed to the values we set in the script. You see the script acts here like a user would do. The script could also read the propertied values and get the result of direct user input.

You are now ready for approach. Go to Step 3.

5.4 Approaching the surface

Step 3 Approaching the surface

We write now the code for approaching automatically to the surface and check if everything went well after it.

The class Approach is now our focus. The script is not moving fast to the surface as a user would do in a first step because the script cannot interpret the video output and does not know therefore when to stop close to the surface.

First we stop the automatically start of imaging after approach, this is nice for a user but not for the script. Then we start the approach and wait until its finished.

```
'-----------------------------------------------------------
' Step3: Approaching the surface
'-----------------------------------------------------------

objAppr.AutoStartImaging = False
objAppr.StartApproach
Do While objAppr.IsMoving : Loop
```

No we have either approached to the surface or a error has occurred. We check this with the method Status and proceed if everything is ok. If not we withdraw from the surface and open a Dialog to display an error message.

```
If objAppr.Status = 3 Then

  ' insert script code of Step 4 to 7 here

Else ' approach error handling
  objAppr.StartWithdraw
  MsgBox "Approach error " & objAppr.Status & " occurred. Withdraw and exit."
  Do While objAppr.IsMoving : Loop
End If
```

That's for now. Save your work. To run it we have to be careful now because we move the scan head to the sample if we use the real microscope! Prepare the sample put it under the microscope and manually coarse approach the it. Now run the script. If you see in the Video camera that anything is going wrong and the tip is crashing into the surface click manually on "Retract".

We did our first real action. What is necessary is always to wait until the action is done if a method's name is **Start...** to synchronize the script to the microscope. If you can do something useful during the action. Just enter the script code in the Do While ... Loop!

Next we program the image script code. Go to Step 4.

5.5 Scan a surface

Step 4 Scan a surface

After the approach was successful we can prepare imaging and start the imaging process. The class Scan doing all this for us.

First we set the imaging size and other properties to our desire. Insert the following code in the If ... End If section of Step 3.

```
'--------------------------------------------------------
' Step4: Scan a Surface
'--------------------------------------------------------

Dim size : size = 50e-6 'm
objScan.ImageSize size,size
objScan.Scantime = 0.7 's
objScan.Points = 256
objScan.Lines  = 256
```

The code above show how to use a variable to store constants and use it to deliver arguments to a method.

No we start a single scan frame and wait until it's finished. During the wait we do some fun. We print the current scan line in the status bar:

```
Dim curline
objScan.StartFrameUp
Do While objScan.IsScanning :
  curline = objScan.Currentline
  objApp.PrintStatusMsg "Current line = " & curline & ". Remaining lines = " &
(objScan.Lines - curline)
  objApp.Sleep 1.0 's
Loop
```

As mentioned in the previous step we can do some useful things in the while loop and do not have just to wait! The code above shows how you can enhance the application and add features by your self not provided by the software.

That's for now. Save your work. To run it you should first withdraw if not already done and start then the script. When everything went ok we should be able to watch the script approaching and measure an image. Look to the bottom left side of the status bars during scanning.

If you would like to speed up the example image change number of lines or scan speed.

Next we withdraw from surface. Go to Step 5.

5.6 Withdraw tip from surface

Step 5 Withdraw from surface

To finish a measurement the tip should be retracted to a save position so that a user can safely remove the sample without destroying the cantilever. Let's develop this code.

First we move carefully a small amount from the surface. Method StartWithdraw and a wait loop is doing this.

```
'-----------------------------------------------------------
' Step5: Withdraw from surface
'-----------------------------------------------------------

objAppr.WithdrawSteps = 300
objAppr.StartWithdraw
Do While objAppr.IsMoving : Loop
```

Then we move away from surface to some larger distance. This is done by a fast Retract which we stop after 3 seconds.

```
  objAppr.StartRetract
  objApp.Sleep 3.0 's
  objAppr.Stop
```

Save your work. Now you have a fully automated imaging script in hand.

But we will add some more features to it. Let's do some image analysis. Go to Step 6.

5.7 Simple image data analysis

Step 6 Image data analysis

As a post measuring image analysis we implement an algorithm which is detecting the minimal and maximal z value measured.

The result is displayed in a message box dialog.

To do this we need to read in all image values and remember the lowest and highest value we find. This is don in a two nested loops over all scan lines and all data points per scan line. The function GetLine is providing us with the data values as a string. We convert this into a VBScript array and process the values.

```
'-----------------------------------------------------------
' Step6: Image analysis. Find min and max value
'-----------------------------------------------------------

Dim scanstring
Dim scanarray
Dim scanline
Dim point
Dim datavalue
Dim min : min = +1.0 ' start value
Dim max : max = -1.0
```

```
' loop through all scan lines and get the values
For scanline = 0 To objScan.Lines-1
  scanstring = objScan.GetLine(0,1,scanline,0,1) ' Z-Topography channel, Filter
RAW, Physical units
  objApp.PrintStatusMsg "Processing line " & scanline

  ' search all data points in a scan line
  scanarray = Split(scanstring,",")
  For Each point In scanarray
    datavalue = CDbl(point)
    ' check range
    If datavalue < min Then
      min = datavalue
    End If
    If datavalue > max Then
      max = datavalue
    End If
  Next
Next

MsgBox "Min value is " & FormatNumber(min*1e6,3) & "um. Max value is " &
FormatNumber(max*1e6,3) & "um"
```

Save your work. To test the calculation of this section create a new script just with the algorithm. First enter the code of step 1 and then insert at the comment just this code of step 6. Now run the new script. It is using the last measured image for it analysis.

Next we want do save the measured image. Go to Step 7.

5.8 Document handling

Step 7 Document handling

A good measurement is worth to be stored to disk. Therefore we create a new image document window with the contents of the Imaging Window and save the document to disk. We will ask the user about the filename in a input dialog.

```
'-----------------------------------------------------------
' Step7: Document handling. Save the scanned image to disk
'-----------------------------------------------------------

objScan.StartCapture
Dim objDoc : Set objDoc = objApp.DocGetActive()
Dim filename : filename = InputBox("Please enter a filename:")
If filename <> "" Then
  objDoc.Save(filename)
End If
```

We are at the end of the tutorial. Please run the full script once through and think about what's going on during the automated process is running.

Hopefully you enjoyed writing this little example and got the kick to write your own script.

Remember you can create also object from other programs like Word or Excel and control them too! What's about storing the result of a image or spectroscopy directly in an Excel sheet ?

6 Script examples

In this chapter we provide additional example scripts to give you more ideas what you could do with the scripting technology.

You find the source of this scripts

at C:\Program files\Nanosurf\Nanosurf\Scripts

or C:\Program files\Nanosurf\Nanosurf\Scripts\Examples

Table of example scripts:

Script name	Description
Imaging Adjust XY-Slope	Adjust the property X/Y-Slopes automatically
Create Height Histogram	Create a new document with a height histogram chart
Erase glitch from line	Removes measurement errors in the current line
Export data to CSV with Header	Saves data points to a file in a custom defined format
Timed Imaging	Measure multiple images with a delay between the scans. Auto saving and filename generation is included.
Lithography	Scratch a shape onto a soft surface by moving the tip with high force over the sample.

6.1 Imaging Adjust XY-Slope

This example demonstrates how calculate and correct the XY-Slopes during scan automatically.

Traditional slope compensation is a time consuming process and needs many steps to perform until the slopes are compensated

This script is performing all necessary steps involved to do this task. It executes the following:

Step 1 Start a 0° Rotated Image Frame
Step 2 Read the last scan line and calculates the slope by Linear Regression algorithm
Step 3 Start a 90° rotated image frame

Step 4 Read the last scan line and calculates the slope by Linear Regression algorithm
Step 5 store the calculated slope values to X and Y-Slope property of the Scan object

Source

```
'--------------------------------------------------
' Script: Imaging Adjust XY-Slope
'--------------------------------------------------
' Calculates the 0 and 90 degree slope and
' adjusts both SlopeX and Y Parameter.
'
' This script is useful during imaging.
' It automates the slope correction process which
' would be a manual task.
'
'--------------------------------------------------
' v1.2   5.8.2005, D.Braendlin, Nanosurf AG
'--------------------------------------------------

Option Explicit

Dim objApp  : Set objApp = SPM.Application
Dim objScan : Set objScan = objApp.Scan
Call Main()
Set objScan = Nothing
Set objApp = Nothing

'--------------------------------------------------
Sub Main()
'--------------------------------------------------
  Dim rot : rot = objScan.Rotation
  Dim ok : ok = vbFalse

  If Not objApp.IsObj(objScan) Then
    MsgBox "Error: Imaging window not active.",vbOKOnly,"Adjust XY-Slopes Script"
    Exit Sub
  End If


  ' adjust x axis
  objScan.Rotation = 0
  objScan.StartFrameUp
  ok = AdjustFastSlope()
  If ok Then

    ' adjust y axis
    objScan.Rotation = 90
    objScan.StartFrameUp
    ok = AdjustFastSlope()
  End  If

  If Not ok Then
    MsgBox "Error: Rotation outside bounds.",vbOKOnly,"Adjust XYSlopes Script"
  End If

  objScan.Rotation = rot
End Sub

'--------------------------------------------------
Function AdjustFastSlope()
```

```
'-------------------------------------------------
  AdjustFastSlope = vbTrue

  If objScan.GetFrameDir() <> 0 Then
      Do While (objScan.Currentline < 0) And objScan.IsScanning : Loop
  End If

  Dim RefLine : RefLine = objScan.Currentline
  If RefLine < 0 Then
    AdjustFastSlope = vbFalse
    Exit Function
  End If

  Dim FastSlope : FastSlope = CalcImageingSlope(RefLine)

  Dim maxdeviation : maxdeviation = 10 'degree
  If abs(objScan.Rotation) < maxdeviation Then
    objScan.SlopeX = objScan.SlopeX - FastSlope
  ElseIf abs(objScan.Rotation - 90) < maxdeviation Then
    objScan.SlopeY = objScan.SlopeY - FastSlope
  ElseIf abs(objScan.Rotation - 180) < maxdeviation Then
    objScan.SlopeX = objScan.SlopeX + FastSlope
  ElseIf abs(objScan.Rotation + 90) < maxdeviation Then
    objScan.SlopeY = objScan.SlopeY + FastSlope
  Else
    AdjustFastSlope = vbFalse
    Exit Function
  End If

End Function

'-------------------------------------------------
Function CalcImageingSlope(scanline_In)
'-------------------------------------------------
  Dim slope : slope = 0.0
  Dim i     : i = 0

  Dim dataline : dataline = objScan.GetLine(0,1,scanline_In,0,1)
  Dim zarray   : zarray = split(dataline,",")

  Dim xstep  : xstep  = objScan.ImageWidth / (objScan.Points -1)
  Dim xarray : ReDim xarray(UBound(zarray))
  xarray(0) = 0.0
  For i=1 To (UBound(xarray))
    xarray(i) = xstep*i
  Next

  Dim lin_coeff
  Dim ok : ok = CalcLinearRegress(xarray,zarray,lin_coeff)
  If ok Then
    slope = lin_coeff(1) * 180.0 / 3.14159265
  End If

  CalcImageingSlope = slope
End Function

'-------------------------------------------------
Function CalcLinearRegress(posarray_In,valarray_In, coeffarray_out)
'-------------------------------------------------
```

```
  Dim points : points = UBound(posarray_In)
  Dim vals   : vals   = UBound(valarray_In)
  Dim i : i = 0
  Dim m : m = 0
  Dim q : q = 0
  CalcLinearRegress = vbFalse

  ' input check: array need to have same length
  If points <> vals Then
    Exit Function
  End If

  ' calc intermediat results
  Dim s_x  : s_x = 0
  For i=0 To points
    s_x = s_x + posarray_In(i)
  Next

  Dim s_x2 : s_x2 = 0
  For i=0 To points
    s_x2 = s_x2 + posarray_In(i)*posarray_In(i)
  Next

  Dim s_y  : s_y = 0
  For i=0 To points
    s_y = s_y + valarray_In(i)
  Next

  Dim s_xy : s_xy = 0
  For i=0 To points
    s_xy = s_xy + posarray_In(i)*valarray_In(i)
  Next

  Dim delta : delta = CalcDetOf2x2Matrix(points+1,s_x,s_x,s_x2)

  ' if slope not indefinit (90°) then calc q and m
  If delta <> 0 Then
    ' y = q + m*x
    q = 1.0 / delta * CalcDetOf2x2Matrix(s_y,s_x,s_xy,s_x2)
    m = 1.0 / delta * CalcDetOf2x2Matrix(points+1,s_y,s_x,s_xy)

    ReDim coeffarray_out(2)
    coeffarray_out(0) = q
    coeffarray_out(1) = m
    CalcLinearRegress = vbTrue
  End If
End Function

'-----------------------------------------------------------
Function CalcDetOf2x2Matrix(a11,a12,a21,a22)
'-----------------------------------------------------------
  CalcDetOf2x2Matrix = a11*a22 - a12*a21
End Function
```

6.2 Create Height Histogram

This example demonstrates how to analyse a data container and create a new document with calculated data.

The script is calculating a height histogram of the data points of the selected data container and create a line chart with the result in a new document.

This script is performing all necessary steps involved to do this task. It executes the following:

Step 1 Check if a data container is selected
Step 2 Calculate value range
Step 3 Calculate height histogram
Step 4 Create a new document with a data container and a chart
Step 5 Saves the histogram result to new the data container

Source

```
'--------------------------------------------------
' Script: Histogram
'--------------------------------------------------
' Calculates a height histogram based of the active
' chart.
'--------------------------------------------------
' v1.1   1.8.2005, D.Braendlin, Nanosurf AG
'--------------------------------------------------

Option Explicit
Dim objApp : Set objApp = SPM.Application
Call Main()

'--------------------------------------------------
Sub Main()
'--------------------------------------------------

  ' get source data

  Dim objSrcDoc : Set objSrcDoc = objApp.DocGetActive()
  If Not objApp.IsObj(objSrcDoc) Then
    MsgBox "Error: No document loaded.",vbOKOnly,"Histogram Script"
    Exit Sub
  End If

  Dim objSrcData  : Set objSrcData  = objSrcDoc.DataGetActive()
  If Not objApp.IsObj(objSrcData) Then
    MsgBox "Please select a chart.",vbOKOnly,"Histogram Script"
    Exit Sub
  End If

  Call CreateHistogramDoc(objSrcData)

End Sub

'--------------------------------------------------
Sub CreateHistogramDoc(objSData)
'--------------------------------------------------
```

```
  ' get data value range ------

  objApp.PrintStatusMsg "Calculating range ..."

  Dim maxval,minval
  CalcMinMax objSData,0,1, minval, maxval

  ' prepare histogram container ------

  Dim objDestDoc  : Set objDestDoc  = objApp.DocCreate("",Nothing)
  Dim objDestData : Set objDestData = objDestDoc.DataCreate(-1,-1,Nothing)

  objDestDoc.DataSetGroupName objDestData.GetGroup(),"Histogram"

  objDestData.Lines  = 1
  objDestData.Points = 256

  objDestData.AxisPointMin   =  minval
  objDestData.AxisPointRange = (maxval - minval)
  objDestData.AxisPointName  = "Height Distribution"
  objDestData.AxisPointUnit  = objSData.AxisSignalUnit

  objDestData.AxisSignalMin   = -32768
  objDestData.AxisSignalRange = 65535
  objDestData.AxisSignalName  = objSData.AxisSignalName
  objDestData.AxisSignalUnit  = ""

  objDestData.AxisLineMin   = 0
  objDestData.AxisLineRange = objDestData.Lines
  objDestData.AxisLineName  = ""
  objDestData.AxisLineUnit  = ""

  ' create histogram data ------------------------------------

  objApp.PrintStatusMsg "Calculating histogram ..."

  Dim h_max
  Dim histogram_vec : histogram_vec =
CalcHistogram(objSData,256,minval,maxval,h_max)
  Dim ok : ok = objDestData.SetLine(0,0,Join(histogram_vec,","))

  ' display histogram chart -------------------------

  Dim objDestChart : Set objDestChart = objDestDoc.ChartCreate(-1,Nothing)
  objDestChart.Type   = 0  ' line chart
  objDestChart.Filter = 0
  objDestChart.Group  = objDestData.GetGroup()
  objDestChart.Signal = objDestData.GetSignal()
  objDestChart.RangeSpan   = h_max
  objDestChart.RangeCenter = h_max / 2
End Sub

'---------------------------------------------------
Function CalcHistogram(objData, resolution, min_val, max_val, h_max_out)
'---------------------------------------------------
  Dim histogram() : ReDim histogram(resolution-1)
  Dim maxvalue    : maxvalue = 0
  Dim curlinestr, curlinearray, h, h_max
```

```
  Dim x,y

  h_max = 0
  If (min_val < max_val) Then
    For y = 0 To (objData.Lines-1)
      curlinestr   = objData.GetLine(y,0,1)
      curlinearray = Split(curlinestr,",")
      For x = 0 To (objData.Points-1)
        h = (CDbl(curlinearray(x))-min_val)/(max_val-min_val) * (resolution-1)
        If (h>=0) And (h<resolution) Then
          histogram(h) = histogram(h) + 1
          If histogram(h) > h_max Then
            h_max = histogram(h)
          End If
        End If
      Next
    Next
  End If
  h_max_out = h_max
  CalcHistogram = histogram
End Function


'-------------------------------------------------
Sub CalcMinMax(objData, filter, mode, min_out, max_out)
'-------------------------------------------------
  Dim maxval : maxval = -1.0e100
  Dim minval : minval = +1.0e100
  Dim curdata, curarray, curvalue
  Dim x,y

  For y = 0 To (objData.Lines-1)
    curdata   = objData.GetLine(y,filter,mode)
    curarray = Split(curdata,",")
    For x = 0 To (objData.Points-1)
      curvalue = CDbl(curarray(x))
      If maxval < curvalue Then
        maxval = curvalue
      End If
      If minval > curvalue Then
        minval = curvalue
      End If
    Next
  Next

  max_out = maxval
  min_out = minval
End Sub
```

6.3 Erase glitch from line

This example demonstrates in place data modification.

This script is modifying the measured data and removes measurement error like jumps in height or small glitches occurring only in one data line.

It calculates new values for the current selected data line by replacing the data points with the average of the points of its neighbor lines.

This script is performing all necessary steps involved to do this task. It executes the following:

Step 1 Check if a data container is selected
Step 2 Extract the two neighbor lines of the selected one
Step 3 Replace the selected line with the average of the two other lines

Source

```
'--------------------------------------------------
' Script: Erase glitch from line
'--------------------------------------------------
' Removes glitches from single data lines.
'
' The current line of the active chart is processed.
'
' The allgorithm uses the two neighbour lines as
' references and calculates new data values.
'--------------------------------------------------
' v1.1   9.8.2005, D.Braendlin, Nanosurf AG
'--------------------------------------------------

Option Explicit
Dim objApp : Set objApp = SPM.Application
Call Main()
Set objApp = Nothing

'--------------------------------------------------
Sub Main()
'--------------------------------------------------

  ' get source data

  Dim objSrcDoc : Set objSrcDoc = objApp.DocGetActive()
  If Not objApp.IsObj(objSrcDoc) Then
    MsgBox "Sorry, no document selected.",vbOKOnly,"Erase glitch"
    Exit Sub
  End If

  Dim objSrcData  : Set objSrcData  = objSrcDoc.DataGetActive()
  If Not objApp.IsObj(objSrcData) Then
    MsgBox "Please select a chart.",vbOKOnly,"Erase glitch"
    Exit Sub
  End If

  Dim ok : ok = RemoveSpikes(objSrcData,objSrcData.Currentline)

  If Not ok Then
    MsgBox "Sorry, this data cannot be processed." & vbCRLF & "Not enough
lines.",vbOKOnly,"Erase glitch"
  End If
End Sub


'-----------------------------------------------------------------------------------
----------
Function RemoveSpikes(objData,Line)
```

```
'---------------------------------------------------------------------------------------
    RemoveSpikes = vbFalse

    If Not objApp.IsObj(objData) Then
      Exit Function
    End If

    If (Line >= objData.Lines) Or (Line < 0) Or (objData.Lines < 2) Then
      Exit Function
    End If

    ' get first referenc line
    Dim line1data
    If Line < (objData.Lines-1) Then
      line1data = objData.GetLine(Line+1,0,0)
    Else
      line1data = objData.GetLine(Line-1,0,0)
    End If
    Dim line1array : line1array = Split(line1data,",")

    ' get second referenc line
    Dim line2data
    If Line > 0 Then
      line2data = objData.GetLine(Line-1,0,0)
    Else
      line2data = objData.GetLine(Line+1,0,0)
    End If
    Dim line2array : line2array = Split(line2data,",")

    ' get line of interest
    Dim curdata  : curdata = objData.GetLine(Line,0,0)
    Dim curarray : curarray = Split(curdata,",")

    ' remove spikes
    Dim x
    For x = 0 To UBound(curarray)
      curarray(x) = (CInt(line1array(x)) + CInt(line2array(x))) / 2
    Next

    curdata = Join(curarray,",")
    objData.SetLine Line,0,curdata

    RemoveSpikes = vbTrue
  End Function
```

6.4 Export data to CSV with Header

This example demonstrates how to program an export function which saves measured data to a file.

The internal export function of the application is enough for most of the data export requirements. But some times a user want to export data in a customized way. This script demonstrates how to do this.

This script is performing all necessary steps involved to do this task. It executes the following:

Step 1 Check if a data container is selected
Step 2 Ask for a target filename
Step 3 Read all data from the container and saves them to file

Source

```
'--------------------------------------------------
' Script: Export data to CVS with Header
'--------------------------------------------------
' Saves current activated data to a file.
' The data is saved as a comma separated value list
' with a header
'--------------------------------------------------
' v1.1   1.8.2005, Pieter van Schendel, Nanosurf AG
'--------------------------------------------------

Option Explicit
Dim objApp : Set objApp = SPM.Application
Call Main()
Set objApp = Nothing

'--------------------------------------------------
Sub Main()
'--------------------------------------------------

 ' get source data --------

  Dim objSrcDoc : Set objSrcDoc = objApp.DocGetActive()
  If Not objApp.IsObj(objSrcDoc) Then
    MsgBox "Error: No document loaded.",vbOKOnly,"Export Script"
    Exit Sub
  End If

  Dim objSrcData  : Set objSrcData  = objSrcDoc.DataGetActive()
  If Not objApp.IsObj(objSrcData) Then
    MsgBox "Please select a chart.",vbOKOnly,"Export Script"
    Exit Sub
  End If

  ' Ask for file ------
  Dim comdlg : Set comdlg = CreateObject("MSComDlg.CommonDialog")
  comdlg.DialogTitle = "Export the data as:"
  comdlg.filter ="CSV file with header|*.csv"
  comdlg.MaxFileSize = 260
  comdlg.CancelError = False
  comdlg.ShowSave

  ' save to disk ------
  Dim targetfile : targetfile = comdlg.filename
  If targetfile <> "" Then
    ExpartDataToFile targetfile,objSrcData
  End If

End Sub


'--------------------------------------------------
```

```
Sub ExpartDataToFile(filename,objdata)
'---------------------------------------------------

  ' Alloc objects ----
  Dim objFS  : Set objFS = CreateObject("Scripting.FileSystemObject")
  Dim objFile: Set objFile= objFS.CreateTextFile(filename)

  ' write header  -----
  objFile.WriteLine "#Points: " & objdata.Points
  objFile.WriteLine "#Lines : " & objdata.Lines
  objFile.WriteLine "#Width : " & objdata.AxisPointRange
  objFile.WriteLine "#Height: " & objdata.AxisLineRange

  ' write data  -------
  Dim linedata
  Dim curline
  Dim lines : lines = objdata.Lines
  For curline = 0 To lines-1
    linedata = objdata.GetLine(curline,0,1) ' RAW data, physical units
    objFile.WriteLine linedata
  Next

  objFile.Close

  ' clean up objects ----
  Set objFile = Nothing
  Set objFS   = Nothing
End Sub
```

6.5 Timer controlled imaging

This example demonstrates how to add a function to measure multiple images autonomous.

If one want to study a surface sample over time to see drift or change in features a possibility to do a series of measurements is needed.

To measure this series could be very time consuming and should be done automatically.

This script is doing exactly this. Measure a image, save it to disc , wait some time, and do it again multiple time. It asks the user the amount of measurement to take, the delay time between two measurements and a filename mask to know how to name the images.

The file mask is the path and the start of the resulting files. The script add to this mask a counting number and the file extension (e.g A file mask of "D:\MyData\MyImages" creates the images in the directory D:\MyData with names like MyImages1.nid, MyImages2.nid, and so on).

Source

```
'-----------------------------------------------------------
' Prog: Timed Imaging - measure a set of images with delay and save the result to
disc
'-----------------------------------------------------------
' Version 1.0 Nanosurf
'-----------------------------------------------------------
```

```
Option Explicit

' startup application and get all needed objects
Dim objApp : Set objApp = SPM.Application
Dim objScan : Set objScan   = objApp.Scan

objScan.Stop

'-----------------------------------------------------------
'  Preparing the measurement
'-----------------------------------------------------------

Dim dTotalImages : dTotalImages = 1
Dim dImageDelay  : dImageDelay  = 60.0
Dim strFilemask  : strFilemask = "c:\Timed Image"


'-----------------------------------------------------------
'  Ask user for details
'-----------------------------------------------------------

Dim retval
retval  = InputBox("Please enter the number of images to take","Script
request",dTotalImages)
If retval >= 1 Then
  dTotalImages = retval

  retval  = InputBox("Please enter the delay time between to images in [s]","Script
request",dImageDelay)
  If retval >= 1 Then
    dImageDelay = retval

    strFilemask  = InputBox("Enter filename mask of the images. 'Cancel' if not
desired.","Script request",strFilemask)

    '-----------------------------------------------------------
    '  Measure the images
    '-----------------------------------------------------------
    Dim dCurImage:
    For dCurImage = 1 To CInt(dTotalImages)

      objApp.PrintStatusMsg "Measuring image " & FormatNumber(dCurImage,0) & " of "
& FormatNumber(dTotalImages,0)
      objScan.StartFrameUp
      Do While objScan.IsScanning : Loop

      objScan.StartCapture
      If strFilemask <> "" Then
        objApp.SaveDocument strFilemask & FormatNumber(dCurImage,0) & ".nid"
      End If

      If CInt(dCurImage) < CInt(dTotalImages) Then
        objApp.PrintStatusMsg "Waiting for " & FormatNumber(dImageDelay,0) & "s
until image " & FormatNumber(dCurImage+1,0) & " of " & FormatNumber(dTotalImages,0)
& " is taken."
        objApp.Sleep dImageDelay
      End If
```

```
    Next

    MsgBox "All images measured. End of script"

  Else
    MsgBox "Bad delay time. Script aborted."
  End If
Else
  MsgBox "Bad number of images. Script aborted."
End If

Set objScan   = Nothing
Set objApp    = Nothing
```

6.6 Lithography

The aim of this script example is to demonstrate the use of the lithography script commands.

This example will scratch a square shape into a sample surface.

It moves first with low set point force to the start point of the square shape, increases the set point and moves four times to scratch the square shape into the surface. After this is completed, it decreases the set point again to a standard not modifying value.

Before you run the script mount your sample and approach to it. Take also an image of the surface before you scratch the shape.

For more general information about lithography please refer to the "Operating Instructions" manual.

Source

```
'---------------------------------------------------
' Script: Simple lithography (Lithomodule)
'---------------------------------------------------
' This script creates a square shape with
' an edge length of 20.0 micrometer.
'
' The AFM static deflection mode is used
' to scratch the shape.
'
'---------------------------------------------------
' v1.0   12.01.2009, Adrian Gersbach, Nanosurf AG
'---------------------------------------------------

Option Explicit

' startup application and get all needed objects
Dim objApp : Set objApp = SPM.Application
Dim objLitho : Set objLitho = objApp.Litho
Dim objScan : Set objScan = objApp.Scan

Call Main()

' clean up
Set objScan = Nothing
```

```
Set objLitho = Nothing
Set objApp = Nothing

'--------------------------------------------------
Sub Main()
'--------------------------------------------------
  ' init variables
  Dim fTipSpeedUp : fTipSpeedUp = 8.0e-6
  Dim fTipSpeedDown : fTipSpeedDown = 4.0e-6

  Dim nXOffset : nXOffset = objScan.CenterPosX
  Dim nYOffset : nYOffset = objScan.CenterPosY
  Dim nZOffset : nZOffset = 0.0

  ' clean up command list
  objLitho.ClearCmdList

  ' add commands
  objLitho.AddCmd_PenUp

  ' set opmode (AFM static deflection mode)
  objLitho.OperatingMode = 2

  ' set tipvoltage to 0.0 V
  objLitho.AddCmd_TipVoltage 0.0

  ' set setpoint to 15.0uN
  objLitho.AddCmd_SetPoint 15.0e-6

  objLitho.AddCmd_TipSpeed fTipSpeedUp

  ' move tip to start position
  objLitho.AddCmd_MoveTip 10.0e-6 + nXOffset, 10.0e-6 + nYOffset, 0.0 + nZOffset

  ' lower tip to start litho
  objLitho.AddCmd_PenDown

  objLitho.AddCmd_TipSpeed fTipSpeedDown

  ' create a square shape
  objLitho.AddCmd_MoveTip +10.0e-6 + nXOffset, -10.0e-6 + nYOffset, 0.0 + nZOffset
  objLitho.AddCmd_MoveTip -10.0e-6 + nXOffset, -10.0e-6 + nYOffset, 0.0 + nZOffset
  objLitho.AddCmd_MoveTip -10.0e-6 + nXOffset, +10.0e-6 + nYOffset, 0.0 + nZOffset
  objLitho.AddCmd_MoveTip +10.0e-6 + nXOffset, +10.0e-6 + nYOffset, 0.0 + nZOffset

  ' retract tip
  objLitho.AddCmd_PenUp

  objLitho.AddCmd_TipSpeed fTipSpeedUp

  ' move tip to center position
  objLitho.AddCmd_MoveTip 0.0 + nXOffset, 0.0 + nYOffset, 0.0 + nZOffset

  ' start lithography session
  objLitho.Start

  ' wait untill litho session is finished
  Do While objLitho.IsWorking : Loop
```

```
End Sub
```

7 Object Reference

This chapter describes in detail all the COM Interface objects of the Nanosurf program.

The complete functionality of the COM Interface is sorted in a hierarchical object structure. Each sub object consists of a set of properties and methods for a special task.

The entry point of the class hierarchy is the COM class Nanosurf_C3000.Application for external calls and `SPM.Application` for internal calls.

It is providing general application specific properties and methods and it is the root to all other objects of the Nanosurf program.

An overview about the defined classes is shown in the following table:

Main Class	Function
Application	Global application specific functions
Online Objects	
System	Provides general online relevant system access functions
Approach	Controls the approach process
Litho	Provides lithography functions.
Scan	Controls the imaging process
Spec	Provides spectroscopy functions
ZController	Z controller feedback loop settings
OperatingMode	Sensor operating mode and mode depending settings
Video	Video observation camera settings
ScanHead	Provides scan head functions
SignalIO	Provides IO functions
CantileverList	Provides access function to the cantilever database
SPMCtrlManager	Provide access to advanced function of the C3000 controller
Stage	Provides access to the stage backend
BatchManager	Provides access to the batch manager backend
Data Processing Objects	
Document	Represents a document with charts of measured data (as stored in nid-Files)

Chart	Controls the visual representation of data values
Data	Represents a block of data for a signal
Info	Represents a set of measurement header information

7.1 Application

The Application class is providing general application specific properties and methods.

It is also the root for online classes which are provided as a property with the same name as the class name.

Access to stored data are given by references to Document class objects by another set of methods.

Retrieving a object pointer to the single instance of the Application class depends on the origin of the caller:

- From a script inside the Nanosurf program (e.g A script written in the Script Editor) there is the named item SPM with the property Application. A call to `SPM.Application` returns an object pointer to the single instance of this class.
- From a external script (e.g WScript.exe) the script need to call `CreateObject("Nanosurf_C3000.Application")`. This will return a object pointer to the single instance of this class.

Table of properties of Application class:

Property name	Purpose
Approach	Returns a object pointer to the single Approach class object
BatchManager	Returns a object pointer to the single BatchManager class object
Litho	Returns a object pointer to the single Litho class object
Scan	Returns a object pointer to the single Scan class object
ScanHead	Returns a object pointer to the single ScanHead class object
SignalIO	Returns a object pointer to the single SignalIO class object
Spec	Returns a object pointer to the single Spec class object
Stage	Returns a object pointer to the single Stage class object
OperatingMode	Returns a object pointer to the single Operating class object
ZController	Returns a object pointer to the single ZController class object
Video	Returns a object pointer to the single Video class object
System	Returns a object pointer to the single System class object
SPMCtrlManager	Returns a object pointer to the single SPMCtrlManager class object

CantileverList	Returns a object pointer to the single System class object
Version	Returns the applications versions string
AutoExit	Close the application after end of script
Simulation	Enable Simulation of Microscope
StatusReadDelay	Sets the delay time used by all Status Properties read
Visible	Show or hide the application
GalleryHistoryAutoIndexing	Toggle auto-indexing when saving measurement files

Table of methods of Application class:

Methode name	Purpose
DocCount	Return the number of open documents
DocCreate	Create a new document object
DocDeleteAll	Delete all open documents
DocDeleteByName	Delete document with given name
DocDeleteByPos	Delete document at given position
DocGetActive	Return document object to current document
DocGetByName	Return document object with given name
DocGetByPos	Return document object at given position
GetGalleryHistoryDirectoryPath	Returns the actual path where the history files are stored
SetGalleryHistoryDirectoryPath	Defines the actual path where the history files are stored
GetGalleryHistoryFilenameMask	Returns the current history filename mask
SetGalleryHistoryFilenameMask	Defines the history filename mask
GetGalleryHistoryFilenameIndex	Returns the history filename index
SetGalleryHistoryFilenameIndex	Defines the history filename index
GetScriptDirectoryPath	Returns the actual path where the script files are stored
SetScriptDirectoryPath	Defines the actual path where the script files are stored
IsObj	Checks if a given object variable is valid or not
IsStartingUp	Monitors the application initialization process
LoadCalibration	Load a new scan head calibration from a hed-file
LoadChartArrangement	Load a set of charts from file
LoadDocument	Load a image document from file
LoadParameter	Load a set of parameter from file

LoadWorkspace	Load a new workspace configuration from file
Log	Simple logging of some message string
LogEx	Log a message with specific channel and log level
LogUserMarker	Generate a user marker into the log system
PrintStatusMsg	Print a message in the status bar
SaveCalibration	Save current scan head calibration to file
SaveChartArrangement	Saves current set of charts to file
SaveDocument	Save current selected image document to file
SaveParameter	Saves current set of parameter to file
SaveWorkspace	Save current workspace configuration to file
Sleep	Wait some seconds

7.1.1 Properties

7.1.1.1 Application::Approach

Returns a dispatch pointer to the sub class Approach. This property is read only.

Syntax

application.**Approach** [read only]

Result

The **Approach** property is returning a pointer to the IDispatch interface of the Approach object.

Remarks

Only one single instance exists of Approach object. All successive read of this property will return the same IDispatch pointer. A read of this property will also open the "Position Window" in the user interface.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp  : Set objApp  = Nanosurf_C3000.Application
Dim objAppr : Set objAppr = objApp.Approach

' do something with the object
```

```
' clean up

objAppr = nul : Set objAppr = Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

Class Approach

7.1.1.2 Application::AutoExit

Returns or set the action at script termination.

Syntax

application.**AutoExit** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if the application should close after last reference to Application object is released otherwise to `False`

Remarks

The AutoExit property is used when the script want to control the Nanosurf program fully automatically and handle the startup and closing by itself. Set this property to True anytime after startup is finished.

If AutoExit is set the application is closed after releasing the last reference to the application object.

Example

```
' open application

Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
Do While objApp.IsStartingUp : Loop

' do something
....

' close application program

objApp.AutoExit = True
objApp = nul : Set objApp = Nothing
```

See also

Method IsStartingUp

7.1.1.3 Application::BatchManager

Returns a dispatch pointer to the sub class BatchManager. This property is read only.

Syntax

application.**BatchManager** [read only]

Result

The **BatchManager** property is returning a pointer to the IDispatch interface of the BatchManager object.

Remarks

Only one single instance exists of the BatchManager object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp          : Set objApp          = Nanosurf_C3000.Application
Dim objBatchManager : Set objBatchManager = objApp.BatchManager

' do something with the object

' clean up

objBatchManager = nul : Set objBatchManager = Nothing
objApp          = nul : Set objApp          = Nothing
```

See also

Class BatchManager

7.1.1.5 Application::Litho

Returns a dispatch pointer to the sub class Litho. This property is read only.

Syntax

application.**Litho** [read only]

Result

The **Litho** property is returning a pointer to the IDispatch interface of the Litho object.

Remarks

Only one single instance exists of Litho object. All successive read of this property will return the same IDispatch pointer. A read of this property will also open the "Lithography Window" in the user interface.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp   : Set objApp   = Nanosurf_C3000.Application
Dim objLitho : Set objLitho = objApp.Litho

' do something with the object

' clean up

objLitho = nul : Set objLitho = Nothing
objApp   = nul : Set objApp   = Nothing
```

See also

Class Litho

7.1.1.6 Application::OperatingMode

Returns a dispatch pointer to the sub class OperatingMode. This property is read only.

Syntax

application.**OperatingMode** [read only]

Result

The **Operating** property is returning a pointer to the IDispatch interface of the OperatingMode object.

Remarks

Only one single instance exists of OperatingMode object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp    : Set objApp    = Nanosurf_C3000.Application
Dim objOpMode : Set objOpMode = objApp.OperatingMode

' do something with the object

' clean up

objOpMode = nul : Set objOpMode = Nothing
objApp    = nul : Set objApp    = Nothing
```

See also

Class OperatingMode

7.1.1.7 Application::Scan

Returns a dispatch pointer to the sub class Scan. This property is read only.

Syntax

application.**Scan** [read only]

Result

The **Scan** property is returning a pointer to the IDispatch interface of the Scan object.

Remarks

Only one single instance exists of Scan object. All successive read of this property will return the same IDispatch pointer. A read of this property will also open the "Imaging Window" in the user interface.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp  : Set objApp  = Nanosurf_C3000.Application
Dim objScan : Set objScan = objApp.Scan

' do something with the object

' clean up

objScan = nul : Set objScan = Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

Class Scan

7.1.1.8 Application::ScanHead

Returns a dispatch pointer to the sub class ScanHead. This property is read only.

Syntax

application.**ScanHead** [read only]

Result

The **ScanHead** property is returning a pointer to the IDispatch interface of the ScanHead object.

Remarks

Only one single instance exists of ScanHead object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp  : Set objApp  = Nanosurf_C3000.Application
Dim objScanHead : Set objScanHead = objApp.ScanHead

' do something with the object
```

```
' clean up

objScanHead = nul : Set objScanHead = Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

Class ScanHead

7.1.1.9 Application::ShowWindow

Defines the display style of the main window.

Syntax

application.**ShowWindow(**style**)**

Arguments

Argument	Type	Description
style	short	Visibility style number

Result

None.

Remarks

The **ShowWindow** method sets the visibility state of the window.

Available styles see Doc.ShowWindow Method

Example

```
objApp.ShowWindow(0) ' hide the imaging window
```

See also

Application::Visible

7.1.1.10 Application::SignalIO

Returns a dispatch pointer to the sub class SignalIO. This property is read only.

Syntax

application.**SignalIO** [read only]

Result

The **SignalIO** property is returning a pointer to the IDispatch interface of the SignalIO object.

Remarks

Only one single instance exists of SignalIO object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp   : Set objApp   = Nanosurf_C3000.Application
Dim objSignalIO : Set objSignalIO = objApp.SignalIO

' do something with the object

' clean up

objSignalIO = nul : Set objSignalIO = Nothing
objApp   = nul : Set objApp   = Nothing
```

See also

Class SignalIO

7.1.1.11 Application::Simulation

Returns or set the interface mode. In simulation mode the program is using an internal microscope simulation as target.

Syntax

application.**Simulation** [= flag]

Settings

Argument	Type	Description
flag	Boolean	Set to `True` if the application should simulate the microscope. Set to `False` to use the real microscope.

Remarks

The **Simulation** property is defining the interface to the microscope. If this property is set to `True` a program internal simulation of a microscope and a surface is used. Most of the functionality of the real scope is simulated.

Switching between simulation and real microscope can be performed any time. Each microscope is initialized at switching. Use property **IsStartingUp** to wait for the end of the switch.

A virtual surface can be imaged with the "Imaging Window" or the Scan object, with the "Spectroscopy Window" or the Spec object a Tip Potential modulation can be performed.

OperatingMode, ZController and Video object settings are not simulated and have no influence in the simulation.

Example

```
' open application
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
Do While objApp.IsStartingUp : Loop

objApp.Simulation = True
Do While objApp.IsStartingUp : Loop
```

See also

Class Scan, Spec, Property IsStartingUp

7.1.1.12 Application::Spec

Returns a dispatch pointer to the sub class Spec. This property is read only.

Syntax

application.**Spec** [read only]

Result

The **Spec** property is returning a pointer to the IDispatch interface of the Spec object.

Remarks

Only one single instance exists of Spec object. All successive read of this property will return the same IDispatch pointer. A read of this property will also open the "Specroscopy Window" in the user interface.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp  : Set objApp  = Nanosurf_C3000.Application
Dim objSpec : Set objSpec = objApp.Spec

' do something with the object

' clean up

objSpec = nul : Set objSpec = Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

Class Spec

7.1.1.13 Application::SPMCtrlManager

The SPM control manager handles access to the SPM subsystem.

A object pointer to this class is provided by the Application.SPMCtrlManager object property.

Table of properties for the SPMCtrlManager class:

Property name	Purpose
LogicalUnit	Returns a object pointer to the single LogicalUnit class object
DataBuffer	Returns a object pointer to the single DataBuffer class object
DataStream	Returns a object pointer to the single DataStream class object
MacroCmd	Returns a object pointer to the single MacroCmd class object

7.1.1.14 Application::Stage

Returns a dispatch pointer to the sub class Stage. This property is read only.

Syntax

application.**Stage** [read only]

Result

The **Stage** property is returning a pointer to the IDispatch interface of the Stage object.

Remarks

Only one single instance exists of the Stage object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp   : Set objApp   = Nanosurf_C3000.Application
Dim objStage : Set objStage = objApp.Stage

' do something with the object

' clean up

objStage = nul : Set objStage = Nothing
objApp   = nul : Set objApp   = Nothing
```

See also

Class Stage

7.1.1.15 Application::StatusReadDelay

Returns or set time usd to delay a read request by all status properties.

Syntax

application.**StatusReadDelay** [= time]

Settings

Argument	Type	Description
time	float	Set or read the time used to delay each status request. Default value is 0.3s

Remarks

The **StatusReadDelay** property defines the time a status property waits until its read the new status and return its value to the caller function.

During this wait time the Nanosurf application still performs its operation and is not delayed.
The usage of this delay is to lower the CPU usage during a wait loop until a certain status is reached by the script program.

All 'obj.*Is...'* properties of the online classes are using these delay timer (e.g *objScan.IsScanning*, *objAppr.IsMoving*, ...) .

The default value of 0.3s can be overwritten by setting the registry key 'Nanosurf/ Application/ScriptingStatusReadDelay=0.3'

Example

```
' open application
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
objApp.StatusReadDelay = 0.0
Do While objApp.IsStartingUp : Loop
```

See also

All Is... properties of classes Approach, Scan, Spec, OperatingMode, ZController

7.1.1.16 Application::System

Enter topic text here.

7.1.1.18 Application::Video

Returns a dispatch pointer to the sub class Video. This property is read only.

Syntax

application.**Video** [read only]

Result

The **Video** property is returning a pointer to the IDispatch interface of the Video object.

Remarks

Only one single instance exists of Video object. All successive read of this property will return the same IDispatch pointer. A read of this property will also open the "Position Window" in the user interface.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp   : Set objApp   = Nanosurf_C3000.Application
Dim objVideo : Set objVideo = objApp.Video

' do something with the object

' clean up

objVideo = nul : Set objVideo = Nothing
objApp   = nul : Set objApp   = Nothing
```

See also

Class Video

7.1.1.19 Application::Visible

Returns or set the interface mode. In simulation mode the program is using an internal microscope simulation as target.

Syntax

application.**Visible** [= flag]

Settings

Argument	Type	Description
flag	Boolean	Set "True" to show the application
		Set "False" to hide the application

Remarks

If the application is started up using the COM interface it is hidden unless the user sets "Visible" to "True".

Example

```
' open application
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
objApp.Visible = true

objApp.Visible = false

objApp   = nul : Set objApp  = Nothing
```

See also

Application::ShowWindow

7.1.1.20 Application:GalleryHistoryAutoIndexing

Returns or set auto-indexing when creating filenames for NID files.

Syntax

application.**GalleryHistoryAutoIndexing** [= flag]

Settings

Argument	Type	Description
flag	Boolean	Set "True" to enable auto-indexing (default) Set "False" to disable auto-indexing

Remarks

If the filemask doesn't specify [INDEX] keyword, it is added when auto-indexing is enabled.
If auto-indexing is disabled, [INDEX] is not added if it missing.
To have effect, it must be called before Application::SetGalleryHistoryFileMask.

Example

```
' open application
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")

objApp.GalleryHistoryAutoIndexing = false
objApp.SetGalleryHistoryFilenameMask("MyUniqueImage")

objApp   = nul : Set objApp  = Nothing
```

See also

Application::SetGalleryHistoryFileMask

7.1.1.21 Application::ZController

Returns a dispatch pointer to the sub class ZController. This property is read only.

Syntax

application.**ZController** [read only]

Result

The **ZController** property is returning a pointer to the IDispatch interface of the ZController object.

Remarks

Only one single instance exists of ZController object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp  : Set objApp  = Nanosurf_C3000.Application
Dim objCtrl : Set objCtrl = objApp.ZController

' do something with the object

' clean up

objCtrl = nul : Set objCtrl = Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

Class ZController

7.1.2 Methods

7.1.2.1 Application::DocCount

Return the number of open documents

Syntax

val = *app*.**DocCount()**

Arguments

none.

Result

Result	Type	Description
val	short	Returns the number of open document.

Remarks

The **DocCount** method counts the number of open document windows.

Example

```
docs = objApp.DocCount()
```

See also

DocGetByPos Method.

7.1.2.2 Application::DocCreate

Returns a new document class object.

Syntax

objDoc = *app*.**DocCreate(**filename,srcobj**)**

Arguments

Argument	Type	Description
filename	string	the document is loaded from disk or not if argument is `""`
srcobj	object	the contents of the source document is copied if *srcobj* is not `Nothing`

Result

Result	Type	Description
objDoc	Object	Returns a IDispatch object for the document at position *pos* or an invalid object

Remarks

The **DocCreate** method returns a IDispatch object to a newly created document. The new document is completely empty with no data objects, no info sections and no charts. If the new document is a valid object can be checked by **objApp.IsObj()**.

If the argument *filename* is not empty the contents of this NID-File is loaded into the document.
If the argument *srcobj* is a valid document object its contents it copied into the new document.
If both argument are defined the NID-File is loaded.

Example

```
' create a new empty document
Set objDoc = objApp.DocCreate("",Nothing)

' create a new document and load data from file
Set objDoc = objApp.DocCreate("MyDocument.nid",Nothing)
If Not objApp.IsObj(objDoc) Then
  MsgBox "File not found"
End If

' Copy current active document
Set objSrcDoc = objApp.DocGetActive()
Set objDoc = objApp.DocCreate("",objSrcDoc)
```

See also

Class Document, DocGetActive Method, IsObj Method

7.1.2.3 Application::DocDeleteAll

Close all open documents

Syntax

done = *app*.**DocDeleteAll()**

Arguments

None.

Result

Result	Type	Description
done	Boolean	Returns `True` if all document could be closed otherwise `False`

Remarks

The **DocDeleteAll** method closes all open documents.

Example

```
' close all documents
ok = objApp.DocDeleteAll()
If objApp.DocCount() > 0 Then
  MsgBox "Error: Could not close all documents"
End If
```

See also

Class Document, DocCount Method

7.1.2.4 Application::DocDeleteByName

Deletes the document with a specified filename

Syntax

done = *app*.**DocDeleteByName(**filename**)**

Arguments

Argument	Type	Description
filename	string	Close the document with this name

Result

Result	Type	Description
done	Boolean	Returns `True` if the document could be closed otherwise `False`

Remarks

The **DocDeleteByName** method closes the document with the name *filename*.
The argument has to be a path string. If no document is found this method return `False`.

Example

```
' close active document
Set oDoc = objApp.DocGetActive()
If objApp.IsObj(oDoc) Then
  objApp.DocDeleteByName(oDoc.Name)
End If
```

See also

Class Document, DocGetActive Method, IsObj Method

7.1.2.5 Application::DocDeleteByPos

Deletes the n'th document

Syntax

done = *app*.**DocDeleteByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Close the document at the specified position

Result

Result	Type	Description
done	Boolean	Returns `True` if the document could be closed otherwise `False`

Remarks

The **DocDeleteByPos** method closes the document at position *pos*.
The argument has to be positive and lower than the value return by **DocCount()**.

Example

```
' close last document
objApp.DocDeleteByPos(objApp.DocCount() - 1)
```

See also

Class Document, DocCount Method, IsObj Method

7.1.2.6 Application::DocGetActive

Returns a *Document* class object of the currently selected document

Syntax

objDoc = *app*.**DocGetActive()**

Arguments

none.

Result

Result	Type	Description
objDoc	Object	Returns a IDispatch object to the selected document or an invalid object if none is selected

Remarks

The **DocGetActive** method returns a IDispatch object to the currently active or selected document. The selected document has a highlighted title bar**.** If no document is loaded or active an invalid object is returned. This can be checked by **objApp.IsObj()**.

Example

```
Set objDoc = objApp.DocGetActive()
If Not objApp.IsObj(objDoc) Then
  MsgBox "Please select an document"
else
  MsgBox "Current document is " & objDoc.Name
End If
```

See also

Class Document

7.1.2.7 Application::DocGetByName

Returns a *Document* class object with the specified name.

Syntax

objDoc = *app*.**DocGetByName(**name**)**

Arguments

Argument	Type	Description
name	string	Name of document

Result

Result	Type	Description
objDoc	Object	Returns a IDispatch object to the document with the given name or an invalid object if no document is not found

Remarks

The **DocGetByName** method returns a IDispatch object to the document with the given name.
If no document with *name* is found a invalid object is returned. This can be checked by **objApp.IsObj()**.

The name of a document is its filename including the full path. The name of a document which is not loaded from file or was never stored is its temporary filename including the path to the backup directory.

Example

```
Set objDoc = objApp.DocGetByName("mydoc.nid")
If Not objApp.IsObj(objDoc) Then
  MsgBox "Document not loaded"
End If
```

See also

Class Document

7.1.2.8 Application::DocGetByPos

Returns a *Document* class object at the specified position.

Syntax

objDoc = *app*.**DocGetByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Documents position number.

Result

Result	Type	Description
objDoc	Object	Returns a IDispatch object for the document at position *pos* or an invalid object if *pos* >= **DocCount**()

Remarks

The **DocGetByPos** method returns a IDispatch object to the document at the position. If position is out of range an invalid object is returned. This can be checked by **objApp.IsObj()**.

The position is the index into an list which keeps track of all open documents and represents the nth document window as shown in the pull down menu "Window".

Example

```
opendocs = objApp.DocCount()
For i = 0 To opendocs-1
  Set objDoc = objApp.DocGetByPos(i)
  MsgBox "Filename = " & objDoc.Name
End For
```

See also

Class Document, DocCount Method**,** DocGetByName Method

7.1.2.9 Application::GetGalleryHistoryDirectoryPath

Returns the history file path to the directory where all *.nid-files will be stored, when captured.

Syntax

filePath = *app*.**GetGalleryHistoryDirectoryPath()**

Arguments

Argument	Type	Description
None		

Result

Result	Type	Description
filepath	String	Returns a String

Remarks

None

Example

```
path = objApp.GetGalleryHistoryDirectoryPath()
MsgBox "Folder = " & path
```

See also

SetGalleryHistoryDirectoryPath

7.1.2.10 Application::GetGalleryHistoryFilenameIndex

Returns the index for the next file name.

Syntax

index = *app*.**GetGalleryHistoryFilenameIndex()**

Arguments

Argument	Type	Description
None		

Result

Result	Type	Description
index	Number	Returns number > 0

Remarks

None

Example

```
index = objApp.GetGalleryHistoryFilenameIndex()
```

See also

SetGalleryHistoryFilenameIndex

7.1.2.11 Application::GetGalleryHistoryFilenameMask

Returns a *Document* class object at the specified position.

Syntax

fileNameMask = *app*.**GetGalleryHistoryFilenameMask()**

Arguments

Argument	Type	Description
None		

Result

Result	Type	Description
fileNameMask	String	Returns a String containing the filename mask e.g. "image[INDEX]"

Remarks

Example

```
mask = objApp.GetGalleryHistoryFilenameMask()
```

See also

SetGalleryHistoryFilenameMask

7.1.2.12 Application::GetScriptDirectoryPath

Returns the script file path to the directory where scripts are stored.

Syntax

filePath = *app*.**GetScriptDirectoryPath(**Index**)**

Arguments

Argument	Type	Description
Index	Number	0 - Index for measurement scripts 1 - Index for analyzing scripts

Result

Result	Type	Description
filepath	String	Returns a String

Remarks

None

Example

```
path = objApp.GetScriptDirectoryPath(0)
MsgBox "Folder = " & path
```

See also

SetScriptDirectoryPath

7.1.2.13 Application::IsObj

Checks if the specified object is valid

Syntax

ok = *app*.**IsObj(**object**)**

Arguments

Argument	Type	Description
object	Object	IDispatch object handler

Result

Result	Type	Description
ok	Boolean	Returns `True` if the IDispatch object is a valid object reference otherwise `False`.

Remarks

The **IsObj** method checks if a COM Object variable is a valid interface to a IDispatch interface or not.
If a method of any class is returning a object variable this method can check if the return value is a valid interface or not.

The `IsObject()` function of Visual Basic is only checking if the variable is of type 'Object' but not if the stored interface is really valid.

Example

```
objApp.DocDeleteAll          ' make shure no document is open

Dim objDoc : Set objDoc = objApp.GetActiveDoc()
MsgBox objApp.IsObj(objDoc) ' display 'false' because no document is available
MsgBox IsObject(objDoc)     ' display 'true' because variable is of type object

If objApp.IsObj(objDoc) Then
  MsgBox "Selected document is " & objDoc.Name
Else
  MsgBox "No document selected"
End If
```

See also

none.

7.1.2.14 Application::IsStartingUp

Checks if the Nanosurf is busy establishing the microscope connection.

Syntax

flag = *application*.**IsStartingUp**

Result

Result	Type	Description
flag	Boolean	Returns `True` if the application is busy with initialization of the microscope.

Remarks

The IsStartingUp property is monitoring the startup or initialization process of the Nanosurf program.
A script should wait until the startup process is finished before it sends the application further commands.

Example

```
' open application
Dim objApp : Set objApp = CreateObject("Nanosurf_C3000.Application")
Do While objApp.IsStartingUp : Loop
```

```
' do something
....
```

See also

none.

7.1.2.15 Application::LoadCalibration

Loads a scan head calibration from file.

Syntax

ok = *app*.**LoadCalibration(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the calibration file. File extension should be .hed

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be loaded otherwise `False`.

Remarks

This method loads a scan head calibration from a file. The file is a special ini-file formatted file with extension .hed.

Example

```
If objApp.LoadCalibration("10-07-233.hed") == False Then
  MsgBox "Could not load file!"
End If
```

See also

Method SaveCalibration

Version info

Software v1.6.0 or later

7.1.2.16 Application::LoadChartArrangement

Loads a set of chart arrangement from file.

Syntax

ok = *app*.**LoadChartArrangement(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the chart file. File extension should be .chart

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be loaded otherwise `False`.

Remarks

This method loads a set of chart arrangement from a file. The file is a special ini-file formatted file with extension .chart.

Example

```
If objApp.LoadChartArrangement("mycharts.chart") == False Then
  MsgBox "Could not load file!"
End If
```

See also

Method SaveChartArrangement

7.1.2.17 Application::LoadDocument

Load a image document from file.

Syntax

ok = *app*.**LoadDocument(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the image document file. File extension

should be .nid

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be loaded otherwise `False`.

Remarks

This method loads a image document from a file. The file is a Nanosurf special file formate with extension .nid.

Example

```
If objApp.LoadDocument("mysample.nid") == False Then
  MsgBox "Could not load image!"
End If
```

See also

Method SaveDocument

7.1.2.18 Application::LoadParameter

Loads a set of parameters from file.

Syntax

ok = *app*.**LoadParameter(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the parameter file. File extension should be .par

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be loaded otherwise `False`.

Remarks

This method loads a set of parameters from a file. The file is a special ini-file formatted file with extension .par.

Example

```
If objApp.LoadParameter("mysample_settings.par") == False Then
  MsgBox "Could not load file!"
End If
```

See also

Method SaveParameter

7.1.2.19 Application::LoadWorkspace

Loads a workspace from file.

Syntax

ok = *app*.**LoadWorkspacer(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the workspace file. File extension should be .gui

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be loaded otherwise `False`.

Remarks

This method loads a workspace configuration from a file. The file is a special binary-file formatted file with extension .gui.

Example

```
If objApp.LoadWorkspace("mysample.gui") == False Then
  MsgBox "Could not load file!"
End If
```

See also

Method SaveWorkspace

Version info

Software v1.6.0 or later

7.1.2.20 Application::Log

Log a simple message string.

Syntax

app.**Log(**strMessage**)**

Arguments

Argument	Type	Description
strMessage	String	Log message

Result

None

Remarks

This method logs the given string to the "Proxy" log channel with log level "Info". This function is non blocking and asynchronously logs the message.

See also

Method LogEx, LogUserMarker

7.1.2.21 Application::LogEx

Log a message string on a channel with a log level.

Syntax

app.**LogEx(**strChannel, nLevel, strMessage**)**

Arguments

Argument	Type	Description
strChannel	String	Log channel
nLevel	Severity	Log level
strMessage	String	Log message

Result

None

Remarks

This method logs the given string to the given log channel and log level. This function is non blocking and asynchronously logs the message.

See also

Method Log, LogUserMarker

7.1.2.22 Application::LogUserMarker

Generate a user marker into the log system.

Syntax

app.**LogUserMarker()**

Arguments

None

Result

None

Remarks

This method logs a user marker to the "UserMarker" channel with a automatically incremented number. This function is non blocking and asynchronously logs the message.

See also

Method Log, LogEx

7.1.2.23 Application::PrintStatusMsg

Prints a message in the status tool bar.

Syntax

application.**PrintStatusMsg(**message**)**

Arguments

Argument	Type	Description
message	String	Text to print in the status tool bar

Remarks

This method print a text in the first pane of the status tool bar.

Example

```
objApp.PrintStatusMsg "Hello world!"
```

See also

none

7.1.2.24 Application::SaveCalibration

Save the current scan head calibration to file.

Syntax

ok = *app*.**SaveCalibration(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the target scan head calibration file. File extension should be .hed

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be saved otherwise `False`.

Remarks

This method saves the current scan head calibration to file. The file is a special ini-file formatted file with extension .hed.

Example

```
If objApp.LoadCalibration("c:\mycalib\3-07-512-hed") == False Then
  MsgBox "Could not save file!"
End If
```

See also

Method LoadCalibration

Version info

Software v1.6.0 or later

7.1.2.25 Application::SaveChartArrangement

Saves current set of chart arrangement to file.

Syntax

ok = *app*.**SaveChartArrangement(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the chart file. File extension should be .chart

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be saved otherwise `False`.

Remarks

This method saves the current set of chart arrangement to file. The file is a special ini-file formatted file with extension .chart.

Example

```
If objApp.SaveChartArrangement("mycharts.chart") == False Then
  MsgBox "Could not save file!"
End If
```

See also

Method LoadChartArrangement

7.1.2.26 Application::SaveDocument

Save current image document to file.

Syntax

ok = *app*.**saveDocument(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the image document file. File extension should be .nid

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be saved otherwise `False`.

Remarks

This method saves the current image document to file. The file is a Nanosurf special file formate with extension .nid.

Example

```
' measure image
objScan.StartFrameUp
Do While objScan.IsScanning : Loop

' create image and save
objScan.StartCapture
If objApp.SaveDocument("mysample.nid") == False Then
  MsgBox "Could not save image!"
End If
```

See also

Method LoadDocument

7.1.2.27 Application::SaveParameter

Save the current set of parameters to file.

Syntax

ok = *app*.**SaveParameter(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the target parameter file. File extension should be .par

Result

Result	Type	Description
ok	Boolean	Returns `True` if the file could be saved otherwise `False`.

Remarks

This method saves the current set of parameters to file. The file is a special ini-file formatted file with extension .par.

Example

```
If objApp.SaveParameter("mysample_settings.par") == False Then
  MsgBox "Could not save file!"
End If
```

See also

Method LoadParameter

7.1.2.28 Application::SaveWorkspace

SetGalleryHistoryDirectoryPathSetGalleryHistoryDirectoryPath

Save the current workspace configuration to file.

Syntax

ok = *app*.**SaveWorkspace(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the target workspace file. File extension should be .gui

Result

Result	Type	Description

ok	Boolean	Returns `True` if the file could be saved otherwise `False`.

Remarks

This method saves the current workspace configuration to file. The file is a special binary-file formatted file with extension .gui.

Example

```
If objApp.SaveWorkspacer("mysample.gui") == False Then
  MsgBox "Could not save file!"
End If
```

See also

Method LoadWorkspace

Version info

Software v1.6.0 or later

7.1.2.29 Application::SetGalleryHistoryDirectoryPath

Defines the file path where captured data shall be stored.

Syntax

*app.***SetGalleryHistoryDirectoryPath(**Path**)**

Arguments

Argument	Type	Description
Path	String	file path like "C:\some\path\to\a\folder"

Result

Result	Type	Description
None		

Remarks

None

Example

```
objApp.SetGalleryHistoryDirectoryPath("C:\Users\Public\Documents")
```

See also

GetGalleryHistoryDirectoryPath

7.1.2.30 Application::SetGalleryHistoryFilenameIndex

Defines the index for the next captured files.

Syntax

app.**SetGalleryHistoryFilenameIndex(**index**)**

Arguments

Argument	Type	Description
Index	Number	must be >= 0

Result

Result	Type	Description
None		

Remarks

The **SetGalleryHistoryFilenameIndex()** method sets the start offset, meaning setting the index to 0(Zero), the next created image will have the index 1(one).

Example

```
objApp.SetGalleryHistoryFilenameIndex(42)
```

See also

GetGalleryHistoryFilenameIndex

7.1.2.31 Application::SetGalleryHistoryFilenameMask

Defines the filename mask for new captured files.

Syntax

app.**SetGalleryHistoryFilenameMask(**Mask**)**

Arguments

Argument	Type	Description
Mask	String	Filename mask. cannot contain white spaces or slashes. Possible wild cards [INDEX] = 00001.nid [DATE] [TIME] the [INDEX] will always be appended no matter what!

Result

Result	Type	Description
None		

Remarks

Example

```
objApp.SetGalleryHistoryFilenameMask("MyFancyExperiment_[INDEX]")
```

See also

GetGalleryHistoryFilenameMask

7.1.2.32 Application::SetScriptDirectoryPath

Defines the file path where scripts are stored.

Syntax

app.**SetScriptDirectoryPath(**Index, Path**)**

Arguments

Argument	Type	Description
Index	Number	0 - Index for measurement scripts 1 - Index for analyzing scripts
Path	String	file path like "C:\some\path\to\a\folder"

Result

Result	Type	Description
None		

Remarks

None

Example

```
objApp.SetScriptDirectoryPath(0, "C:\Users\Public\Documents")
```

See also

GetScriptDirectoryPath

7.1.2.33 Application::Sleep

Delay the script execution.

Syntax

application.**Sleep(**time**)**

Arguments

Argument	Type	Description
time	double	Delay time in [s].

Remarks

This method delay the execution of the script by the amount of seconds given as argument. The delay precision depends on the workload of the PC and should not be used as a precision timer. Minimal delay is 50ms.

Example

```
' do something

objApp.Sleep(30.0) '[s]

' do something else
```

See also

none

7.2 Approach

The Approach class handles the microscope's approach system.

Controlling the coarse distance between the sensor tip and the sample surface is the main goal of this class. This process can be divided into two separate phases:

- The sensor can be moved fast toward or away from the surface with the methods **StartAdvance** and **StartRetract**.
- The critical action of finally closing the distance between sample and tip until the z feedback controller can sense the surface is done by **StartApproach**. A first release of the contact is done by **StartWithdraw**.

All movements are asynchronously handled by the microscope control electronics. To stop any movement call the method **Stop**. To know if a movement is in process call **IsMoving**. To know if a movement was successful or not call method **Status**.

A object pointer to this class is provided by the Application.Approach object property.

Table of properties for Approach class:

Property name	Purpose
ApproachSpeed	Define the speed used by StartApproach()
WithdrawSpeed	Define the speed used by StartWithdraw()
ApproachMaxSteps	Defines the maximal retries during an automatic approach
AutoStartImaging	This flags defines if the imaging process is started after approach
AutoReloadSettings	This flag defines if prior a approach the parameters are load from file
ShowApproachDoneDialog	Defines if the success dialog is shown or not
ApproachPos	Defines the tip position during approach or readjust the position
IsMoving	Retrieve the information whether a movement is in process or not
AFMApproachMode	Define the approach mode
AFMStepByStepSpeed	Define the speed of movement in Step-By-Step approach mode
AFMStepByStepRange	Define the move range in Step-By-Step approach mode

Table of methods for Approach class:

Method name	Purpose
ShowWindow	Controls the visibility of the imaging window

StartApproach	Starts the automatic final approach toward sample
StartWithdraw	Retract the sensor from surface by a controlled small amount
StartAdvance	Start a fast movement toward sample
StartRetract	Start a fast movement away from sample
StartHome	Start a fast movement until the home position is reached
Stop	Stop any movement
Status	Retrieve the current status of a movement
ForceApproachStatus	Sets the approach status to a given state value
ZMotorStep	Performs a Z motor step
ZMotorStepStop	Stops Z motors step
ForceZMotorPosUpdate	Requests an update of the Z motor positions
ZMotorSetPosZero	Sets current position of given Z Motor to 0.0
LevelScanhead	Levels the scanhead
ZMotorReference	References Z Motors
ZMotorReferenceAndMoveBack	References Z Motors and goes back to the previous position
IsZMotorReferenced	Checks whether Z Motors are referenced
GetZMotorPosition	Returns position of given Z Motor

7.2.1 Properties

7.2.1.1 Approach::ApproachMaxSteps

Returns or set the maximal length of an automatic tip approach.

Syntax

approach.**ApproachMaxSteps** [= steps]

Setting

Argument	Type	Description
steps	long	Defines the number of maximal steps allowed until an abort of the automatic tip approach.

Remarks

The automatic tip approach aborts its search for the surface after a defined number of unsuccessful retries.

In the current AFM scan head design a linear motor is used to move the scan stage. Therefore the number of steps is a time slice during the motor is rotating than an actual step of the motor.

Example

```
objAppr.ApproachMaxSteps = 20000
objAppr.StartApproach
```

See also

Method StartApproach, Property ApproachSpeed

7.2.1.4 Approach::ApproachPos

Returns or set the tip position at approach.

Syntax

approach.**ApproachPos** [= pos]

Setting

Argument	Type	Description
pos	double	Defines the tip position during AFM approach or reposition the tip position

Remarks

The approach position of the tip is controlled by this property. It has two usages:

1. Defines the tip position during the approach process. This is usually 0um which corresponds to mid range of the full z-scan range. Other values are used to approach an measure small high features or narrow deep holes.
2. The stage can by readjusted after approach to re-center the mean tip position. This is usually used if the sample has large drifts. If the ApproachPos property is set after an approach and the z-controller has closed contact the stage is moved my the approach motor until the z-controller's output reaches the new position defined by the property. The movement speed is controlled by ApproachSpeed property. The process can by stopped by the Stop Method.

These two concept are excluding and the user has to select a practical compromise. If the surface is rough and the tip sharpness is not so critical a faster approach speed can be chosen. If the surface has very small details and a sharp tip should be preserved a slower approach speed should be set. Practical values are in the range of 5% to 30%.

Example

```
objAppr.ApproachPos = -1e-6 '[um]
objAppr.StartApproach
```

See also

ApproachSpeed Property, Stop Method

Version info

Available since Software v1.5.0

7.2.1.5 Approach::ApproachSpeed

Returns or set the speed of automatic tip approach or withdraw.

Syntax

approach.**ApproachSpeed** [= speed]

Setting

Argument	Type	Description
speed	double	Defines the speed of automatic approach and withdraw in percent of full speed motor movement

Remarks

The speed of the automatic tip approach should be selected with two ideas in mind:

- To reach the surface as quick as possible a high moving speed would be interesting
- To prevent the tip from damage at closing contact with the surface a smooth and careful approach is desired

These two concept are excluding and the user has to select a practical compromise. If the surface is rough and the tip sharpness is not so critical a faster approach speed can be chosen. If the surface has very small details and a sharp tip should be preserved a slower approach speed should be set. Practical values are in the range of 5% to 30%.

Example

```
objAppr.ApproachSpeed = 10.0 '[%]
objAppr.StartApproach
```

See also

Method StartApproach**,** StartWithdraw

7.2.1.6 Approach::AutoReloadSettings

Returns or set the flag to define if microscope parameter settings should be reload before each approach.

Syntax

approach.**AutoReloadSettings** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if the settings in the current parameter file should be reloaded before each approach.

Remarks

The settings of the microscopes parameter can be automatically reloaded prior an approach is executed. This is useful where each multiple images should be measured exactly with the same settings. A repetitive equal sample measurement in a quality control environment is an example where this flag could be used to ensure equal measurement conditions.

The settings are loaded form the currently active parameter file shown in the status bar.

See also

Method StartApproach

7.2.1.7 Approach::AutoStartImaging

Returns or set the flag to define if imaging is started automatically after a successful approach.

Syntax

approach.**AutoStartImaging** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if imaging should be started after an approach is successful done. `False` if no action should be executed.

Remarks

To automatically start the imaging process after the approach this property can be set. This is useful where the user should take over the instrument after the approach is done. If no user interaction is desired the flag could be set to false in order to control the microscope from the script only.

The start of the imaging is only triggered if a successful "approach done" could be executed. See method Status.

See also

Method StartApproach, Property Status

7.2.1.8 Approach::ShowApproachDoneDialog

Returns or set the flag to define if the "Approach Done" Dialog should be displayed after a successful approach.

Syntax

approach.**ShowApproachDoneDialog** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if the Dialog should be displayed after an approach is successful done. `False` if no dialog should be displayed.

Remarks

This property defines if a dialog should be displayed after a successful approach has been executed. If approach is executed in a script environment this dialog is in many cases unwanted an can be switched of by this property.
A script displayling the dialog should enable it at the end of the script again.

See also

Method StartApproach, Property Status

Version info

Software v1.4.0 or later

7.2.1.9 Approach::WithdrawSteps

Returns or set the length of an automatic tip withdraw.

Syntax

approach.**WithdrawSteps** [= steps]

Setting

Argument	Type	Description
steps	long	Defines the number of steps counted during an automatic tip withdraw.

Remarks

The automatic tip withdraw is used to perform a small tip release from the surface. Normally this is done to move the surface underneath the tip and reapproach afterward.

In the current AFM scan head design a linear motor is used to move the scan stage. Therefore the number of steps is a time slice during the motor is rotating than an actual step of the motor.

Example

```
objAppr.WithdrawSteps = 1000
objAppr.StartWithdraw
```

See also

Method StartWithdraw, Property ApproachSpeed

7.2.2 Methods

7.2.2.1 Approach::IsMoving

Checks if any z approach motor movement is in process.

Syntax

flag = *approach*.**IsMoving**

Result

Result	Type	Description
flag	Boolean	Returns `True` if the z approach motor is moveing.

Remarks

The **IsMoving** property is monitoring the movement of the z approach motor. A script should wait after any call of a Start... method until the movement is finished.

Example

```
' park scan stage
objAppr.StartRetract
Do While objAppr.IsMoving : Loop
```

See also

Method StartApproach, StartWithdraw, StartAdvance, StartRetract

7.2.2.2 Approach::ShowWindow

Defines the display style of the Positioning window.

Syntax

objAppr.**ShowWindow(**style**)**

Arguments

Argument	Type	Description
style	short	Visibility style number

Result

None.

Remarks

The **ShowWindow** method sets the visibility state of the window.

Available styles see Doc.ShowWindow Method

Example

```
objAppr.ShowWindow(0) ' hide the window
```

See also

None.

Version info

Software v1.4.0 or later

7.2.2.3 Approach::StartAdvance

Starts advancing the tip to the surface.

Syntax

approach.**StartAdvance**

Remarks

This method is moving the tip toward the surface. This is a fast movement and is used to shorten the automatic approach. After a preparation of a new sample usually the sensor is far away from the surface and a slow automatic approach would be timeconsuming. During the movement a read of **IsMoving** is `True`. To stop the movement call **Stop** method.

Attention

Because no exact control of the movement is provided this method should be used with great care! Any tip sample contact could damage the tip and measurement with such a tip will be degraded or completely impossible. Use **StartApproach** instead.

See also

Method IsMoving, Stop, StartApproach

7.2.2.4 Approach::StartApproach

Starts the automatic tip approach to the surface.

Syntax

approach.**StartApproach**

Remarks

This method is starting the automatically approach process. Approaching the surface is a first step process before the microscope is ready to perform other surface analysis method as imaging or spectroscopy. During the approach process the tip is moved to the sample surface and the sensor's signal is monitored. The approach is stopped when the sensor signal has reached the setpoint value defined by the z feedback controller.

Operating Mode settings and Z Feedback controller settings should be set to reasonable values prior an approach. Depending on the operating mode special sensor calibration sequences could be executed prior the actual approach movement starts.

The script can wait for the end of the approach by reading the **IsMoving** method. After **IsMoving** returns `False` the reason why the approach stopped should be read with the method **Status**. If an error condition was the reason an appropriate action should be taken by the script (e.g. Display a message box and withdrawing from the surface). To abort the approach call method **Stop**.

Example

```
' prepare approach
objAppr.ApproachSpeed = 10.0
objAppr.AutoStartImaging = False

' approach
objAppr.StartApproach
Do While objAppr.IsMoving : Loop

' if successful do something
If objAppr.Status = 3 Then

  ' approach done -> do something (start imaging or ....)

Else  ' approach error
  MsgBox "Approach error = " & objAppr.Status
End If

 ' finish
 objAppr.StartWithdraw
 Do While objAppr.IsMoving : Loop
```

See also

Property ApproachSpeed, ApproachMaxSteps, AutoStartImaging,
Method IsMoving, Status, Stop
Class OperatingMode, ZController

7.2.2.5 Approach::StartRetract

Starts retracting the tip from surface.

Syntax

approach.**StartRetract**

Remarks

This method is moving the tip away from the surface. This is a fast movement and is used to park the sensor in a far away position from the surface in order to exchange sample or prior shut down of the microscope power. During the movement a read of **IsMoving** is `True`. To stop the movement call **Stop** method. Prior the movement any scanning is stopped and the tip is retracted from the surface.

Because no exact control of the movement is provided this method should be used only in combination with a user interface or a delay timer to define the duration and the length of the movement.

A special case is parking the AFM scan stage in the most retracted upper position. You can call **StartRetract** and wait until **IsMoving** is `False`. Then the scan stage is moved into the upper end switch.

Example

```
' finish
objAppr.StartRetract
Sleep(500)
objAppr.Stop
```

See also

Method IsMoving, Stop

7.2.2.7 Approach::StartWithdraw

Starts withdrawing the tip from the surface.

Syntax

approach.**StartWithdraw**

Remarks

This method is moving the tip away from the surface by a controlled amount. The withdraw length is set by **WithdrawSteps** property.
Prior the movement any scanning is stopped and the tip is retracted from the surface.

The script can wait for the end of the withdraw by reading the IsMoving method. To abort the withdraw call method Stop.
The speed of the withdraw is the same as the approach speed an is set by property **ApproachSpeed**.

Example

```
' finish
objAppr.StartWithdraw
Do While objAppr.IsMoving : Loop
```

See also

Property ApproachSpeed
Method IsMoving, Stop

7.2.2.9 Approach::Status

Returns the current status of the z approach motor and the approach process.

Syntax

status = *approach*.**Status**

Result

Result	Type	Description
status	long	A number naming the state of the z approach stage. See table below.

Remarks

Read this Method to get more information about the state of z approach motor stage. You can call this method during a movement or after the end. It gives you information if a movement was successful or not and why.

Table of approach state number and description:

State No.	Name	Description
0	ApprStat_Standby	No movement
1	ApprStat_Initializing	Preparing of automatic approach in process
2	ApprStat_Approaching	Automatic approach in process
3	ApprStat_ApproachDone	Automatic approach successful finished
4	ApprStat_ApproachAborted	Approach automatically aborted
5	ApprStat_MoveToParkPosition	Moving to park position in process
6	ApprStat_ParkPositionReached	Park position reached
7	ApprStat_MoveAway	Retracting tip from sample in process

8	ApprStat_MoveToward	Advancing tip toward tip in process
9	ApprStat_SensorFailed	AFM sensor error
10	ApprStat_LimitFailed	AFM approach stage failure
11	ApprStat_CalibrationFailed	Initialisation or calibration process failed
12	ApprStat_UserAbort	Movement was stopped by Stop method
13	ApprStat_MaxOut	End of movement reached
14	ApprStat_InitDone	Sensor initialisation finished
15	ApprStat_AdjustingTipPos	readjusting tip position while in contact

Example

```
' approach
objAppr.StartApproach
Do While objAppr.IsMoving : Loop

' check state
If objAppr.Status <> 3 Then
  MsgBox "Approach error = " & objAppr.Status
End If
```

See also

Method StartApproach, StartWithdraw, StartAdvance, StartRetract

7.2.2.10 Approach::Stop

Stops any movement of the z approach motor.

Syntax

approach.**Stop**

Remarks

This method stops any on going movement z approach motor movement

Example

```
' approach with timeout
objAppr.StartApproach
sleep(10000)
If objAppr.IsMoving Then
  objAppr.Stop
  MsgBox "No surface found"
```

```
End If
```

See also

Method IsMoving, StartApproach, StartWithdraw, StartAdvance, StartRetract

7.3 BatchManager

The Stage class handles the batch manager subsystem.

A object pointer to this class is provided by the Application.BatchManager object property.

Table of properties for the BatchManager class:

Property name	Purpose
CurrentPointIndex	Current point index
HasConfigurationFilename	Says if the configuration as a file name associated with it
IsIdle	Says if the batch manager is idle
IsPaused	Says if the batch manager is paused
IsStopFlag	Says if the batch manager has the stop flag set
IsUnconfigured	Says if the batch manager is unconfigured
IsWorking	Says if the batch manager is working

Table of methods for the BatchManager class:

Method name	Purpose
AppendNewPointRecord	Appends a new point record to the list
AppendNewPointRecordFromCurrentPosition	Appends a new point record with the current coordinates
CreateNewConfiguration	Creates a new batch manager configuration
GetChangeSamplePosition	Returns the change sample position
GetConfigurationDescription	Returns the configuration description text
GetPointRecordArgument	Returns a point record argument
GetPointRecordPoint	Returns a point record position
GetReferencePosition	Returns the reference position
GetScript	Returns the script text
LoadConfigurationFile	Loads a configuration file
MoveToChangeSamplePosition	Moves the stage to the change sample position

Pause	Pauses the batch manager processor
RemovePointRecord	Removes a specific point record from the list
SaveConfigurationFile	Saves the configuration
SaveConfigurationFileEx	Saves the configuration to given configuration file
SetChangeSamplePosition	Sets the change sample position
SetConfigurationDescription	Sets the configuration description text
SetPointRecordArgument	Sets a point record argument
SetPointRecordPoint	Sets a point record position
SetReferencePosition	Sets the reference position
SetScript	Sets the script text
Start	Stars the batch manager processor from given list item
Stop	Stops the batch manager processor

7.3.1 Properties

7.3.1.1 BatchManager::CurrentPointIndex

Returns the current point index of the batch manager process. This property is read only.

Syntax

objBatchManager.**CurrentPointIndex** [= index] [read only]

Setting

Argument	Type	Description
index	Boolean	Current point index of batch manager process

Remarks

This returns the current point index of the batch manager process.The index starts with 0 and ends at "point count" - 1.

See also

-

Version info

Software v3.5.0.0 or later

7.3.1.2 BatchManager::HasConfigurationFilename

Returns a flag which says if the batch manager has a configuration file name set or not. This property is read only.

Syntax

objBatchManager.**HasConfigurationFilename** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if a configuration file name is set

Remarks

This flag says if the batch manager has a configuration file name set or not. This is necessary to save the configuration and is implicitly set when **LoadConfigurationFile** was used.

See also

Method CreateNewConfiguration, LoadConfigurationFile, SaveConfigurationFile, SaveConfigurationFileEx

Version info

Software v3.5.0.0 or later

7.3.1.3 BatchManager::IsIdle

Returns a flag which says if the batch manager is idle or not. This property is read only.

Syntax

objBatchManager.**IsIdle** [= flag] [read only]

Setting

Argument	Type	Description

flag	Boolean	`True` if idle

Remarks

This flag says if the batch manager is idle.

See also

Property IsWorking, IsPaused, IsStopFlag

Version info

Software v3.5.0.0 or later

7.3.1.4 BatchManager::IsPaused

Returns a flag which says if the batch manager is paused or not. This property is read only.

Syntax

objBatchManager.**IsPaused** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if is paused

Remarks

This flag says if the batch manager is paused.

See also

Property IsIdle, IsWorking, IsStopFlag

Version info

Software v3.5.0.0 or later

7.3.1.5 BatchManager::IsStopFlag

Returns a flag which says if the batch manager stop flag is set or not. This property is read only.

Syntax

objBatchManager.**IsStopFlag** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if StopFlag is set

Remarks

This flag says if the batch manager has the stop flag set. Because a batch manager operation may need a lot of time to
shutdown, a stop flag signals that a stop is in progress.

See also

Property IsIdle, IsWorking, IsPaused, Method Stop

Version info

Software v3.5.0.0 or later

7.3.1.6 BatchManager::IsUnconfigured

Returns a flag which says if the batch manager is configured or not. This property is read only.

Syntax

objBatchManager.**IsUnconfigured** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if not configured

Remarks

This flag says if the batch manager is configured or not. Most functions can't be used before the batch manager is configured.

See also

Method CreateNewConfiguration, LoadConfigurationFile

Version info

Software v3.5.0.0 or later

7.3.1.7 BatchManager::IsWorking

Returns a flag which says if the batch manager is working or not. This property is read only.

Syntax

objBatchManager.**IsWorking** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if is working

Remarks

This flag says if the batch manager is working.

See also

Property IsIdle, IsPaused, IsStopFlag

Version info

Software v3.5.0.0 or later

7.3.2 Methods

7.3.2.1 BatchManager::AppendNewPointRecord

This method appends a new point record and returns the point list index of it.

Syntax

retval = *objBatchManager*.**AppendNewPointRecord()**

Argument

None

Result

Result	Type	Description
retval	int32	Point list item index

Remarks

The **AppendNewPointRecord** method appends a new point record and returns the point list item index of it

See also

Method RemovePointRecord, AppendNewPointRecordFromCurrentPosition

Version info

Software v3.5.0.0 or later

7.3.2.2 BatchManager::AppendNewPointRecordFromCurrentPosition

This method appends a new point record with the current stage coordinates and returns the point list index of it.

Syntax

retval = *objBatchManager*.**AppendNewPointRecordFromCurrentPosition()**

Argument

None

Result

Result	Type	Description
retval	int32	Point list item index

Remarks

The **AppendNewPointRecord** method appends a new point record with the current stage coordinates and returns the point list item index of it

See also

Method RemovePointRecord, AppendNewPointRecord

Version info

Software v3.5.0.28 or later

7.3.2.3 BatchManager::CreateNewConfiguration

This method creates a new batch manager configuration.

Syntax

objBatchManager.**CreateNewConfiguration()**

Argument

None

Result

None

Remarks

The **CreateNewConfiguration** method creates a new batch manager configuration without file name. If an open configuration has unsaved changes, those are lost. The new configuration is used by the batch manager process immediately and is idle.

See also

Method LoadConfigurationFile, SaveConfigurationFile, SaveConfigurationFileEx, Property HasConfigurationFilename

Version info

Software v3.5.0.0 or later

7.3.2.4 BatchManager::GetChangeSamplePosition

This method returns the change sample position for given axis.

Syntax

retval = *objBatchManager*.**GetChangeSamplePosition(***nVirtualAxisId***)**

Argument

Parameter	Type	Description
nVirtualAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Change sample position

Remarks

The **GetChangeSamplePosition** method returns the change sample position of an axis. The change sample position is a special position that can be moved to, to change the sample.

See also

Method SetChangeSamplePosition, MoveToChangeSamplePosition

Version info

Software v3.5.0.0 or later

7.3.2.5 BatchManager::GetConfigurationDescription

This method returns the configuration file description.

Syntax

retval = *objBatchManager*.**GetConfigurationDescription()**

Argument

None

Result

Result	Type	Description
retval	String	Configuration file description

Remarks

The **GetConfigurationDescription** method returns the configuration file description. This is an unprocessed string which can help identify a configuration or write something about it.

See also

Method SetConfigurationDescription

Version info

Software v3.5.0.0 or later

7.3.2.6 BatchManager::GetPointRecordArgument

This method returns the point argument for given point list item and argument name.

Syntax

retval = *objBatchManager*.**GetPointRecrodArgument(***nPointListIndex, nVirtualAxisId***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index
strArgumentName	String	Argument name

Result

Result	Type	Description
retval	String	Point argument value

Remarks

The **GetPointRecrodArgument** method returns the point argument value of an point list item and argument name.

See also

Method SetPointRecordArgument

Version info

Software v3.5.0.0 or later

7.3.2.7 BatchManager::GetPointRecordPoint

This method returns the point position for given axis and point list item.

Syntax

retval = *objBatchManager*.**GetPointRecrodPoint(***nPointListIndex, nVirtualAxisId***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index
nVirtualAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Point position

Remarks

The **GetPointRecrodPoint** method returns the point position of an axis and point list item.

See also

Method SetPointRecordPoint

Version info

Software v3.5.0.0 or later

7.3.2.8 BatchManager::GetReferencePosition

This method returns the reference position for given axis.

Syntax

retval = *objBatchManager*.**GetReferencePosition(***nVirtualAxisId***)**

Argument

Parameter	Type	Description
nVirtualAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Reference position

Remarks

The **GetReferencePosition** method returns the reference position of an axis. The reference position is added to any point position in the batch manager process.

See also

Method SetReferencePosition

Version info

Software v3.5.0.0 or later

7.3.2.9 BatchManager::GetScript

This method returns the batch manager script.

Syntax

retval = *objBatchManager*.**GetScript()**

Argument

None

Result

Result	Type	Description
retval	String	Batch manager script

Remarks

The **GetScript** method returns the batch manager script. This is the operational heart of the batch manager. While the batch manager is changing the position from point to point, the script is run to perform tasks on the points.

See also

Method SetScript

Version info

Software v3.5.0.0 or later

7.3.2.10 BatchManager::LoadConfigurationFile

This method loads a batch manager configuration from file.

Syntax

objBatchManager.**LoadConfigurationFile(***strFilename***)**

Argument

Parameter	Type	Description
strFilename	String	Batch manager configuration file name

Result

None

Remarks

The **LoadConfigurationFile** method loads a batch manager configuration from file. The configuration is used by the batch manager process immediately and is idle.

See also

Method CreateNewConfiguration, SaveConfigurationFile, SaveConfigurationFileEx, Property HasConfigurationFilename

Version info

Software v3.5.0.0 or later

7.3.2.11 BatchManager::MoveToChangeSamplePosition

This method moves the stage to the change sample position.

Syntax

objBatchManager.**MoveToChangeSamplePosition()**

Argument

None

Result

None

Remarks

The **MoveToChangeSamplePosition** method moves the stage to the change sample position.

See also

Method SetChangeSamplePosition, GetChangeSamplePosition

Version info

Software v3.5.0.0 or later

7.3.2.12 BatchManager::Pause

This method pauses the batch manager process.

Syntax

objBatchManager.**Pause()**

Argument

None

Result

None

Remarks

The **Pause** method pauses the batch manager process. The pause will occur just before the next point would be processed.

See also

Method Start, Stop

Version info

Software v3.5.0.0 or later

7.3.2.13 BatchManager::RemovePointRecord

This method removes the point record with given point list index.

Syntax

objBatchManager.**RemovePointRecord(***nPointListIndex***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index

Result

None

Remarks

The **RemovePointRecord** method sets the point argument value of given point list item and argument name.

See also

Method AppendNewPointRecord

Version info

Software v3.5.0.0 or later

7.3.2.14 BatchManager::SaveConfigurationFile

This method saves a batch manager configuration to file.

Syntax

objBatchManager.**SaveConfigurationFile()**

Argument

None

Result

None

Remarks

The **SaveConfigurationFile** method saves the batch manager configuration to file. **HasConfigurationFilename** must return `True` for this method to work. Else **SaveConfigurationFileEx** must be used.

See also

Method CreateNewConfiguration, LoadConfigurationFile, SaveConfigurationFileEx, Property HasConfigurationFilename

Version info

Software v3.5.0.0 or later

7.3.2.15 BatchManager::SaveConfigurationFileEx

This method saves a batch manager configuration to file.

Syntax

objBatchManager.**SaveConfigurationFileEx(***strFilename***)**

Argument

Parameter	Type	Description
strFilename	String	Batch manager configuration file name

Result

None

Remarks

The **SaveConfigurationFileEx** method saves the batch manager configuration to file. The configuration file name is changed permanently to the saved destination which allows **SaveConfigurationFile** to be used next time.

See also

Method CreateNewConfiguration, LoadConfigurationFile, SaveConfigurationFile, Property HasConfigurationFilename

Version info

Software v3.5.0.0 or later

7.3.2.16 BatchManager::SetChangeSamplePosition

This method sets the change sample position for given axis.

Syntax

objBatchManager.**SetChangeSamplePosition(***nVirtualAxisId, val***)**

Argument

Parameter	Type	Description
nVirtualAxisId	int32	Virtual axis id
val	double	Axis value

Result

None

Remarks

The **SetChangeSamplePosition** method sets the change sample position of given axis. The change sample position is a special position that can be moved to, to change the sample.

See also

Method GetChangeSamplePosition, MoveToChangeSamplePosition

Version info

Software v3.5.0.0 or later

7.3.2.17 BatchManager::SetConfigurationDescription

This method sets the configuration file description.

Syntax

objBatchManager.**SetConfigurationDescription(***strDescription***)**

Argument

Parameter	Type	Description
strDescription	String	Batch manager configuration file description

Result

None

Remarks

The **SetConfigurationDescription** method sets the configuration file description. This is an unprocessed string which can help identify a configuration or write something about it.

See also

Method GetConfigurationDescription

Version info

Software v3.5.0.0 or later

7.3.2.18 BatchManager::SetPointRecordArgument

This method sets the change sample position for given axis and point list item.

Syntax

objBatchManager.**SetPointRecordArgument(***nPointListIndex, strArgumentName, val***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index

strArgumentName	String	Argument name
val	String	Argument value

Result

None

Remarks

The **SetPointRecordArgument** method sets the point argument value of given point list item and argument name.

See also

Method GetPointRecordArgument

Version info

Software v3.5.0.0 or later

7.3.2.19 BatchManager::SetPointRecordPoint

This method sets the change sample position for given axis and point list item.

Syntax

objBatchManager.**SetPointRecordPoint(***nPointListIndex, nVirtualAxisId, val***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index
nVirtualAxisId	int32	Virtual axis id
val	double	Axis value

Result

None

Remarks

The **SetPointRecordPoint** method sets the point position of given axis and point list item.

See also

Method GetPointRecordPoint

Version info

Software v3.5.0.0 or later

7.3.2.20 BatchManager::SetReferencePosition

This method sets the reference position for given axis.

Syntax

objBatchManager.**SetReferencePosition(***nVirtualAxisId, val***)**

Argument

Parameter	Type	Description
nVirtualAxisId	int32	Virtual axis id
val	double	Axis value

Result

None

Remarks

The **SetReferencePosition** method sets the reference position of given axis. The reference position is added to any point position in the batch manager process.

See also

Method GetReferencePosition

Version info

Software v3.5.0.0 or later

7.3.2.21 BatchManager::SetScript

This method sets the batch manager script description.

Syntax

objBatchManager.**SetScript(***strScript***)**

Argument

Parameter	Type	Description
strScript	String	Batch manager script

Result

None

Remarks

The **SetScript** method sets the batch manager script. This is the operational heart of the batch manager. While the batch manager is changing the position from point to point, the script is run to perform tasks on the points.

See also

Method GetScript

Version info

Software v3.5.0.0 or later

7.3.2.22 BatchManager::Start

This method starts the batch manager process at given location.

Syntax

objBatchManager.**Start(***nPointListIndex***)**

Argument

Parameter	Type	Description
nPointListIndex	int32	Point list index

Result

None

Remarks

The **Start** method starts the batch manager process at given location. The batch manager must be idle. To start from the beginning, the location 0 must be set.

See also

Method Stop, Pause

Version info

Software v3.5.0.0 or later

7.3.2.23 BatchManager::Stop

This method stops the batch manager process.

Syntax

objBatchManager.**Stop()**

Argument

None

Result

None

Remarks

The **Stop** method stops the batch manager process. If a script method is running, the stop will occur after this method is finished.

See also

Method Start, Pause

Version info

Software v3.5.0.0 or later

7.4 Chart

The Chart class represents a graphical display of data in a documents window. The data, a chart is displaying, is stored in a associated data container in the document. The properties Group and Signal are specifying this data container.

A chart can display the contents in many styles. They are defined by a set of Properties. These properties are similar to the buttons found in the Application's Chart Toolbar.

Multiple charts are stored in a list by the document and are referenced by a position.

Table of properties of class Chart:

Property name	Purpose
Pos	Display position of the chart in the document window
Group	Group index of the displayed data container
Signal	Signal number of the displayed data container
Type	Type of chart
Filter	Mathematical line by line filter applied to the display data
Active	Activation flag of the chart
AxisShow	Show or hides the axis
RangeAutoSet	Enable the automatically data range algorithm
RangeCenter	Set the centre value of the display data range
RangeSpan	Set the span value of the display data range
ViewSize	Defines the size of the chart in the window

Table of methods of class Chart:

Method name	Purpose
GetDocument	Retrieves the IDispatch object to the charts parent document
OptimiseRange	Calls the range optimization algorithm and updates RangeCenter and RangeSpan
CopyToClipboard	Copy the current chart as a bitmap to the clipboard

7.4.1 Properties

7.4.1.1 Chart::Active

Returns or sets the chart's activation flag

Syntax

objChart.**Active** [= flag]

Setting

Argument	Type	Description
flag	Boolean	*Active* defines the selection state of the chart.

Remarks

The **Active** property reflects the selection state of the chart. Only one chart can be active at a single time. If the activation property is set to `True` and another chart was active this old chart will loos its selection state. Also the user can change the activation by clicking with the left mouse button anywhere in the chart's window area.

Example

```
If objChart.Active Then
  ' do something
End If
```

See also

None.

7.4.1.2 Chart::AxisShow

Returns or sets the chart's axis visibility flag

Syntax

objChart.**AxisShow** [= flag]

Setting

Argument	Type	Description
flag	Boolean	*AxisShow* defines if the axis of the graph is drawn or not.

Remarks

The **AxisShow** property defines if the chart is drawing axis label information or not. Set this property to `True` if axis labels should be displayed.

Example

```
' draw axis labels
objChart.AxisShow = True
```

See also

None.

7.4.1.3 Chart::Filter

Returns or sets the chart's mathematical filter

Syntax

objChart.**Filter** [= filter]

Setting

Argument	Type	Description
filter	short	*filter* defines the mathematical algorithm. It has to be one of the values defined in the table below

Remarks

The **Filter** property defines the mathematical algorithm applyed to each data line prior it is drawn to the chart.

Table of implemented Filter:

Type number	Description
0	RAW Data. (No operation)
1	Mean fit.
2	Line fit
3	Derived Data
4	Parabola fit
5	Polynomial fit

Detailed description of the algorithm are described in the Software Reference Manual.

Example

```
objChart.Filter  = 2  ' activate line fit algo.
```

See also

Software Reference Manual.

7.4.1.4 Chart::Group

Returns or sets the group index of the chart's associated data container.

Syntax

objChart.**Group** [= group]

Setting

Argument	Type	Description
group	short	*group* defines the index of the data container displayed by the chart

Remarks

The **Group** property is storing the group index of the data container display by the chart. To identify a data container the Property Signal has to be set correctly too.

It is legal to set Group and Signal to values which has no associated data container in the document. An empty chart will be display in this case. Negative values are not allowed and are reset to zero.

Example

```
' activate a specific data container (Scan Forward, Topography)
objChart.Group  = 0
objChart.Signal = 1
```

See also

Signal Property

7.4.1.5 Chart::Pos

Returns or sets the position of the chart the document window.

Syntax

objChart.**Pos** [= pos]

Setting

Argument	Type	Description
pos	short	*pos* defines the position of the chart in the list of a document

Remarks

Chart class instances are stored in the parent document in a list. The **Pos** property is containing the list position of a chart. Charts are displayed in the Document window in their list position starting by position zero.

The position of a chart is defining its place on screen. The charts are arranged to fit best the document's window size. The chart with position zero is display first in the top left corner of the document window, subsequent charts are placed below the last chart until the size of the window is reached. Then the new chart is placed one row to the right at the top of the window.

If the value -1 is assigned to the Pos property the chart class is placed at the end of the list and the Pos property value is set accordingly.

Example

```
' move a chart to the end
objChart.Pos = -1
```

See also

Doc.ChartCreate Method

7.4.1.6 Chart::RangeAutoSet

Returns or sets the chart's flag for automatically range selection

Syntax

objChart.**RangeAutoSet** [= flag]

Setting

Argument	Type	Description
flag	Boolean	*RangeAutoSet* defines if the chart's data range is automatically optimized or not.

Remarks

The **RangeAutoSet** defines if the chart's data range is automatically optimized or not. Set this property to `True` if optimisation is desired.

The optimisation algorithm uses histogram analysis to detect the optimal display range for the data. Display range in a document is only optimized at change of properties like Group, Signal and Filter.

To optimize the display range for data calculated by a script call OptimiseRange Method after the calculation is done.

Example

```
' enable optimisation
objChart.RangeAutoSet = True
```

See also

None.

7.4.1.7 Chart::RangeCenter

Returns or sets the chart's center of the display data

Syntax

objChart.**RangeCenter** [= center]

Setting

Argument	Type	Description
center	double	Defines the center value of the displayed data range..

Remarks

The **RangeSpan** is used together with **RangeCenter** and defines values which are displayed. The values have to be inside this range to be display.

Minimal data value = RangeCenter - RangeSpan/2

Maximal data value = RangeCenter + RangeSpan/2

The chart implements a algorithm to optimize RangeCenter and RangeSpan. See OptimiseRange Method.

Example

```
' change the brightness of a chart
objChart.RangeCenter = objChart.RangeCenter * 1.1
```

See also

RangeSpan Property.

7.4.1.8 Chart::RangeSpan

Returns or sets the chart's span of the display data

Syntax

objChart.**RangeSpan** [= span]

Setting

Argument	Type	Description
span	double	*D*efines the span of the displayed data range..

Remarks

The **RangeSpan** is used together with **RangeCenter** and defines values which are displayed. The values have to be inside this range to be display.

Minimal data value = RangeCenter - RangeSpan/2

Maximal data value = RangeCenter + RangeSpan/2

The chart implements a algorithm to optimize RangeCenter and RangeSpan. See OptimiseRange Method.

Example

```
' change the contrast of a chart
objChart.RangeSpan = objChart.RangeSpan*2
```

See also

RangeCenter Property.

7.4.1.9 Chart::Signal

Returns or sets the signal number of the chart's associated data container.

Syntax

objChart.**Signal** [= signal]

Setting

Argument	Type	Description
signal	short	*signal* defines the channel number of the data container displayed

by the chart

Remarks

The **Signal** property is storing the channel number of the data container display by the chart. To identify a data container the property Group has to be set correctly too.

It is legal to set Group and Signal to values which has no associated data container in the document. An empty chart will be display in this case. Negative values are not allowed and are reset to zero.

Example

```
' activate a specific data container (Scan Forward, Topography)
objChart.Group  = 0
objChart.Signal = 1
```

See also

Group Property

7.4.1.10 Chart::Type

Returns or sets the chart's display style for the data values.

Syntax

objChart.**Type** [= type]

Setting

Argument	Type	Description
type	short	*type* defines the display style. It has to be one of the values defined in the table below

Remarks

The **Type** property defines the style of the graph used to display the data values of the data container.

Table of Type styles:

Type number	Description
0	Line graph style
1	Colour map style

2	3D view style
3	Shaded colour map style
4	Dual line graph style
5	XY line graph style

Example

```
objChart.Type  = 3  ' activate shaded colour map
```

See also

none.

7.4.1.11 Chart::ViewSize

Returns or sets the chart's size on screen

Syntax

objChart.**ViewSize** [= size]

Setting

Argument	Type	Description
size	short	Defines the size of the chart on screen in pixel.

Remarks

The **ViewSize** defines the size of the charts output in pixel. Not the outer chart frame size is defined but the actual plot area of the data. This helps preventing aliasing or moire effects on the display if the output size has a even size compared to the number of data measured in a data container.

Example

```
' change the size of a chart
objChart.ViewSize = 256
```

See also

Class Data.

7.4.2 Methods

7.4.2.1 Chart::CopyToClipboard

Copy the current chart as a bitmap to the clipboard.

Syntax

objChart.**CopyToClipboard()**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if successful

Remarks

none

Example

```
objChart.CopyToClipboard
```

See also

7.4.2.2 Chart::GetDocument

Returns a IDispatch object to the parent Document class.

Syntax

objDoc = *objChart*.**GetDocument()**

Arguments

none.

Result

Result	Type	Description
objDoc	Object	A IDispatch object to the parent document class

Remarks

The **GetDocument()** method returns a IDispatch object to the Document class where this class is stored.

Example

```
Set objDoc = objChart.GetDocument()
if objApp.IsObj(objDoc) then
  MsgBox "The chart's parent is : " & objDoc.Name
end if
```

See also

Class Document

7.4.2.3 Chart::OptimiseRange

Recalculate the display range values RangeCenter and RangeSpan.

Syntax

objChart.**OptimiseRange()**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if successful

Remarks

The **OptimiseRange** method calculates new RangeSpan and RangeCenter property values in order to optimize the visibility of the data.

It's using depending on the charts display type and filter different algorithm.

- Colour map types is using a calculation an histogram of the data and find the best value range out of this analysis.

- The line graph types is using a histogram analysis too but with different thresholds.

It's useful to call this method after a script has calculated new data and filled them into a data container.

Example

```
objChart.OptimiseRange ' maximize and activate this document
```

See also

RangeAutoSet Property, RangeCenter Property, RangeSpan Property.

7.5 Data

The Data class represents a storage container for measured data values. The data values are named as points. Multiple points are organized in a line. Multiple such Data lines are stored in the container. Another way on looking at the stored data is that of a 2D-Matrix with a with of Points and a height of Lines.

Data are stored as 16 bit values in the matrix but the Data class knows the physical data values and is able to convert between the internal 16Bit Raw data and the physical values. Therefore the class saves for each axis a name, a unit, a minimum and a range value. See

The contents of each line can by flagged with attributes about its validity. This is useful for algorithms or chart display classes to know which contents is meaningful or new. See method **SetLineFlag** or property **BufferEmpty**.

Table of properties of class Data:

Property name	Purpose
Points	Number of data values per line
Lines	Number of data lines per container
CurrentLine	Active line
BufferEmpty	Flags if container has real data stored or is just initialized
AxisPointName	Name string of the point axis
AxisPointUnit	Physical unit of the point axis
AxisPointMin	Physical value of first point in line
AxisPointRange	Physical value range of from first to last point in line

AxisLineName	Name string of the line axis
AxisLineUnit	Physical unit of the line axis
AxisLineMin	Physical value of first line in container
AxisLineRange	Physical value range of from first to last line in container
AxisSignalName	Name string of the signal axis
AxisSignalUnit	Physical unit of the signal axis
AxisSignalMin	Physical value of most negative data value
AxisSignalRange	Physical value range of over the full 16Bit range
LineDataPoints	Number of data values of a specified line
LineDataMin	Physical value of first point in a specified line
LineDataRange	Physical value range of from first to last point in a specified line

Table of methods of class Data:

Method name	**Purpose**
SetLine / SetLine2	Write an string array of points in the container in different data format. Value passed as String or Variant Array
GetLine / GetLine2	Retrieve an string array of points from the container in different data format. Value passed as String or Variant Array
SetPixel / SetPixel2	Write a data point in different data format. Value passed as String / Variant
GetPixel / GetPixel2	Read a data point in different data format. Value passed as String / Variant
SetLineRAW / SetLineRAW2	Save an array of points in the container as 16/32Bit values. Value passed as String / Variant Array
GetLineRAW / SetLineRAW2	Retrieve an array of points from the container as 16/32Bit values. Value passed as String / Variant Array
SetPixelRAW / SetPixelRAW2	Write a 16/32Bit data point. Value passed as String / Variant
GetPixelRAW / GetPixelRAW2	Read a 16/32Bit data point. Value passed as String / Variant
SetLineFlags	Modify the state flag of a stored line
GetLineFlags	Read the state flag
GetDocument	Retrieves the IDispatch object to the charts parent document

GetGroupID	Retrieves the ID associated with this container
GetGroup	Retrieves the group index associated with this container
GetSignal	Retrieves the signal number associated with this container
RemoveLine	Remove a specified data line
SwapLines	Swap the content of two lines

7.5.1 Properties

7.5.1.1 Data::AxisLineMin

Returns or sets the physical minimal value used by the line axis.

Syntax

objData.**AxisLineMin** [= minium]

Setting

Argument	Type	Description
minimum	double	Physical mininimal value

Remarks

The **AxisLineMin** physical value corresponds to the line with index zero (bottom one).

Example

```
' set the physical range of the line axis
objData.AxisLineUnit  = "m"    'meter
objData.AxisLineMin   = 0.0
objData.AxisLineRange = 1e-6
```

See also

AxisLineUnit Property, AxisLineRange Property

7.5.1.2 Data::AxisLineName

Returns or sets the name of the line axis.

Syntax

objData.**AxisLineName** [= name]

Setting

Argument	Type	Description
name	string	Name of the axis

Remarks

Each axis has its own name. This name is display along the graph in the chart display.

Example

```
' set the name of the axis
objData.AxisPointName  = "X-Axis"
objData.AxisLineName   = "Y-Axis"
objData.AxisSignalName = "Topography"
```

See also

AxisLineUnit Property, AxisLineMin Property, AxisLineRange Property

7.5.1.3 Data::AxisLineRange

Returns or sets the physical range value used by the line axis.

Syntax

objData.**AxisLineRange** [= range]

Setting

Argument	Type	Description
range	double	Physical range of the axis

Remarks

The **AxisLineRange** value defines the physical value range span over all data lines in the container.
The maximal physical value of the top line **Lines**-1 is **AxisLineMin**+**AxisLineRange**.

Example

```
' set the physical range of the line axis
objData.AxisLineUnit  = "m"    'meter
objData.AxisLineMin   = 0.0
objData.AxisLineRange = 1e-6
```

See also

AxisLineUnit Property, AxisLineMin Property

7.5.1.4 Data::AxisLineUnit

Returns or sets the physical unit used by the line axis.

Syntax

objData.**AxisLineUnit** [= unit]

Setting

Argument	Type	Description
unit	string	Physical unit name of the axis

Remarks

The values of an axis can be display by physical units. The unit has to be defined is in its base without exponential extension like 'n' for nano. The chart is responsible to display the values in an appropriate way.

Example

```
' set the physical range of the line axis
objData.AxisLineUnit  = "m"    'meter
objData.AxisLineMin   = 0.0
objData.AxisLineRange = 1e-6
```

See also

AxisLineMin Property, AxisLineRange Property

7.5.1.5 Data::AxisPointMin

Returns or sets the physical minimal value used by the point axis.

Syntax

objData.**AxisPointMin** [= minium]

Setting

Argument	Type	Description
minimum	double	Physical mininimal value

Remarks

The **AxisPointMin** physical value corresponds to the point with index zero (most left one).

Example

```
' set the physical range of the point axis
objData.AxisPointUnit  = "m"     'meter
objData.AxisPointMin   = 0.0
objData.AxisPointRange = 1e-6
```

See also

AxisPointUnit Property, AxisPointRange Property

7.5.1.6 Data::AxisPointName

Returns or sets the name of the point axis.

Syntax

objData.**AxisPointName** [= name]

Setting

Argument	Type	Description
name	string	Name of the axis

Remarks

Each axis has its own name. This name is display along the graph in the chart display.

Example

```
' set the name of the axis
objData.AxisPointName  = "X-Axis"
objData.AxisLineName   = "Y-Axis"
objData.AxisSignalName = "Topography"
```

See also

AxisPointUnit Property, AxisPointMin Property, AxisPointRange Property

7.5.1.7 Data::AxisPointRange

Returns or sets the physical range value used by the point axis.

Syntax

objData.**AxisPointRange** [= range]

Setting

Argument	Type	Description
range	double	Physical range of the axis

Remarks

The **AxisPointRange** value defines the physical value range span over all data point in a line.
The maximal physical value of the last point **Points**-1 is **AxisPointMin** +**AxisPointRange**.

Example

```
' set the physical range of the point axis
objData.AxisPointUnit  = "m"    'meter
objData.AxisPointMin   = 0.0
objData.AxisPointRange = 1e-6
```

See also

AxisPointUnit Property, AxisPointMin Property

7.5.1.8 Data::AxisPointUnit

Returns or sets the physical unit used by the point axis.

Syntax

objData.**AxisPointUnit** [= unit]

Setting

Argument	Type	Description

unit	string	Physical unit name of the axis

Remarks

The values of an axis can be display by physical units. The unit has to be defined is in its base without exponential extension like 'n' for nano. The chart is responsible to display the values in an appropriate way.

Example

```
' set the physical range of the point axis
objData.AxisPointUnit  = "m"    'meter
objData.AxisPointMin   = 0.0
objData.AxisRangeRange = 1e-6
```

See also

AxisPointMin Property, AxisPointRange Property

7.5.1.9 Data::AxisSignalMin

Returns or sets the physical minimal value defined for the minimal data value

Syntax

objData.**AxisSignalMin** [= minium]

Setting

Argument	Type	Description
minimum	double	Physical mininimal value

Remarks

The **AxisSignalMin** physical value corresponds to the minimal 16Bit data value of -32768 (-2^15).

Example

```
' set the physical range of the data values to +-10V
objData.AxisSignalUnit  = "V"    'voltage
objData.AxisSignalMin   = -10.0
objData.AxisSignalRange = 20.0
```

See also

AxisSignalUnit Property, AxisSignalRange Property

7.5.1.10 Data::AxisSignalName

Returns or sets the name of the signal values stored in the container.

Syntax

objData.**AxisSignaöName** [= name]

Setting

Argument	Type	Description
name	string	Name of the axis

Remarks

The data values stored in a container can be lable by this name. This name is display on top of the graph in the chart display.

Example

```
' set the name of the axis
objData.AxisPointName  = "X-Axis"
objData.AxisLineName   = "Y-Axis"
objData.AxisSignalName = "Topography"
```

See also

AxisSignaöUnit Property, AxisSignalMin Property, AxisSignalRange Property

7.5.1.11 Data::AxisSignalRange

Returns or sets the physical range value defined for the full data range

Syntax

objData.**AxisSignalRange** [= range]

Setting

Argument	Type	Description
range	double	Physical range of data values

Remarks

The **AxisSignalRange** value defines the physical value range span over the 16Bit data value range.
The maximal physical value of the maximal data value (2^15-1=+32767) is **AxisSignalMin**+**AxisSignalRange**.

Example

```
' set the physical range of the data values to +-10V
objData.AxisSignalUnit  = "V"    'voltage
objData.AxisSignalMin   = -10.0
objData.AxisSignalRange = 20.0
```

See also

AxisSignalUnit Property, AxisSignalMin Property

7.5.1.12 Data::AxisSignalUnit

Returns or sets the physical unit used by the signal axis.

Syntax

objData.**AxisSignalUnit** [= unit]

Setting

Argument	Type	Description
unit	string	Physical unit name of the axis

Remarks

The values of the data values stored in the container can be display by physical units. The unit has to be defined is in its base without exponential extension like 'n' for nano. The chart is responsible to display the values in an appropriate way.

Example

```
' set the physical range of the data values to +-10V
objData.AxisSignalUnit  = "V"    'voltage
objData.AxisSignalMin   = -10.0
objData.AxisSignalRange = 20.0
```

See also

AxisSignalMin Property, AxisSignalRange Property

7.5.1.13 Data::BufferEmpty

Returns or sets the flag indicating if the data container has valid data or not

Syntax

objData.**BufferEmpty** [= flag]

Setting

Argument	Type	Description
flag	Boolean	`True` if no data are stored in the container

Remarks

The container is flagged as empty when the buffer is initialized or set by this property manually.
It is automatically flagged as not empty if one of the data store methods are called.

Example

```
' display the contents of a container
If Not objData.BufferEmpty Then
  MsgBox "Stored signal is :" & objData.AxisSignalName
End If
```

See also

SetLine Method, SetLineRAW Method, SetPixel Method, SetPixelRAW Method

7.5.1.14 Data::CurrentLine

Returns or sets the number of data lines stored in the container.

Syntax

objData.**CurrentLine** [= line]

Setting

Argument	Type	Description
line	short	defines which line index should be the current one

Remarks

One data line is marked as the current one. These marking is distributed to all data container of a document with the same GroupID. The current line will be used by charts to highlight the special line. The current line is automatically set by data modification methods like **SetLine**().

The range of valid numbers for CurrentLine is 0 to **Lines**-1.

Example

```
' extract the first data value of the current line
val = objData.GetPixelRAW(0,objData.CurrentLine)
```

See also

Lines Property, **SetLine Method**

7.5.1.15 Data::Lines

Returns or sets the number of data lines stored in the container.

Syntax

objData.**Lines** [= lines]

Setting

Argument	Type	Description
lines	short	*lines* defines the number data lines stored in the container

Remarks

Data values are stored in the container as a matrix in the form point x lines. The memory reserved for the matrix is defined by the **Points** and **Lines** properties.

The minimum matrix size is a 1 x 1 matrix. The maximum a 2048 x 2048. The size do not have to be symmetrical (e.g A single measurement line of 128 data points is stored in a 128 x 1 matrix).

If the size of the matrix is changed all data are lost and the matrix is initialized with zero values, all line flags are set to *Invalid* and the buffer is marked as empty.

Example

```
' initialize a data container for a single measurement line
objData.Points = 256
objData.Lines  = 1
```

See also

Points Property, BufferEmpty Property, Setline flags Method

7.5.1.16 Data::Points

Returns or sets the number of data values stored in each data line.

Syntax

objData.**Points** [= points]

Setting

Argument	Type	Description
points	short	*points* defines the number data values stored in each line

Remarks

Data values are stored in the container as a matrix in the form point x lines. The memory reserved for the matrix is defined by the **Points** and **Lines** properties.

The minimum matrix size is a 1 x 1 matrix. The maximum a 2048 x 2048. The size do not have to be symmetrical (e.g A single measurement line of 128 data points is stored in a 128 x 1 matrix).

If the size of the matrix is changed all data are lost and the matrix is initialized with zero values, all line flags are set to *Invalid* and the buffer is marked as empty.

Example

```
' initialize a data container for a single measurement line
objData.Points = 256
objData.Lines  = 1
```

See also

Lines Property, BufferEmpty Property, Setline flags Method

7.5.2 Methods

7.5.2.1 Data::GetDocument

Returns a IDispatch object to the parent Document class.

Syntax

objDoc = *objData*.**GetDocument()**

Arguments

none.

Result

Result	Type	Description
objDoc	Object	A IDispatch object to the parent document class

Remarks

The **GetDocument()** method returns a IDispatch object to the Document class where this class is stored.

Example

```
Set objDoc = objData.GetDocument()
if objApp.IsObj(objDoc) then
  MsgBox "The data's are is stored in: " & objDoc.Name
end if
```

See also

Class Document

7.5.2.2 Data::GetGroup

Returns the data objects group index of the parent Document class.

Syntax

pos = *objData*.**GetGroup()**

Arguments

none.

Result

Result	Type	Description
pos	short	Returns the group index of the data container

Remarks

The **GetGroup()** method returns the group index where this data class ist stored in the list of containers of the parent document class. The exact position is defined in combination with **GetSignal()** method.

Example

```
mysignal = objData.GetSignal()
mygroup  = objData.GetGroup()
```

See also

Class Document, GetSignal Method

7.5.2.3 Data::GetGroupID

Returns the data objects group ID number of the parent Document class.

Syntax

id = *objData*.**GetGroupID()**

Arguments

none.

Result

Result	Type	Description
id	short	Returns the ID number of the data container

Remarks

The **GetGroupID()** method returns the group ID associated with this data container in the parent document class.

Example

```
myid    = objData.GetGroupID()
mygroup = objData.GetGroupPos()
```

See also

Class Document, GetGroupPos Method

7.5.2.4 Data::GetLine / GetLine2

Returns a array of data values of a stored data line.

Syntax

str_array = *objData*.**GetLine(***line,filter,conversion***)**
variant_array = *objData*.**GetLine2(***line,filter,conversion***)**

Argument

Parameter	Type	Description
line	short	desired line index
filter	short	index of mathematical filter to be used
conversion	short	index of conversion type of results

Result

Result	Type	Description
str_array	String	Character string with comma separated values of all the values of the data line
variant_array	double	Variant array of numbers of all the values of the data line

Remarks

This method returns a string of data values of a data line stored in the container. The signal will be extracted and the data values are processed with a filters as available for the user in the "Chart Toolbar". The result is in a comma separated string in different numerical formats.

The argument *line* is the number of the data line to extract. 0 is the bottom line and the value property **Lines** -1 the top most one.

The argument *filter* defines the data processing algorithm to be used.

Table of filter index:

Filter No.	Filter Name	Description
0	FilterRaw	No data processing

1	FilterMean	The mean value is subtracted
2	FilterPlane	The background plane is subtracted
3	FilterDerive	The derivative of the signal is calculated
4	FilterParabola	A second order fit is subtracted
5	FilterPolynominal	A forth order fill is subtracted

For more detailed description of the filter algorithm please refer to the Nanosurf Software Reference Manual.

The argument *conversion* defines the format of the resulting string array.

Table of conversion index:

Conversion No.	Conversion Name	Description
0	ConversionBinary16	Output as signed 16bit data values
1	ConversionPhysical	Output as floating point values in physical base unit
2	ConversionBinary32	Output as signed 32bit data values

Example

```
' get data line 5 with no filter and as 16bit values
dataline = objData.GetLine(5,0,0)
MsgBox dataline

' calc mean value of current line, plane fit filter active and in physical units
dataline = objData.GetLine(objData.Currentline,2,1)
dataarray = Split(dataline,",")
sum = 0.0
For i = 0 To objData.Points-1
  sum = sum + CDbl(dataarray(i))
Next
MsgBox "Mean value of line " & objData.CurrentLine & " is " & (sum /
objData.Points)
```

See also

Lines Property, SetLine Method

7.5.2.5 Data::GetLineFlags

Get the line attributes

Syntax

mask = *objData*.**GetLineFlag(***line***)**

Argument

Parameter	Type	Description
line	short	desired line index

Result

Result	Type	Description
mask	short	Current list of attributs set for the line

Remarks

This method reads the attributes of a line.

See SetLineFlags Method for defined attributs.

Example

```
' calc mean value of data container but ignore invalid lines
sum = 0.0
validlines = 0
For y = 0 To objData.Lines-1
  If objData.GetLineFlags(y) <> 0 Then
    dataline = objData.GetLine(y,0,1)
    dataarray = Split(dataline,",")
    For x = 0 To objData.Points-1
      sum = sum + CDbl(dataarray(x))
    Next
    validlines = validlines + 1
  End If
Next
if validlines > 0 then
  MsgBox "Mean value of container is " & (sum / validlines)
else
  MsgBox "No valid data in container"
end if
```

See also

Lines Property, SetLineFlags Method

7.5.2.6 Data::GetLineRAW / GetLineRAW2

Returns a string of data values or a variant array of a stored data line.

Syntax

str_array = *objData*.**GetLineRAW(***line***)**
varinat_array = *objData*.**GetLineRAW2(***line***)**

Argument

Parameter	Type	Description
line	short	desired line index

Result

Result	Type	Description
str_array	String	Character string with comma separated values of all the values of the data line
varinat_array	int16, Int32	Variant array of all the values in the line

Remarks

This method returns a array of data values of a data line stored in the container.
The result is in a comma separated string in a numerical format.
The range of this numbers is for C3000 32Bit, for all other 16Bit.

The argument *line* is the number of the data line to extract. 0 is the bottom line and the value property **Lines** -1 the top most one.

This is a faster but simpler version of GetLine Method. Not data processing nor conversion is done.

Example

```
' get quickly the current line
dataline = objData.GetLine(objData.CurrentLine)
MsgBox dataline
```

See also

Lines Property, GetLine Method, SetLineRAW Method

7.5.2.7 Data::GetPixel / GetPixel2

Returns the data value of a specified point as string

Syntax

str_val = *objData*.**GetPixel(***point,line,filter,conversion***)**
variant_val = *objData*.**GetPixel2(***point,line,filter,conversion***)**

Argument

Parameter	Type	Description
point	short	desired point number
line	short	desired line index
filter	short	index of mathematical filter to be used
conversion	short	index of conversion type of results

Result

Result	Type	Description
val	String	String of the data value in the desired conversion format
variant_val	double	Number of the data value in the desired conversion format

Remarks

This method returns a string with the data value at a specified (point,line) position. The data value is processed with a filter defined by *filter*. The result is a string value in different numerical formats.

The argument *point* is the position index in the data line to be read. The index has to be from 0 to **Points** -1.
The argument *line* is the number of the data line to extract. 0 is the bottom line and **Lines** -1 the top most one.

The argument *filter* and *conversion* defines the data processing algorithm and formatting to be used.
See parameter tables at GetLine.

Example

```
' get data at (10,20) with no filter and as 16bit values
dataxy = objData.GetPoint(10,20,0,0)
MsgBox dataxy
```

See also

SetPixel Method, GetLine Method

7.5.2.8 Data::GetPixelRAW / GetPixelRAW2

Returns the data value of a specified point as string

Syntax

str_val = *objData*.**GetPixel(***point,line***)**
variant_val = *objData*.**GetPixel(***point,line***)**

Argument

Parameter	Type	Description
point	short	desired point number
line	short	desired line index

Result

Result	Type	Description
str_val	long	data value as string
variant_val	int16, int32	data value as integer

Remarks

This method returns the data value at a specified (point,line) position.
The result is in a string in a numerical format.
The range of this numbers is for C3000 32Bit, for all other 16Bit.

The argument *point* is the position index in the data line to be read. The index has to be from 0 to **Points** -1.
The argument *line* is the number of the data line to extract. 0 is the bottom line and **Lines** -1 the top most one.

This is a faster but simpler version of GetPixel Method. Not data processing nor conversion is done.

Example

```
' get data at (10,20)
dataxy = objData.GetPointRAW(10,20)
MsgBox dataxy
```

See also

SetPixel Method, SetPixelRAW Method, GetLine Method

7.5.2.9 Data::GetSignal

Returns the data objects signal number of the parent Document class.

Syntax

pos = *objData*.**GetSignal()**

Arguments

none.

Result

Result	Type	Description
pos	short	Returns the signal position number of the data container

Remarks

The **GetSignal()** method returns the signal position number where this data class ist stored in the list of containers of the parent document class. The exact position is defined in combination with **GetGroup()** method.

Example

```
mysignal = objData.GetSignal()
mygroup  = objData.GetGroup()
```

See also

Class Document, GetGroup Method

7.5.2.10 Data::SetLine / SetLine2

Store a string of data values into the container

Syntax

ok = *objData*.**SetLine(***line,conversion, str_dataarray***)**
ok = *objData*.**SetLine(***line,conversion, variant_dataarray***)**

Argument

Parameter	Type	Description
line	short	desired line index
conversion	short	conversion type used for processing data string
dataarray	short	String array with comma separated values
variant_dataarray	number	variant of numbers array with comma separated values

Result

Result	Type	Description
ok	Boolean	`True` is successful

Remarks

This method write a string of data values into a data line of the container.

The argument *line* is the number of the data line to be overwritten. 0 is the bottom line and the value property **Lines** -1 the top most one.

The argument *conversion* defines the format of the data string array. Table of conversion index:

Conversion No.	Conversion Name	Description
0	ConversionBinary16	Values are signed 16bit data number
1	ConversionPhysical	Values are floating point number in physical base unit
2	ConversionBinary32	Values are signed 32bit data number

The actual data is parameter *dataarray*. It have to be a comma separated string array of values in the specified format as declared in *conversion*.

Note: There are localization version of the operating systems where numbers are displayed with a comma as decimal points (e.g. German version of Windows). To support these OS versions, its possible to use a semi column character to separate numbers.

Example

```
' flatten and apply maximum threshold
MaxValue = 10.0e-9 'm

For curline = 0 To objData.Lines-1
  dataline = objData.GetLine(curline,2,1)
  dataarray = Split(dataline,",")

  For i = 0 To objData.Points-1
    If CDbl(dataarray(i)) > MaxValue Then
      dataarray(i) = MaxValue
    End If
  Next

  dataline = Join(dataarray,";")
  ok = objData.SetLine(curline,1,dataline)
Next
```

See also

Lines Property, Points Property, GetLine Method

Version info

Semi column as separator character: Software v1.6.1 or later

7.5.2.11 Data::SetLineFlags

Set the line attributes

Syntax

ok = *objData*.**SetLineFlag(***line, mask***)**

Argument

Parameter	Type	Description
line	short	desired line index
mask	short	List of attributes to set

Result

Result	Type	Description
ok	Boolean	`True` is successful

Remarks

This method defines the attributes of a line. to set multible attributes just added they values together.

Table of attributes index:

Atrribute value	Conversion Name	Description
1	DataValid	The values in the line are valid number for processing
2	CurrentData	This attribute marks the data values in the line as new

At initialisation of a Data object or after resizing the *DataVaild* attribute is cleared. It is set automatically by a call of SetLine() method. Data processing algorithm should ignore data lines with cleared DataVaild attribute.
A data line can have the *CurrentData* attribute set. This is useful to distinguish between old and new data in the same data container (e.g during a imaging a container may be partly filled by data measured by an up frame while scanning down and some data are overwriten with the new scan line as they are measured).

Example

```
' mark line zero as Valid and Current
ok = objData.SetLineFlag(0,1+2)
```

See also

Lines Property, GetLineFlags Method

7.5.2.12 Data::SetLineRAW / SetLineRAW2

Store a string of data values into the container

Syntax

ok = *objData*.**SetLineRAW(***line, str_dataarray***)**
ok = *objData*.**SetLineRAW2(***line, variant_dataarray***)**

Argument

Parameter	Type	Description

line	short	desired line index
str_dataarray	short	String array with comma separated values
varinat_dataarray	long	binary array of values

Result

Result	Type	Description
ok	Boolean	`True` is successful

Remarks

This method write a string of data values into a data line of the container.

The argument *line* is the number of the data line to be overwritten. 0 is the bottom line and the value property **Lines** -1 the top most one.

The actual data is parameter *dataarray*.
The result is in a string array in a numerical format.
The range of this numbers is for the C3000 controller 32Bit, for all other systems 16Bit.

This is a faster but simpler version of SetLine Method. Not data processing nor conversion is done.

Example

```
' replace some data values in the top line
ok = objData.SetLine(objData.Lines-1,"-1,2,-3,4,-5,6,-7,8")
```

See also

Lines Property, Points Property, GetLineRAW Method, SetLine Method

7.5.2.13 Data::SetPixel / SetPixel2

Overwrite a data point with new value

Syntax

ok = *objData*.**SetPixel(***point,line,conversion, str_value***)**
ok = *objData*.**SetPixel2(***point,line,conversion, variant_value***)**

Argument

Parameter	Type	Description
point	short	point index of destination position
line	short	line index of destination position
conversion	short	conversion type used for processing data string
str_value	string	string with value in specified format
variant_value	double	double value in specified format

Result

Result	Type	Description
ok	Boolean	`True` is successful

Remarks

This method write a new value to a specified position in the container.

The argument *point* is the position index in the data line to be read. The index has to be from 0 to **Points** -1.
The argument *line* is the number of the data line to extract. 0 is the bottom line and **Lines** -1 the top most one.
The argument *conversion* defines the data format to be used. See parameter table at SetLine.
The argument *value* contains the new value in the specified format as described in *conversion*.

Example

```
' write at (0,0) the value 1nm
objData.AxisSignalUnit = "m"
ok = objData.SetPixel(0,0,1,"1e-9")
```

See also

Lines Property, Points Property, SetLine Method

7.5.2.14 Data::SetPixelRAW / SetPixelRAW2

Overwrite a data point with new value

Syntax

ok = *objData*.**SetPixel(***point,line,str_value***)**
ok = *objData*.**SetPixel2(***point,line,variant_value***)**

Argument

Parameter	Type	Description
point	short	point index of destination position
line	short	line index of destination position
str_value	long	New data value as string
variant_value	long	new data value as number

Result

Result	Type	Description
ok	Boolean	`True` is successful

Remarks

This method write a new value to a specified position in the container.

The argument *point* is the position index in the data line to be read. The index has to be from 0 to **Points** -1.
The argument *line* is the number of the data line to extract. 0 is the bottom line and **Lines** -1 the top most one.
The argument *value* contains the new value to be stored.
The range of this number is for the C3000 controller 32Bit, for all other systems 16Bit.

This is a faster but simpler version of SetPixel Method. Not data processing nor conversion is done.

Example

```
' write at (0,0) the value 1nm
objData.AxisSignalUnit = "m"
ok = objData.SetPixel(0,0,1,"1e-9")
```

See also

Lines Property, Points Property, SetLine Method

7.6 Document

The Document class is a container for measured data and its visual representation.

Complete documents can be loaded or stored from/to the file system.

Its information is stored in three lists of the following types:

1. Measured values: Data values for signal channels are stored in data container. Referenced by Data classes.
2. Visual appearance: Charts are displaying measured data with different styles on screen. Each chart is stored in a Chart class.
3. General information: Additional information is grouped in sections of key value pairs. Each info section is stored in a Info class.

Objects in these lists are retrieved by search methods.

New objects can be created and existing objects in the lists can be deleted.

For detailed description on how these lists are organized, refer to the individual chapter of Class Data, Class Chart and Class Info.

Table of properties for Document class:

Property name	Purpose
Name	Contains then filename of the document

Table of methods for general usage of Document class:

Method name	Purpose
Load	Load the contents of a file into the document
Save	Saves the content of the document into a file
ShowWindow	Control the windows visual state

Table of methods for Chart object of document class:

Method name	Purpose
ChartCount	Retrieves the number of charts displayed in the document window
ChartCreate	Create a new Chart object and display it
ChartGetActive	Return a Chart object to the currently active chart

ChartGetByPos	Return a Chart object to the chart at a position
ChartDeleteByPos	Removes the chart at position
ChartDeleteAll	Removes all charts of this document

Table of methods for Data object of document class:

Method name	**Purpose**
DataGroupCount	Retrieves the number of data groups
DataSignalCount	Retrieves the number of data objects in a specified group
DataCreate	Creates a new Data class for a specified group and signal
DataGetActive	Returns a Data object of the signal displayed by the active chart
DataGetByName	Returns a Data object with the specified group and signal name
DataGetByPos	Returns a Data object with the specified group and signal number
DataDeleteByName	Deletes the stored values of a specified group and signal name
DataDeleteByPos	Deletes the stored values of a specified group and signal number
DataDeleteGroup	Deletes a complete group of values
DataDeleteAll	Deletes all measured values
DataGetGroupID	Retrieves the ID number of a specified group
DataSetGroupID	Sets the ID number of a specified group
DataGetGroupName	Retrieves the name of a specified group
DataSetGroupName	Change the name of a specified group
DataGetGroupPos	Retrieves the index of a named group
DataGetSignalPos	Retrieves the number of a signal in a group an known signal name

Table of methods for Info objects of document class:

Methode name	**Purpose**
InfoCount	Retrieves the number of info section in the document
InfoCreate	Creates a new Info class with a specified name
InfoGetByName	Returns a Info object with a specified name
InfoGetByPos	Returns a Info object at a specified position
InfoDeleteByName	Removes a information section with a specified name
InfoDeleteByPos	Removes a information section at a specified position

InfoDeleteAll	Removes all sections

7.6.1 Properties

7.6.1.1 Document::Name

Returns or sets the filename of the document.

Syntax

objDoc.**Name** [= filename]

Setting

Argument	Type	Description
filename	String	*filename* is a string containing the path and filename of the document.

Remarks

The Name property is containing the unique name of the document. If it is loaded from file or stored already to a file the name is its path and filename.
The name of a newly created document is a path to its temporary storage and a automaticaly assigne name.

Example

```
Dim objDoc : Set objDoc = objApp.DocCreate("",Nothing)
MsgBox "Auto assigned name is " & objDoc.Name
```

See also

DocCreate Method, Load Method, Save Method

7.6.2 Methods

7.6.2.1 Document::ChartCount

Retrieves the number of charts displayed for this document

Syntax

count = *objDoc*.**ChartCount()**

Arguments

none.

Result

Result	Type	Description
count	short	Returns the number of charts displayed

Remarks

The **ChartCount()** method retrieves the number of charts currently defined and displayed for this document. Returns zero if no charts is defined yet.

Example

```
count = objDoc.ChartCount()
```

See also

Class Chart, ChartCreate Method

7.6.2.2 Document::ChartCreate

Creates a new charts and returns an *Chart* object to it.

Syntax

objChart = *objDoc*.**ChartCreate(**pos,srcchart**)**

Arguments

Argument	Type	Description
pos	string	The display position of the chart
srcchart	object	The contents of the source chart is copied if *srcchart* is not `Nothing`

Result

Result	Type	Description
objChart	Object	Returns an IDispatch object to the new chart or an invalid object

Remarks

The **ChartCreate()** method creates a new data display chart in the documents window.

The chart is inserted in the list of charts at position specified in the argument. If the position is already occupied by another chart the old chart is shifted to the next higher position. If the new position is higher than the last position it is replaced by the next highest position. If the position is negative the chart is placed at the end of the list. More information about the charts position reffer to Chart.Pos Property.

If the second argument *srcchart* is not `Nothing` the source charts contents is copied.

Example

```
' create a new chart at the top left corner of the window
Set objChart = objDoc.ChartCreate(0,Nothing)

' Create a Copy of the selected chart and append it
Set objSrc   = objDoc.ChartGetActive()
Set objChart = objDoc.ChartCreate(-1,objSrc)
```

See also

Class Chart

7.6.2.3 Document::ChartDeleteAll

Removes all charts of the document

Syntax

done = *objDoc*.**ChartDeleteAll()**

Arguments

None.

Result

Result	Type	Description
done	Boolean	Returns `True` if all charts could be removed otherwise `False`

Remarks

The **ChartDeleteAll()** method removes all charts of the document.

Example

```
' close all charts of active document
Set objDoc = objApp.DocGetActive()
If objApp.IsObj(objDoc) Then
  objDoc.ChartDeleteAll
End If
```

See also

Class Chart

7.6.2.4 Document::ChartDeleteByPos

Deletes the n'th chart

Syntax

done = *objDoc*.**ChartDeleteByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Removes the chart at specified position

Result

Result	Type	Description
done	Boolean	Returns `True` if the chart could be deleted otherwise `False`

Remarks

The **ChartDeleteByPos()** method deletes the chart with position *pos*.
The argument has to be positiv and lower than the value return by **ChartCount()**.

Example

```
' close last chart
objDoc.ChartDeleteByPos(objDoc.ChartCount() - 1)

' close active chart
Set objChart = objDoc.ChartGetActive()
objDoc.ChartDeleteByPos(objChart.Pos)
```

See also

Class Chart, ChartCount Method, Chart.Pos Property

7.6.2.5 Document::ChartGetActive

Returns a *Chart* class object associated with the current active chart.

Syntax

objChart = *objDoc*.**ChartGetActive()**

Arguments

none

Result

Result	Type	Description
objChart	Object	Returns a IDispatch object to the chart object which is active or an invalid object reference if no active chart is available.

Remarks

The **ChartGetActive()** method returns a IDispatch object to the active chart. If no chart is selected an invalid object is returned. This can be checked by **objApp.IsObj()**.

Example

```
' get access to the current chart
Set objChart = objDoc.ChartGetActive()
If Not objApp.IsObj(objChart) Then
  MsgBox "No chart selected"
End If
```

See also

Class Chart, **Chart.Active Property**

7.6.2.6 Document::ChartGetByPos

Returns a *Chart* class object at the specified position.

Syntax

objChart = *objDoc*.**ChartGetByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	chart position number

Result

Result	Type	Description
objChart	Object	Returns a IDispatch object for the chart at the given position or an invalid object if *pos* >= **ChartCount**()

Remarks

The **ChartGetByPos** method returns a IDispatch object to the chart at a specified position. If position is out of range an invalid object is returned. This can be checked by **objApp.IsObj()**.

The position is the index into an list which keeps track of all charts of a document. It represents the n'th chart counted from top to down and left to right in the document window.

Example

```
' get name of signal displayed in the first chart
Set objChart = objDoc.ChartGetByPos(0)
If objApp.IsObj(objInfo) Then
  Set objData = objDoc.DataGetByPos(objChart.Group,objChart.Signal)
  MsgBox "First chart displayes signal = " & objData.AxisSignalName
Else
  MsgBox "No Chart available"
End If
```

See also

Class Chart,

7.6.2.7 Document::DataCreate

Creates a new data container object and returns a reference to it.

Syntax

objData = *objDoc*.**DataCreate(**group,signal,srcinfo**)**

Arguments

Argument	Type	Description
group	short	Index of the group. If group does not exists it is created.Use -1 to create the data container in a new group with automaticaly choosen free index.
signal	short	Number of the signal channel. If signal is not existing ist is created. Use -1 to create the data container with automaticaly choosen free signal number.
srcdata	object	A reference to a source data container to copy its contents into the new one or `Nothing` .

Result

Result	Type	Description
objData	Object	Returns an IDispatch object to the new data container or an invalid object

Remarks

The **DataCreate** method creates a new data container object. A Data container stores the values of a signal as a result of a measurement or a calculation. Tha Data containers which are measured synchronously are stored in a group (e.g Group "Scan Forward" with two data container for signal "Topography" and "Phase").

Multiple groups of data containers can be stored in a document (e.g A document contains group "Scan Forward", "Spectroscopy Forward" and "Spectroscopy Backward", another document just contains the group "Cross section").

To place a Data Container in a document one have to define its group index and its signal number. If the Imaging or Spectroscopy Modul created the document two of the signal numbers have a fix assosiation with the measurement channels.

0 - Z-Feedback Error input signal
1 - Topography Signal

Note:
Individual signal should be referenced for future compatibility reason by their signal names as much as possible. Use the signal number for loops through all signals or as result of **DataGetSignalPos()**. Also the group indexes should only be used with loops or as result of **DataGetGroupPos()**.

DataCreate() cannot overwriting an existing data container and returns a invalid reference if a data container at the arguments position is allredy defined.

If one just need a new group to place a result of a calculation one can use -1 as a group index for the *group* and or the *signal* argument.

If a new created data container should be preprared with existing data values set the argument *srcdata* to a valid source data object.

Example

```
' create a new channel in a new group and call the allocated group 'Result'
Set objData = objDoc.DataCreate(-1,-1,Nothing)
objDoc.SetGroupName objData.GetGroupPos(),"Result"

' Copy the selected data into a new data container of a new document
Set objSrcData  = objSrcDoc.DataGetActive()
If objApp.IsObj(objSrcData) Then
  Set objDestDoc = objApp.DocCreate("")
  Set objDestData = objDestDoc.DataCreate(-
1,objSrcData.GetSignalPos(),objSrcData)
End If
```

See also

Class Data, DataGetGroupPos Method, DataGetSignalPos Method, Application.IsObj Method

7.6.2.8 Document::DataDeleteAll

Deletes all data containers of a document.

Syntax

ok = *objDoc*.**DataDeleteAll()**

Arguments

none

Result

Result	Type	Description
ok	boolean	`True` if all groups could be deleted.

Remarks

The **DataDeleteAll()** method deletes all data containers and all groups within a document.
If deletion could not be done `False` is returned.

Example

```
' Empty a document from all data
ok = objDoc.DataDeleteAll()
```

See also

Class Data, DataDeleteByName Method, DataDeleteAll Method, DataDeleteGroup Method, DataDeleteAll Method

7.6.2.9 Document::DataDeleteByName

Deletes a data container.

Syntax

ok = *objDoc*.**DataDeleteByName(**groupname, signalname**)**

Arguments

Argument	Type	Description
groupname	string	name of group
signalname	string	name of signal

Result

Result	Type	Description
ok	boolean	`True` if data container could be deleted.

Remarks

The **DataDeleteByName()** method deletes a data container with specified group name and signal name.
If the data container does not exists `False` is returned.

Example

```
' Delete a specific data container
ok = objDoc.DataDeleteByName("Cross Section","Phase")
```

See also

Class Data, DataDeleteByPos Method, DataDeleteGroup Method, DataDeleteAll Method

7.6.2.10 Document::DataDeleteByPos

Deletes a data container.

Syntax

ok = *objDoc*.**DataDeleteByPos(**group, signal**)**

Arguments

Argument	Type	Description
group	short	index of group.
signal	short	signal number

Result

Result	Type	Description
ok	boolean	`True` if data container could be deleted.

Remarks

The **DataDeleteByPos()** method deletes a data container with specified group index and signal number.
If the data container does not exists `False` is returned.

Example

```
' Delete current data container
Set objData = objDoc.DataGetActive()
If objApp.IsObj(objData) Then
  ok = objDoc.DataDeleteByPos(objData.GetGroupPos, objData.GetSignalPos)
End If
```

See also

Class Data, DataDeleteByName Method, DataDeleteGroup Method, DataDeleteAll Method

7.6.2.11 Document::DataDeleteGroup

Deletes a group of data containers.

Syntax

ok = *objDoc*.**DataDeleteGroup(**group**)**

Arguments

Argument	Type	Description
group	short	index of group.

Result

Result	Type	Description
ok	boolean	`True` if group could be deleted.

Remarks

The **DataDeleteGroup()** method deletes all data containers within a specified group and the group itself.
If the group with groupindex does not exists `False` is returned.

Example

```
' Delete backward spectroscopy
ok = objDoc.DataDeleteGroup(objDoc.GetGroupPos("Specroscopy Backward"))
```

See also

Class Data, DataDeleteByName Method, DataDeleteGroup Method, DataDeleteAll Method

7.6.2.12 Document::DataGetActive

Returns a *Data* class object associated with the current active chart.

Syntax

objData = *objDoc*.**DataGetActive()**

Arguments

none

Result

Result	Type	Description
objData	Object	Returns a IDispatch object to the data object which is displayed by the active chart or an invalid object if no active chart is available.

Remarks

The **DataGetActive()** method returns a IDispatch object to the data container which is displayed by the active chart. If no chart is selected an invalid object is returned. This can be checked by **objApp.IsObj()**.

Example

```
' get access to the current data
Set objData = objDoc.DataGetActive()
If Not objApp.IsObj(objData) Then
  MsgBox "No chart selected"
End If
```

See also

Class Data, Class Chart, **Chart.Active Property**

7.6.2.13 Document::DataGetByName

Returns a *Data* class object with specified name.

Syntax

objData = *objDoc*.**DataGetByName(**groupname, signalname**)**

Arguments

Argument	Type	Description

group	string	name of group.
signal	string	name of signal

Result

Result	Type	Description
objData	Object	Returns a IDispatch object for the data object with given name. If no data container is found an invalid object is returned.

Remarks

The **DataGetByName()** method returns a IDispatch object to the data container with spec ivied names. If no container is found an invalid object is returned. This can be checked by **objApp.IsObj()**.

Example

```
' get access to topography of forward scan
Set objData = objDoc.DataGetByName("Spectroscopy Forward","Deflection")
If Not objApp.IsObj(objData) Then
  MsgBox "No Image available"
End If
```

See also

Class Data

7.6.2.14 Document::DataGetByPos

Returns a *Data* class object at the specified position.

Syntax

objData = *objDoc*.**DataGetByPos(**group, signal**)**

Arguments

Argument	Type	Description
group	short	index of group.
signal	short	number of signal in group

Result

Result	Type	Description
objData	Object	Returns a IDispatch object for the data object at selected position or an invalid object if no data object is at selected position.

Remarks

The **DataGetByPos()** method returns a IDispatch object to the data container at a specified position. If position is out of range an invalid object is returned. This can be checked by **objApp.IsObj()**.

The *group* index is a zero based index number. The index have to be less than **DataGroupCount().**
The *signal* number is a zero based number of the channel. The number of to be less than **DataChannelCount()**.

Example

```
' get access to topography of forward scan
Set objData = objDoc.DataGetByPos(objDoc.GetGroupPos("Scan Forward"),1)
If Not objApp.IsObj(objData) Then
  MsgBox "No Image available"
End If
```

See also

Class Data, DataGroupCount Method, DataSignalCount Method, DataGetGroupPos Method

7.6.2.15 Document::DataGetGroupID

Gets the ID value of a group.

Syntax

id = *objDoc*.**DataGetGroupID(**group**)**

Arguments

Argument	Type	Description
group	short	index of group.

Result

Result	Type	Description
id	short	ID of group or -1 if group not found

Remarks

The **DataGetGroupID()** method return the ID number of a group. If the group is not defined a value of -1 is returned.

Example

```
' delete all phase channels of a spectroscopy in a document
SpecID = 1
For g = 0 To objDoc.DataGroupCount()-1
  If objDoc.GetGroupID(g) = SpecID Then
    ok = objDoc.DataDeleteByPos(g,objDoc.DataGetSignalPos(g,"Phase"))
  End If
Next
```

See also

Class Data, DataSetGroupID Method

7.6.2.16 Document::DataGetGroupName

Returns the name of a group.

Syntax

groupname = *objDoc*.**DataGetGroupName(**group**)**

Arguments

Argument	Type	Description
group	short	index of group.

Result

Result	Type	Description
groupname	string	Name of group or "" if not group index is out of range.

Remarks

The **DataGetGroupName()** method returns the name of a group. If the group with the given index is not defined an empty string is returned.

Example

```
' Display a list of all groups
groupnames = ""
For i = 0 To objDoc.GetGroupCount()-1
  groupnames = groupnames & vbCRLF & objDoc.DataGetGroupName(i)
Next
MsgBox "Available Groups in Document:" & groupnames
```

See also

Class Data, DataGroupCount Method

7.6.2.17 Document::DataGetGroupPos

Returns the group index of a specified group name.

Syntax

index = *objDoc*.**DataGetGroupPos(**groupname**)**

Arguments

Argument	Type	Description
groupname	string	name of group

Result

Result	Type	Description
index	short	index number of the group.

Remarks

The **DataGetGroupPos()** method returns the index number into the list of defined groups for the group with specified name. If no group is found a value of -1 is returned.

To get a specific group it is recommended to get its index by this method because the group index of a certain group can vary from document to document. (e.g: "Scan Backward" group can have index 0 or 1 depending on the measurement mode during imaging)

Example

```
' search for topography of backward scan
```

```
scanpos = objDoc.DataGetGroupPos("Scan Backward")
If scanpos > 0 Then
  Set objData = objDoc.DataGetByPos(scanpos,1)
End If
```

See also

Class Data, DataGetByPos Method

7.6.2.18 Document::DataGetSignalPos

Returns the signal number of a specified signal name.

Syntax

pos = *objDoc*.**DataGetSignalPos(**group, signalname**)**

Arguments

Argument	Type	Description
group	short	index of group.
signalname	string	name of signal

Result

Result	Type	Description
pos	short	position of the signal in the selected group.

Remarks

The **DataGetSignalPos()** method returns the number of the signal with the given name. If no signal is found a value of -1 is returned.

Example

```
' search for tip current signal number
pos = objDoc.DataGetSignalPos(0,"Tip Current")
If pos < 0 Then
  MsgBox "No tip current data available"
End If
```

See also

Class Data

7.6.2.19 Document::DataGroupCount

Retrieves the number of data groups in this document

Syntax

count = *objDoc*.**DataGroupCount()**

Arguments

none.

Result

Result	Type	Description
count	short	Returns the number of data groups

Remarks

The **DataGroupCount** method retrieves the number of data groups available in this document. Returns zero if no group is defined.
The *Data* objects of synchronous measured signals are stored in a group. The groups are sequentially numbered from `zero` to `count-1`.

Example

```
count = objDoc.DataGroupCount()
```

See also

Class Data, DataCreate Method, DataGetByPos Method, DataDeleteByPos Method

7.6.2.20 Document::DataSetGroupID

Sets the ID value of a group.

Syntax

ok = *objDoc*.**DataSetGroupID(**group, groupid**)**

Arguments

Argument	Type	Description
group	short	index of group.
groupid	short	id number of group

Result

Result	Type	Description
ok	boolean	`True` if ID could be changed.

Remarks

The **DataSetGroupID()** method set the ID number of group.

ID numbers a used to identify groups which contains data of the same style. (e.g Scan Forward, and Scan Backward groups contains similar data, also Spectroscopy forward and Spectroscopy Backward). The ID number of such groups can be set to an equal number. The software or a script can then process data containers of a groups together if desired.

The "Current Line" arrow of "Color Map" charts is using this feature to change the current line of all signal channels in all groups with the same group id number if the user drag the arrow up and down.

Each new created group by DataCreate() gets a new group id in the range 256 to 32767.

It is recommend to used ID number in the range from 128 to 255 for user defined ID and overwrite only dynamically defined ID but not standard IDs set by the main applications modules.

Predefined group IDs are in the range 0 to 127. The following are defined:

Group ID	Description
0	Scan group ID. Data groups created by the imaging module.
1	Spectroscopy group ID. Data groups created by the spectroscopy module.

Example

```
' Create two new data container of individual groups
' and mark them with the same user defined group ID
Set objDataOne = objDoc.DataCreate(-1,-1,Nothing)
Set objDataTwo = objDoc.DataCreate(-1,-1,Nothing)
If objApp.IsObj(objDataOne) Then
  objDoc.DataSetGroupID(objDataOne.GetGroupPos,127)
End If
If objApp.IsObj(objDataTwo) Then
  objDoc.DataSetGroupID(objDataTwo.GetGroupPos,127)
```

```
End If
```

See also

Class Data, DataCreate Method

7.6.2.21 Document::DataSetGroupName

Sets the name of a group.

Syntax

ok = *objDoc*.**DataSetGroupName(**group, groupname**)**

Arguments

Argument	Type	Description
group	short	index of group.
groupname	string	name of group

Result

Result	Type	Description
ok	boolean	`True` if name for specified group could be set

Remarks

The **DataSetGroupName()** method set the name of group.

Use this function to give a group created by **DataCreate()** a nice name. It is not recommended to overwrite group names generated be the imaging or spectroscopy modul.

Example

```
' Create a new data container in a new group and name the group
Set objData = objDoc.DataCreate(-1,-1,Nothing)
If objApp.IsObj(objData) Then
  objDoc.DataSetGroupName(objData.GetGroupPos,"My Analysis")
End If
```

See also

Class Data, DataCreate Method

7.6.2.22 Document::DataSignalCount

Retrieves the maximal number of signal channels stored in a group

Syntax

count = *objDoc*.**DataSignalCount(**group**)**

Arguments

Argument	Type	Description
group	short	position index for group of interest

Result

Result	Type	Description
count	short	Returns the number of signals in the specified group

Remarks

The **DataSignalCount** method retrieves the number of signal channels available in a group. Returns zero if no channels are available.
The *Data* objects of synchronous measured signals are stored in the same group.

Not all of the available signal channels of a group have to be measured. If referenced by DataGetByPos() an undefined *Data* object is returned if the position of the signal channel is between zero and count-1 but contains no data.

Example

```
' get the amount of real measured signal channels
count    = objDoc.DataSignalCount(0)
measured = 0
For pos = 0 To count-1
  If objApp.IsObj(objDoc.DataGetByPos(pos)) Then
    measured = measured + 1
  End If
Next
MsgBox "Available signal channels: " & measured
```

See also

Class Data, DataGroupCount Method, DataCreate Method, DataGetByPos Method, DataDeleteByPos Method

7.6.2.23 Document::InfoCount

Retrieves the number of information sections of this document

Syntax

count = *objDoc*.**InfoCount()**

Arguments

none.

Result

Result	Type	Description
count	short	Returns the number of information sections

Remarks

The **InfoCount** method retrieves the number of information sections of this document. Returns zero if no section is created.

Example

```
count = objDoc.InfoCount()
```

See also

Class Info, InfoCreate Method, InfoGetByPos Method, InfoDeleteByPos Method

7.6.2.24 Document::InfoCreate

Creates a new information section and returns an *Info* object to it.

Syntax

objInfo = *objDoc*.**InfoCreate(**sectioname,pos,srcinfo**)**

Arguments

Argument	Type	Description
sectionname	string	The titel of the new section
pos	short	The position in the list of information section
srcinfo	object	The contents of the source info is copied if *srcinfo* is not `Nothing`

Result

Result	Type	Description
objInfo	Object	Returns an IDispatch object to the new information section or an invalid object

Remarks

The **InfoCreate** method creates a new information section in the documents list of informations.

The titel of the new section is provided with the argument *sectionname*.
The *pos* argument defines the initilal display position of the new section. A value of -1 palces the section to the end of the list.
If the argument *srcinfo* is not `Nothing` its contents is copied.

Example

```
' create a new empty section
Set objInfo = objDoc.InfoCreate("Analysis Results",0,Nothing)

' Copy the contents of the 'Scan' Section from one document to another
Set objSrcInfo  = objSrcDoc.InfoGetByName("Scan")
Set objDestInfo = objDestDoc.InfoCreate("Scan",0,objSrcInfo)
```

See also

[Class Info]

7.6.2.25 Document::InfoDeleteAll

Removes all information section of a document

Syntax

done = *objDoc*.**InfoDeleteAll()**

Arguments

None.

Result

Result	Type	Description
done	Boolean	Returns `True` if all section could be removed otherwise `False`

Remarks

The **InfoDeleteAll** method removes all information section of the document.

Example

```
' empfty information section
ok = objDoc.InfoDeleteAll()
If objDoc.InfoCount() > 0 Then
  MsgBox "Error: Could not remove all information sections!"
End If
```

See also

Class Info, InfoCount Method

7.6.2.26 Document::InfoDeleteByName

Deletes the info section with a specified name

Syntax

done = *objDoc*.**InfoDeleteByName(**name**)**

Arguments

Argument	Type	Description
name	string	Remove the information section from the document with this name

Result

Result	Type	Description
done	Boolean	Returns `True` is section could be found and removed otherwise `False`

Remarks

The **InfoDeleteByName** method removes the information section with the title *name*. The argument has to be a string. If the information section is found this method returns `False`.

Example

```
' remove analysis section
Set oDoc = objApp.DocGetActive()
If objApp.IsObj(oDoc) Then
  objDoc.InfoDeleteByName("Result")
End If
```

See also

Class Info, Name Property

7.6.2.27 Document::InfoDeleteByPos

Deletes the n'th information section

Syntax

done = *objDoc*.**InfoDeleteByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Removes the information section at specified position

Result

Result	Type	Description
done	Boolean	Returns `True` if the section could be deleted otherwise `False`

Remarks

The **InfoDeleteByPos** method deletes the information section with position *pos*. The argument has to be positiv and lower than the value return by **InfoCount()**.

Example

```
' close last document
objDoc.InfoDeleteByPos(objDoc.InfoCount() - 1)
```

See also

Class Info, InfoCount Method

7.6.2.28 Document::InfoGetByName

Returns a *Info* class object with the specified name.

Syntax

objInfo = *objDoc*.**InfoGetByName(**name**)**

Arguments

Argument	Type	Description
name	string	Name of information section

Result

Result	Type	Description
objInfo	Object	Returns a IDispatch object to the information section with the specified name or an invalid object if no section is not found

Remarks

The **InfoGetByName** method returns a IDispatch object to the information section with the specified name in the argument.
If no section with *name* is found a invalid object is returned. This can be checked by **objApp.IsObj()**.

The name of a section is its title displayed in the Data Info Panel.

Example

```
Set objInfo = objDoc.InfoGetByName("Scan")
If Not objApp.IsObj(objDoc) Then
  MsgBox "No Section called 'Scan' found"
```

```
End If
```

See also

Class Info

7.6.2.29 Document::InfoGetByPos

Returns a *Info* class object at the specified position.

Syntax

objInfo = *objDoc*.**InfoGetByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Section position number.

Result

Result	Type	Description
objInfo	Object	Returns a IDispatch object for the info section at position *pos* or an invalid object if *pos* >= **InfoCount**()

Remarks

The **InfoGetByPos** method returns a IDispatch object to the information section at a specified position. If position is out of range an invalid object is returned. This can be checked by **objApp.IsObj()**.

The position is the index into an list which keeps track of all information section of a document. It represents the n'th section as shown in the Data Info Panel.

Example

```
Set objInfo = objDoc.InfoGetByPos(0)
If objApp.IsObj(objInfo) Then
  MsgBox "First section is = " & objInfo.Name
End If
```

See also

Class Info, InfoCount Method, InfoGetByName Method

7.6.2.30 Document::Load

Loads the contents of an nid-File into the document.

Syntax

ok = *objDoc*.**Load(**filename**)**

Arguments

Argument	Type	Description
filename	string	Filename of a document or an empty string (`""`) to open a file open dialog

Result

Result	Type	Description
ok	Boolean	Returns `True` if successful

Remarks

The **Load** method loads the file with the path in *filename*. If *filename* is an empty string a file open dialog is displayed.

If the user click abort or the file could not be loaded the method return `False`.

Example

```
If objDoc.Load("") Then
  MsgBox "File" & objDoc.Name & "is loaded"
End If
```

See also

Name Property, Save Method

7.6.2.31 Document::Save

Save the document content to an nid-File.

Syntax

ok = *objDoc*.**Save(**filename**)**

Arguments

Argument	Type	Description
filename	string	Filename of a document or an empty string ("") to open a file save dialog

Result

Result	Type	Description
ok	Boolean	Returns `True` if successful

Remarks

The **Save** method stores document object to a *filename*. If *filename* is an empty string a file save dialog is displayed.

If the user click abort or the file could not be saved the method return `False`.

Example

```
If objDoc.Save("MyDocument.nid") Then
  MsgBox "Document saved to " & objDoc.Name
End If
```

See also

Name Property, Load Method

7.6.2.32 Document::ShowWindow

Defines the display style of the document window.

Syntax

objDoc.**ShowWindow(**style**)**

Arguments

Argument	Type	Description
style	short	Visibility style number

Result

None.

Remarks

The **ShowWindow** method sets the visibility state of the window. Use one of the following values:

Name	Value	Description
SW_HIDE	0	Hides this window and passes activation to another window
SW_NORMAL	1	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position
SW_MINIMIZED	2	Activates the window and displays it as an icon
SW_MAXIMIZED	3	Activates the window and displays it as a maximized window
SW_SHOWNOACTIVE	4	Displays the window in its most recent size and position. The window that is currently active remains active
SW_ACTIVATE	5	Activates the window and displays it in its current size and position
SW_MINIMIZE	6	Minimizes the window and activates the top-level window in the system's list.
SW_MINNOACTIVE	7	Displays the window as an icon. The window that is currently active remains active
SW_SHOWNA	8	Displays the window in its current state. The window that is currently active remains active
SW_RESTORE	9	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position

Example

```
objDoc.ShowWindow(3) ' maximize and activate this document
```

See also

None.

7.7 Info

The Info class is a container which stores a set of values along the measured data in a document. These information are displayed in the DataInfo Panel.

A document can store multiple Info classes in a list. To identify individual members each instant has a name and a position in the list.

The values in a Info class are stored as name and value pairs. To reference a value one can use its name or position in the container. The position in the container also defines the display order in the DataInfo Panel.

The application is using these Info classes to store measurement parameters like Feedback settings or scan head calibration information. Predefined Info class names are the following:

Section names	Purpose
Global	Version numbers and calibration information
Feedback	Z-Feedback controller parameters like set point
Scan	Contains scan parameters like scan speed
Spec	Spectroscopy parameters
Tool	Active Tools result
Result	Results of operation point adjustment in dynamic force mode
Modules	Information about installed Modules

The user is free to define new Info sections for their own purpose (e.g: Store analysis results of a script function or sample preparation information)

Table of properties for Info class:

Property name	Purpose

Name	Contains name or title of the information section
Pos	Position in the list of info class of the document

Table of methods for general usage of Document class:

Method name	Purpose
GetDocument	Returns the IDispatch class of the parent document
Count	Returns the number of values stored in this class
SetByName	Set a value with a specified name
GetByName	Get a value with a specified name
SetByPos	Set a value with a specified position
GetByPos	Get a value with a specified position
GetNameByPos	Get the name of a value at a specified position
DeleteByName	Delete a value with a specified name
DeleteByPos	Delete a value with a specified position
DeleteAll	Deletes all name value pairs

7.7.1 Properties

7.7.1.1 Info::Name

Returns or sets the name of the info section.

Syntax

objInfo.**Name** [= name]

Setting

Argument	Type	Description
name	String	*name* is a string containing the new name of the section

Remarks

The **Name** property is containing the name of the info section. It is unique or one document.
It is displayed in the Data Info Panel as a title on to of the values.

The name of a newly created info class is assigned by the *objDoc*.**InfoCreate**()

method.

Example

```
Dim objInfo : Set objInfo = objDoc.InfoCreate("Test",Nothing)
MsgBox "Name is " & objInfo.Name
```

See also

Doc.InfoCreate, Pos Property

7.7.1.2 Info::Pos

Returns or sets the position of the info section in the document.

Syntax

objInfo.**Pos** [= pos]

Setting

Argument	Type	Description
pos	short	*pos* defines the position of the section in the list of a document

Remarks

Info class instances are stored in the parent document in a list. The **Pos** property is containing the list position of a info section. Info sections are displayed in the Data Info Panel in their list position starting by position zero.

If the value -1 is assigned to the Pos property the info class is placed at the end of the list and the Pos property value is set accordingly.

Example

```
' move a info section to the end
objInfo.Pos = -1
```

See also

Doc.InfoCreate

7.7.2 Methods

7.7.2.1 Info::Count

Retrieves the number of values stored in this section

Syntax

count = *objInfo*.**Count()**

Arguments

none.

Result

Result	Type	Description
count	short	Returns the number of stored values

Remarks

The **Count()** method retrieves the number of values currently defined and displayed for this information section. Returns zero if no values are defined.

Example

```
count = objInfo.Count()
```

See also

Class Info, SetByName Method, SetByPos Method

7.7.2.2 Info::DeleteAll

Deletes all information of a section.

Syntax

ok = *objInfo*.**DeleteAll()**

Arguments

none.

Result

Result	Type	Description
ok	boolean	`True` if all information could be deleted

Remarks

The **DeleteAll()** method deletes all information entries of the info section.

Example

```
' delete all
ok = objInfo.DeleteAll()
```

See also

Class Info, DeleteByName Method, DeleteByPos Method

7.7.2.3 Info::DeleteByName

Deletes the information with a given name.

Syntax

ok = *objInfo*.**DeleteByName(**name**)**

Arguments

Argument	Type	Description
name	string	Name of the information to be deleted

Result

Result	Type	Description
ok	boolean	`True` if value for specified name could be deleted

Remarks

The **DeleteByName()** method deletes a information entry defined by its name. The method searchs for the information in a none case sensitive manner.

Example

```
' delete an value from the roughness analysis result
Set objInfo = objDoc.InfoGetByName("Area Roughness")
```

```
If objApp.IsObj(objInfo) Then
  objInfo.DeleteByName "Sm"
End If
```

See also

Class Info, DeleteByPos Method, DeleteAll Method

7.7.2.4 Info::DeleteByPos

Deletes the information at a given position.

Syntax

ok = *objInfo*.**DeleteByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	Position of the information to be deleted

Result

Result	Type	Description
ok	boolean	`True` if value for specified position could be deleted

Remarks

The **DeleteByPos()** method deletes an information entry defined by its position.

Example

```
' delete first entry in a info section
ok = objInfo.DeleteByPos(0)
```

See also

Class Info, DeleteByName Method, DeleteAll Method

7.7.2.5 Info::GetByName

Returns the value of a information with a given name.

Syntax

value = *objInfo*.**GetByName(**name**)**

Arguments

Argument	Type	Description
name	string	Name of the value

Result

Result	Type	Description
value	string	Stored value for the named argument or an empty string if not found

Remarks

The **GetByName()** method retrieves a value for a specified information defined by its name.

The name is not case sensitive. If no information is found an empty string is returned

Example

```
' Create a new info section and store some value
Set objInfo = objDoc.InfoGetByName("Scan")
If objApp.IsObj(objInfo) Then
  MsgBox "Used scan speed was = " & objInfo.GetByName "Time/Line"
End If
```

See also

Class Info, GetByPos Method

7.7.2.6 Info::GetByPos

Returns the value of a information at a given position.

Syntax

value = *objInfo*.**GetByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	position of the value

Result

Result	Type	Description
value	string	Stored value at the specified position by *pos*. Is an empty string if position is out of range.

Remarks

The **GetByPos()** method retrieves a value for a specified information defined by its position.

If no information is found an empty string is returned

Example

```
' list all information for section "Scan"
Set objInfo = objDoc.InfoGetByName("Scan")
If objApp.IsObj(objInfo) Then
  For i = 0 To objInfo.Count() - 1
    MsgBox objInfo.GetNameByPos(i) & " = " & objInfo.GetByPos(i)
  Next
End If
```

See also

Class Info, GetByName Method

7.7.2.7 Info::GetDocument

Returns a IDispatch object to the parent Document class.

Syntax

objDoc = *objInfo*.**GetDocument()**

Arguments

none.

Result

Result	Type	Description
objDoc	Object	A IDispatch object to the parent document class

Remarks

The **GetDocument()** method returns a IDispatch object to the Document class where this class is stored.

Example

```
Set objDoc = objInfo.GetDocument()
if objApp.IsObj(objDoc) then
  MsgBox "objInfo is stored in : " & objDoc.Name
end if
```

See also

Class Info, Class Document

7.7.2.8 Info::GetNameByPos

Returns the name of a information at a given position.

Syntax

name = *objInfo*.**GetNameByPos(**pos**)**

Arguments

Argument	Type	Description
pos	short	position of the value

Result

Result	Type	Description
name	string	Name of value at the specified position by *pos*. Is an empty string if position is out of range.

Remarks

The **GetNameByPos()** method retrieves the name for a specified information defined by its position.

If no information is found an empty string is returned

Example

```
' list all information for section "Scan"
Set objInfo = objDoc.InfoGetByName("Scan")
If objApp.IsObj(objInfo) Then
  For i = 0 To objInfo.Count() - 1
    MsgBox objInfo.GetNameByPos(i) & " = " & objInfo.GetByPos(i)
  Next
End If
```

See also

Class Info

7.7.2.9 Info::SetByName

Sets the value of a information with a given name.

Syntax

ok = *objInfo*.**SetByName(**name, value**)**

Arguments

Argument	Type	Description
name	string	Name of the value
value	string	New value to be set

Result

Result	Type	Description
ok	boolean	True if value for specified name could be set

Remarks

The **SetByName()** method sets a new value for a specified information defined by its name

If the name is not already defined a new entry in the list of information is created at the end of the list. The name is not case sensitive but stored as it is defined.

Example

```
' Create a new info section and store some value
Set objInfo = objDoc.InfoCreate("My Analysis",-1,Nothing)
If objApp.IsObj(objInfo) Then
  objInfo.SetByName "Algo","SuperCalc"
  objInfo.SetByName "Result",1.2234
End If
```

See also

Class Info, GetByName Method, SetByPos Method

7.7.2.10 Info::SetByPos

Sets the value of a information at a given position

Syntax

ok = *objInfo*.**SetByPos(**pos, value**)**

Arguments

Argument	Type	Description
pos	short	position of the value
value	string	New value to be set

Result

Result	Type	Description

ok	boolean	`True` if value at specified position could be set

Remarks

The **SetByPos()** method sets a new value for a specified information defined by its position. The position has to be positiv and lower than the value returned by **Count()** otherwith the method return `False`.

It is recommended to use this function only to overwrite values predefined by **SetByName**().

Example

```
' Create a new info section and store some value
Set objInfo = objDoc.InfoCreate("My Analysis",-1,Nothing)
If objApp.IsObj(objInfo) Then
  objInfo.SetByName "Algo","SuperCalc"
  objInfo.SetByName "Result",0

  ' overwrite Result value (pos = 1)
  objInfo.SetByPos(1,3.1415)
End If
```

See also

Class Info, GetByPos Method, SetByName Method

7.8 Litho

The Litho class handles the microscope's lithography subsystem.

A object pointer to this class is provided by the Application.Litho object property.

A lithography session is assembled offline by adding commands to the command list. Call Start to start the session after completing the command list. Before assembling a new lithography session the ClearCmdList method must be called.

Table of properties for Litho class:

Property name	Purpose
OperationMode	Set the lithography operating mode
PenUpMode	Set the PenUp mode

Table of methods for general usage of Document class:

Method name	Purpose
Start	Start a lithography sequence
Stop	Stop an ongoing lithography sequence
IsMoving	Retrieve the information whether a lithography Cmd is in process or not
IsWorking	Retrieve the information whether a lithography is in process or not
ClearCmdList	Clear the command list
AddCmd_MoveTip	Add a MoveTip command to the CmdList
AddCmd_Wait	Add a Wait command to the CmdList
AddCmd_SetPoint	Add a SetPoint command to the CmdList
AddCmd_TipVoltage	Add a TipVoltage command to the CmdList
AddCmd_VibratingAmpl	Add a VibratingAmpl command to the CmdList
AddCmd_PenUp	Add a PenUp command to the CmdList
AddCmd_PenDown	Add a PenDown command to the CmdList
StartCapture	Prepare a image capture if scanning or do it immediately
StopCapture	Clear a prepared image capture
IsCapturing	Retrieve the information whether a capture is prepared or not
StartFrameUp	Start a single scan frame direction upward
StopFrameUp	Stop a single scan frame
IsScanning	Retrieve the information whether a scanning is in process or not

7.8.1 Properties

7.8.1.1 Litho::OperatingMode

Get or set the lithography operating mode.

Syntax

litho.**OperatingMode** [= mode]

Argument

Parameter	Type	Description
mode	LONG	Defines the operating mode for lithography. See modes in the table below.

Remarks

In order to do lithography you may select one of the following operating modes.

Table of lithography operation mode values and description:

State No.	Name	Description
0	LithoOpMode_User	Undefined
1	LithoOpMode_STM	For STM scan heads use this index
2	LithoOpMode_StaticAFM	AFM only: Static deflection mode
3	LithoOpMode_DynamicAFM	AFM only: Dynamic force mode

See also

None

7.8.1.2 Litho::InactivePenMode

Get or set the inactive pen mode.

Syntax

*litho.***InactivePenMode** [= mode]

Argument

Parameter	Type	Description
mode	LONG	Defines the inactive pen mode. See modes in the table below.

Remarks

Scan lines can be measured differently. This property defines this.

Table of inactive pen mode values and description:

State No.	Name	Description
0	InactivePenMode_LiftTip	Lift tip while moving to the next start position.
1	InactivePenMode_ChangeOpMode	Change back to the scan operating mode while moving to the next start position.

See also

None

7.8.2 Methods

7.8.2.1 Litho::AddCmd_MoveTip

Move the tip from the current position to a destination coordinate.

Syntax

litho.**AddCmd_MoveTip(***x*,*y*,*z***)**

Argument

Parameter	Type	Description
x	double	X-Axis component of the destination position. Unit in meter [m]
y	double	Y-Axis component of the destination position. Unit in meter [m]
z	double	Z-Axis component of the destination position. Unit in meter [m]

Remarks

This method adds a MoveTip command to the command list. The coordinate system of the destination position is the scanner coordinate system. I.e. the position (0,0,0) is the center position of the scanner.

Example

```
' move tip to x=10e-6m, y=10e-6m, z=0m
objLitho.AddCmd_MoveTip 10e-6, 10e-6, 0
' move tip to x=15e-6m, y=20e-6m, z=0m
objLitho.AddCmd_MoveTip 15e-6, 20e-6, 0
```

See also

Method ClearCmdList

7.8.2.2 Litho::AddCmd_PenDown

Add a PenDown command to the command list.

Syntax

litho.**AddCmd_PenDown**

Remarks

This method adds a PenDown command to the command list. The PenDown command switches from the InactivePenMode to the lithography mode.

Example

```
' pen down
objLitho.AddCmd_PenDown
```

See also

Method ClearCmdList
Property InactivePenMode

7.8.2.3 Litho::AddCmd_PenUp

Add a PenUp command to the command list.

Syntax

litho.**AddCmd_PenUp**

Remarks

This method adds a PenUp command to the command list. The PenUp command switches to the InactivePenMode. Use this command to start moving from one position to another without performing lithography.

Example

```
' pen up
objLitho.AddCmd_PenUp
```

See also

Method ClearCmdList
Property InactivePenMode

7.8.2.4 Litho::AddCmd_SetPoint

Add a SetPoint command to the command list.

Syntax

litho.**AddCmd_SetPoint(***setpoint***)**

Argument

Parameter	Type	Description
setpoint	double	Defines the reference value for the sensor signal from the scan head.

Remarks

This method adds a SetPoint command to the command list. This command changes the Set point that is used in lithography mode.

The unit depends on the operating mode selected by property Litho.OperatingMode.

Op. mode	Input Signal	Unit
STM	Tunneling Current	Ampere
Static AFM	Deflection	Newton
Dynamic AFM	Amplitude	Percentage of resonance peak [0 .. 100%]

Example

```
' set setpoint 10uN (static AFM)
```

```
objLitho.AddCmd_SetPoint 10e-6 ' N
```

See also

Method ClearCmdList

7.8.2.5 Litho::AddCmd_TipSpeed

Add a TipSpeed command to the command list.

Syntax

*litho.***AddCmd_TipSpeed(***speed***)**

Argument

Parameter	Type	Description
speed	double	Tip speed. Unit in meter/second [m/s]

Remarks

This method adds a TipSpeed command to the command list. This command changes the Tip speed that is used in lithography mode.

Example

```
objLitho.AddCmd_TipSpeed 4.0e-6 ' m/s
```

See also

Method ClearCmdList

7.8.2.6 Litho::AddCmd_TipVoltage

Add a TipVoltage command to the command list.

Syntax

*litho.***AddCmd_TipVoltage(***voltage***)**

Argument

Parameter	Type	Description
voltage	double	Defines the potential applied to the tip in voltage. Valid range from -10V to +10V.

Remarks

This method adds a TipVoltage command to the command list. This command changes the Tip voltage that is used in lithography mode.

Example

```
objLitho.AddCmd_TipVoltage 1.0 ' V
```

See also

Method ClearCmdList

7.8.2.7 Litho::AddCmd_VibratingAmpl

Add a VibratingAmpl command to the command list.

Syntax

*litho.***AddCmd_VibratingAmpl(***voltage***)**

Argument

Parameter	Type	Description
voltage	double	Defines the free vibrating amplitude in [V].

Remarks

This method adds a VibratingAmpl command to the command list. This command changes the free vibration amplitude that is used in lithography mode.

This command is only affective if the operating mode "Dynamic force" is used.

Example

```
objLitho.AddCmd_VibratingAmpl 1.0 ' V
```

See also

Method ClearCmdList

7.8.2.8 Litho::AddCmd_Wait

Add a Wait command to the command list.

Syntax

*litho.***AddCmd_Wait(***time***)**

Argument

Parameter	Type	Description
time	double	Defines the wait time in seconds.

Remarks

This method adds a Wait command to the command list.

Example

```
objLitho.AddCmd_Wait 2.0 ' s
```

See also

Method ClearCmdList

7.8.2.9 Litho::ClearCmdList

Clear the command list.

Syntax

litho.**ClearCmdList**

Remarks

This method clears the command list.

Use this method before creating a new command list.

7.8.2.10 Litho::IsCapturing

Returns if a capture is pending or not.

Syntax

result = *litho*.**IsCapturing**

Result

Parameter	Type	Description
result	Boolean	Returns True if a capture is pending

Remarks

This method is returning True if a capture is pending.

Example

```
If objLitho.IsCapturing Then
  objLitho.StopCapture
End If
```

See also

Method StartCapture, StopCapture

7.8.2.11 Litho::IsMoving

Returns if a tip move is pending or not.

Syntax

result = *litho*.**IsMoving**

Result

Parameter	Type	Description
result	Boolean	Returns True if the tip is moving

Remarks

This method is returning True if a tip move is pending.

Example

```
If objLitho.IsMoving Then
  objLitho.Stop
End If
```

See also

Method Start, Stop

7.8.2.12 Litho::IsScanning

Returns if a scan is in process or not.

Syntax

result = *litho*.**IsScanning**

Result

Parameter	Type	Description
result	Boolean	Returns True if imaging is in process

Remarks

This method is returning True if a scan is currently running.

Example

```
' measure image
objLitho.StartFrameUp
Do While objLitho.IsScanning : Loop

' copy image date
objLitho.StartCapture
```

See also

Method StartFrameUp

7.8.2.13 Litho::IsWorking

Returns if a lithography session is pending or not.

Syntax

result = *litho*.**IsWorking**

Result

Parameter	Type	Description
result	Boolean	Returns True if a lithography session is pending

Remarks

This method is returning True if a lithography session is pending.

Example

```
If objLitho.IsWorking Then
  objLitho.Stop
End If
```

See also

Method Start, Stop

7.8.2.14 Litho::Start

Start the lithography session.

Syntax

litho.**Start**

Remarks

This method is starting the lithography session. The lithography session ends when the last command has been executed.

The lithography session may be stopped at any time using the method **Stop**.

Example

```
' prepare litho
objLitho.ClearCmdList
objLihto.AddCmd_TipSpeed 10.0e-6
objLitho.AddCmd_MoveTip 1.0, 1.0, 0.0

' start litho
objLitho.Start

' do something else ...

' finish immediately
objLitho.Stop
```

See also

Method Stop

7.8.2.15 Litho::StartCapture

Create a new image document.

Syntax

litho.**StartCapture**

Remarks

This method copies the measured data to a new image document. If a scanning process is running at the time **StartCapture** is called a new image document is created each time a frame is measured.

A pending capture can be canceled with StopCapture. If a capture is pending read method IsCapturing.

Example

```
' start imaging
objLitho.StartFrameUp

' prepare image copy
objLitho.StartCapture

' wait until copy is taken at end of frame
Do While objLitho.IsCapturing : Loop
```

See also

Method StopCapture, IsCapturing
Method Application.SaveDocument

7.8.2.16 Litho::StartFrameUp

Starts a single up frame image.

Syntax

litho.**StartFrameUp**

Remarks

This method is starting a single image starting from the bottom to the top. During the scan process IsScanning is True and if StartCapturing is called during the frame a new document is created after the scan frame is finished. At the end the tip is moved to the center of the image.

The size and other properties of a scan frame should be predefined prior the start but can be changed anytime also during scanning.

Prior to be able to scan a z-approach should be performed successfully.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7

' measure image
objLitho.StartFrameUp
Do While objLitho.IsScanning : Loop

' copy image date
objLitho.StartCapture
```

See also

Method IsScanning

7.8.2.17 Litho::Stop

Stop the lithography session.

Syntax

litho.**Stop**

Remarks

This method stops an ongoing lithography session immediately. The current executed command will be aborted.

A possible pending capture flag is also aborted and no document is created.

Example

```
' start litho
objLitho.Start

' do something else ...

' finish immediately
objLitho.Stop
```

See also

Method Start

7.8.2.18 Litho::StopCapture

Cancel a pending capture

Syntax

litho.**StopCapture**

Remarks

This method cancel a pending capture. If a capture is pending read method IsCapturing.

Example

```
' start imaging
```

```
objScan.StartFrameUp

' prepare image copy
objScan.StartCapture

' do something

If objScan.IsCapturing Then
  objScan.StopCapture
End If
```

See also

Method StartCapture, IsCapturing

7.8.2.19 Litho::StopFrameUp

Stops imaging immediately.

Syntax

litho.**StopFrameUp**

Remarks

This method stops any scan process immediately after the current scan line is finished. The tip is moved to the center of the image.

A possible pending capture flag is also aborted and no document is created.

Example

```
' start scan
objLitho.StartFrameUp

' do something else ...

' finish immediately
objLitho.StopFrameUp
```

See also

Method StartFrameUp

7.9 OperatingMode

The OperatingMode class is responsible for all the different operation modes of a SPM electronics.

For AFM many different operating modes are usable. They differ on how the cantilever deflection signal is preprocessed and interpreted. Switching between modes is as easy as write to the property **OperatingMode**.

Also different type of cantilevers can be used with different mechanical properties as stiffness or resonance frequency. The property **Cantilever** handles the details about them and adjust the internal microscope electronics accordingly.

Most of the mode dependent properties are automatically set. But if desired the **Auto...**

properties can be set to False and with mode specific properties manual settings can be defined. Of course a read to any of these properties returns the automatically or manual set values.

A object pointer to this class is provided by the Application.OperatingMode object property.

Table of general properties for OperatingMode class:

Property name	Purpose
OperatingMode	Defines the active operating mode of the sensor
Cantilever	Defines the type of the mounted cantilever
Measurement Environment	Defines the type of environment for the measurement
FreqSweepSetInfoCount	Returns the number of available frequency sweep buffers
FreqSweepStart	Returns the start frequency of a specified buffer index
FreqSweepEnd	Returns the end frequency of a specified buffer index
FreqSweepStep	Returns the step frequency of a specified buffer index

Table of general methods for OperatingMode class:

Method name	Purpose
GetFreqSweepLine / GetFreqSweepLine2	Retrieve the data of a freq. sweep line. Returns value as String or Variant
GetFreqSweepLineEx / GetFreqSweepLine2Ex	Retrieve the data of a freq. sweep line of a specified buffer index. Returns value as String or Variant

Table of "Dynamic force"-Mode properties and methods:

Property name	Purpose
VibratingAmpl	Defines the amplitude of the cantilever vibration
ReferenceAmpl	Returns the excitation amplitude used to reach the vibration amplitude
VibratingFreq	Defines the excitation frequency of the cantilever
AutoVibratingFreq	Enable automatically adjustment of excitation frequency
ShowFreqSearchChart	Shows or hides the result of excitation frequency search
Method name	
SearchVibratingFreq	Triggers the excitation frequency search manually
IsFreqSearchRunning	Flags if a frequency search is active

FreqSearchResult	Returns the status of the frequency search
CaptureFreqSearchChart	Create a image document of the frequency search bode plot chart

Table of "Phase Contrast"-Mode properties and methods:

Property name	**Purpose**
VibratingAmpl	Defines the amplitude of the cantilever vibration
VibratingFreq	Defines the excitation frequency of the cantilever
AutoVibratingFreq	Enable automatically adjustment of excitation frequency
ReferencePhase	Defines the phase reference for the phase chart
AutoReferencePhase	Enable automatically adjustment of the reference phase
ShowFreqSearchChart	Shows or hides the result of excitation frequency search
Method name	
SearchVibratingFreq	Triggers the excitation frequency search manually
IsFreqSearchRunning	Flags if a frequency search is active
FreqSearchResult	Returns the status of the frequency search
CaptureFreqSearchChart	Create a image document of the frequency search bode plot chart
SearchReferencePhase	Triggers the reference phase search manually
IsPhaseSearchRunning	Flags if a reference phase search is active

Table of "Force Modulation"-Mode properties and methods:

Property name	**Purpose**
ForceModAmpl	Defines the amplitude of the cantilever excitation
ForceModFreq	Defines the frequency of the cantilever excitation
Method name	
SearchVibratingFreq	Triggers the excitation frequency search manually
IsFreqSearchRunning	Flags if a frequency search is active
FreqSearchResult	Returns the status of the frequency search
CaptureFreqSearchChart	Create a image document of the frequency search bode plot chart

Table of advanced properties for "Dynamic force ", "Phase Contrast" and "Force Modulation"-Mode:

Property name	Purpose
FreqSearchStart	Defines the lower frequency of the vibrating frequency search range
FreqSearchEnd	Defines the upper frequency of the vibrating frequency search range
FreqSearchStep	Defines the step resolution of the vibrating frequency search
AutoFreqSearchRange	Flags if the frequency search area is automatically set
PeakAmplReduction	Defines the shift of operating frequency point from peak maximum
PeakUpperSideBand	Flags if the shift is to the upper or lower side of the peak

7.9.1 Properties

7.9.1.1 OperatingMode::AutoFreqSearchRange

Returns or set a flag to define if automatic search range calculation is active or not.

Syntax

opmode.**AutoFreqSearchRange** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if automatic calculation of start, end and step frequency values is enabled.

Remarks

This property defines if the software is calculation the frequency search rage automatically or not. If in auto mode the frequency range is calculated from the active cantilever's typical resonance value. Therefore accurate cantilever selection with property **Cantilever** and it definition is important.

The settings of this property is used in the operating modes "Dynamic force", "Phase Contrast" and "Force Modulation".

The resonance peak search can be executed manually by method **SearchVibratingFreq**.

Example

```
' execute approach with fully automatically adjustment

' activate auto modes and Phase Contrast Mode
```

```
objOpMode.AutoFreqSearchRange = True
objOpMode.AutoVibratingFreq   = True
objOpMode.AutoReferencePhase  = True
objOpMode.OpertingMode = 4
objOpMode.Cantilever   = 2

objOpMode.VibratingAmpl = 0.05 'V
objZCtrl.SetPoint       = 50 '%
objAppr.StartApproach
```

See also

Property OperatingMode, FreqSearchStart, FreqSearchEnd, FreqSearchStep
Method SearchVibratingFreq

7.9.1.2 OperatingMode::AutoReferencePhase

Returns or set a flag to define if automatic reference phase calibration is active or not.

Syntax

opmode.**AutoRefeencePhase** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if automatic recalibration of reference phase is enabled.

Remarks

This property defines if the property **ReferencePhase** is set automatically or not. If in auto mode after an approach a recalibration of the phase measurement is executed and the reference phase is set to the new value.

The setting of this property is used in the operating mode "Phase Contrast".

The calibration of the reference phase can also be started manually by method **SearchReferencePhase**.

Example

```
' see example at method SearchReferencePhase
```

See also

Property OperatingMode, ReferencePhase
Method SearchVibratingFreq

7.9.1.3 OperatingMode::AutoVibratingFreq

Returns or set a flag to define if automatic resonance frequency detection is active or not.

Syntax

opmode.**AutoVibratingFreq** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if automatic set of excitation frequency is enabled.

Remarks

This property defines if the property **VibratingFreq** is set automatically or not. If in auto mode prior to an approach, operating mode change or cantilever exchange a resonance frequency search is executed and the vibration frequency is set to the found peak.

The settings of this property is used in the operating modes "Dynamic force" and "Phase Contrast".

The resonance peak search can also be executed manually by method **SearchVibratingFreq**.

Example

```
' see example at method SearchVibratingFreq
```

See also

Property OperatingMode, VibratingFreq
Method SearchVibratingFreq

7.9.1.4 OperatingMode::ForceModAmpl

Returns or set the excitation amplitude.

Syntax

opmode.**ForceModAmpl** [= ampl]

Setting

Argument	Type	Description
ampl	double	Defines the excitation amplitude of the cantilever in [V].

Remarks

This property sets the amplitude of the excitation signal of the cantilever. The excitation frequency is defined by **ForceModFreq**.

The setting of this property is used in the operating mode "Force Modulation".

See also

Property OperatingMode, ForceModFreq

7.9.1.5 OperatingMode::ForceModFreq

Returns or set the excitation frequency.

Syntax

opmode.**ForceModFreq** [= freq]

Setting

Argument	Type	Description
freq	double	Defines the excitation frequency of the cantilever in [Hz].

Remarks

This property sets the frequency of the excitation signal of the cantilever. The excitation amplitude is defined by **ForceModAmpl**.

The setting of this property is used in the operating mode "Force Modulation".

See also

Property OperatingMode, ForceModAmpl

7.9.1.6 OperatingMode::FreqSearchEnd

Returns or set the end frequency of the frequency peak search range.

Syntax

opmode.**FreqSearchEnd** [= freq]

Setting

Argument	Type	Description
freq	double	Defines the end frequency of the search range in [Hertz].

Remarks

This property sets the end frequency of the search range for a frequency resonance peak. This frequency has to be higher than the **FreqSearchEnd** value. If **AutoFreqSearchRange** is enabled the start and end point of the sweep is automatically calculated from the cantilever's properties.

The setting of this property is used in the operating modes "Dynamic force", "Phase Contrast" and "Force Modulation".

See also

Property OperatingMode, FreqSearchStart, FreqSearchStep, AutoFreqSearchRange
Method SearchVibratingFreq

7.9.1.7 OperatingMode::FreqSearchStart

Returns or set the start frequency of the frequency peak search range.

Syntax

opmode.**FreqSearchStart** [= freq]

Setting

Argument	Type	Description
freq	double	Defines the start frequency of the search range in [Hertz].

Remarks

This property sets the start frequency of the search range for a frequency resonance peak. This frequency has to be lower than the **FreqSearchEnd** value. If

AutoFreqSearchRange is enabled the start and end point of the sweep is automatically calculated from the cantilever's properties.

The setting of this property is used in the operating modes "Dynamic force", "Phase Contrast" and "Force Modulation".

See also

Property OperatingMode, FreqSearchEnd, FreqSearchStep, AutoFreqSearchRange
Method SearchVibratingFreq

7.9.1.8 OperatingMode::FreqSearchStep

Returns or set the frequency increment of the frequency peak search range.

Syntax

opmode.**FreqSearchStep** [= increment]

Setting

Argument	Type	Description
increment	double	Defines the increment frequency in [Hertz].

Remarks

This property sets the frequency step used by the search for a frequency resonance peak. The frequency step is used for the coarse frequency search only. Increasing the step with a faster sweep is performed but for high Q-Factor cantilever this can end up with bad detection of the peak. The number of amplitude and phase measurements per sweep can be calculated:

*Datapoints = (**FreqSearchEnd** - **FreqSearchStart**) / **FreqSearchStep** + 1*

The setting of this property is used in the operating modes "Dynamic force", "Phase Contrast" and "Force Modulation".

See also

Property OperatingMode, FreqSearchStart, FreqSearchEnd, AutoFreqSearchRange
Method SearchVibratingFreq

7.9.1.13 OperatingMode::LeverExcitationMode

Defines the configuration of the cantilever tip signal.

Syntax

mode = *opmode*.**LeverExcitationMode**

Result

Argument	Type	Description
mode	long	Defines the mode of operation for the cantilever shaking piezo

Remarks

This property defines the configuration of the excitation signal to the shaking piezo of the cantilever.
The excitation can be applied internally by the controller or externally by a user defined source.

Table of available mode values:

State No.	Name	Description
0	`LeverMode_InternalSource`	Excitation signal to the shaking piezo is driven by the internal oscillator of the controller.
1	`LeverMode_ExternalSource`	Excitation signal to the shaking piezo is driven by a external source connected to the easyScan 2 Signal Access Module Advanced BNC "Excitation input".

See also

None

Version info

Software v1.5.1.1 or later

7.9.1.14 OperatingMode::OperatingMode

Returns or set the sensor operating mode.

Syntax

opmode.**OperatingMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode of operating of the sensor system. See valid mode index in the table below.

Remarks

For AFM many different operating modes are usable. They differ on how the cantilever deflection signal is preprocessed and interpreted. This property defines them.

Some modes has their special settings properties. Many of them are automatically set. But if desired the **Auto**... properties can be set to `False` and with mode specific properties manual settings can be defined.

Attention: If a operating mode is changed a change of cantilever is also necessary for proper operation. Set **Cantilever** accordingly.
For more information please refer to the Nanosurf Software Reference Manual.

Table of operating mode values and description:

State No.	Name	Description
0	OpMode_User	Undefined
1	OpMode_STM	For STM scan heads use this index
2	OpMode_StaticAFM	AFM only: Static deflection mode
3	OpMode_DynamicAFM	AFM only: Dynamic force mode
4	OpMode_PhaseContrast	AFM only: Phase contrast mode
5	OpMode_ForceModulation	AFM only: Force modulation mode
6	OpMode_SpreadingResistance	AFM only: Spreading resistance mode
7	OpMode_ConstPhase	AFM only: Constant phase mode
8	OpMode_A_Probe_dF	A probe only: Frequency modulation mode
9	OpMode_Lateral Force	AFM only: Lateral force mode

Example

```
' enable dynamic AFM and use NCLR Lever
objOpMode.OperatingMode = 3
objOpMode.Cantilever = 1
```

See also

Property Cantilever.

7.9.1.15 OperatingMode::PeakAmplReduction

Returns or set the amplitude reduction from the resonance peak at auto peak search.

Syntax

opmode.**PeakAmplReduction** [= value]

Setting

Argument	Type	Description
value	double	Defines the reduction of the amplitude from resonance peak maximum in [%].

Remarks

This property sets the amplitude reduction value used at automatically resonance peak searches. The actual vibrating frequency is set not to the resonance peak of the cantilever but at either side of the peak with a small reduction of the amplitude. The amount of this reduction is defined with this property. To which side of the resonance peak the frequency shift is done set property **PeakUpperSideBand**.

The setting of this property is used in the operating modes "Dynamic force" and "Phase Contrast".

See also

Property OperatingMode, PeakUpperSideBand

7.9.1.16 OperatingMode::PeakUpperSideBand

Returns or set the to which side of the resonance peak the frequency should be shifted.

Syntax

opmode.**PeakUpperSideBand** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if the frequency is shifted to higher frequency. `False` shifts the vibrating frequency to lower values.

Remarks

This property sets the amplitude reduction value used at automatically resonance peak searches. The actual vibrating frequency is set not to the resonance peak of the cantilever but at either side of the peak with a small reduction of the amplitude. To which side of the resonance peak the frequency shift is done set this property.

The setting of this property is used in the operating modes "Dynamic force" and "Phase Contrast".

See also

Property OperatingMode, PeakAmplReduction

7.9.1.17 OperatingMode::ReferencePhase

Returns or set the reference phase for phase measurement.

Syntax

opmode.**ReferencePhase** [= phase]

Setting

Argument	Type	Description
phase	double	Defines the reference phase in [radian].

Remarks

This property sets the reference phase for the phase measurement of the cantilever returning vibrating signal. If **AutoReferencePhase** is set the reference phase is set automatically after approach. The microscope electronics is measuring the difference between the reference phase signal and the sensor return signal. Best performance of this measurement is done at 90° phase difference between these signals. Readjusment of the phase can be triggered with **SearchReferencPhase**.

The setting of this property is used in the operating mode "Phase Contrast".

Example

```
' set the reference phase to 30°
objOpMode.ReferencePhase = 30.0 / 180.0 * 3.1415
```

See also

Property OperatingMode, AutoReferencePhase
Method SearchReferencePhase

7.9.1.18 OperatingMode::ShowFreqSearchChart

Returns or set a flag to define if bode plot charts as a result of frequency peak search are shown or not.

Syntax

opmode.**ShowFreqSearchChart** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` if charts should be displayed.

Remarks

This property defines if all bode plot charts as a result of a frequency peak search is displayed in new image documents or not.

Frequency peak searches can be performed automatically if **AutoVibrartingFreq** is enabled or if **SearchVibratingFreq** is called.
To display the chart only if desired call **CaptureFreqSearchChart**.

Example

```
' see example at method SearchVibratingFreq
```

See also

Property AutoVibratingFreq
Method SearchVibratingFreq, CaptureFreqSearchChart

7.9.1.19 OperatingMode::TipSignalMode

Defines the configuration of the cantilever tip signal.

Syntax

mode = *opmode*.**TipSignalMode**

Result

Argument	Type	Description
mode	long	Defines the mode of operation for tip signal

Remarks

This property defines the configuration of the tip connection.
The tip signal can be wired differently to user in/outputs or internal signals of the controller.

Table of available mode values:

State No.	Name	Description
0	`TipSig_CurrentSensInput`	Tip signal is configured as current measurement input.
1	`TipSig_VoltageOutput`	Tip signal is configured as a voltage output
2	`TipSig_DirectFeedtrough`	Tip signal is connected directly to the easyScan 2 Signal Access Modul Advanced BNC Input "Tip Signal"

See also

Property objCtrl.TipVoltage

Version info

Software v1.5.1.1 or later

7.9.1.20 OperatingMode::VibratingAmpl

Returns or set the free vibrating amplitude.

Syntax

opmode.**VibratingAmpl** [= ampl]

Setting

Argument	Type	Description
ampl	double	Defines the free vibrating amplitude in [V].

Remarks

This property sets the free amplitude of the cantilever. The excitation of the cantilever is set so that the returning sensor signal is at this value. During the adjustment of the amplitude the cantilever is withdrawn from the surface. The excitation frequency is defined by **VibratingFreq**.

The setting of this property is used in the operating modes "Dynamic force" and "Phase Contrast".

Example

```
' enable dynamic AFM and set amplitude to 50mV
objOpMode.OperatingMode = 3
objOpMode.VibratingAmpl = 0.05 '[V]
```

See also

Property OperatingMode, VibratingFreq

7.9.1.22 OperatingMode::VibratingFreq

Returns or set the vibrating frequency.

Syntax

opmode.**VibratingFreq** [= freq]

Setting

Argument	Type	Description
freq	double	Defines the vibrating frequency in [Hertz].

Remarks

This property sets the excitation frequency of the cantilever. If **AutoVibratingFreq** is set the frequency is set automatically to the resonance peak of the cantilever. The amplitude if the vibration is defined by **VibratingAmpl**.

The setting of this property is used in the operating modes "Dynamic force" and "Phase Contrast".

Example

```
' read the resonance frequency of the cantilever
freq = objOpMode.VibratingFreq
```

See also

Property OperatingMode, AutoVibratingFreq, VibratingAmpl,

7.9.2 Methods

7.9.2.1 OperatingMode::CaptureFreqSearchChart

Creates an image document with the last frequency search bode plot data.

Syntax

opmode.**CaptureFreqSearchChart**

Remarks

This method creates an new image document with the last executed frequency search result.

Example

```
' define search range
objOpMode.FreqSearchStart = 100000 'Hz
objOpMode.FreqSearchEnd   = 200000 'Hz
objOpMode.ShowFreqSearchChart = False

objOpMode.SearchVibratingFreq
Do While objOpMode.IsFreqSearchRunning : Loop
objOpMode.CaptureFreqSearchChart
```

See also

Method SearchVibratingFreq, IsFreqSearchRunning
Method Application.SaveDocument

7.9.2.2 OperatingMode::FreqSearchResult

Returns or set the sensor operating mode.

Syntax

status = *opmode*.**FreqSearchResult**

Setting

Argument	Type	Description
status	long	Returns a status number which informs about the last frequency search result. See possible status numbers in the table below.

Remarks

This method returns the status of the last executed frequency search. Either called automatically at approach or manually by **SearchVibratingFreq**.

Table of possible status results:

Status No.	Name	Description
0	FreqSweepStat_PeakNotFound	A frequency peak could not be found
1	FreqSweepStat_PeakFound	Frequency peak successfully found. Read **VibratingFreq** for value.
2	FreqSweepStat_Running	Frequency search is in process.

Example

```
' see example at method SearchVibratingFreq
```

See also

Property AutoVibratingFreq
Method SearchVibratingFreq, IsFreqSearchRunning

7.9.2.3 OperatingMode::GetFreqSweepLine / GetFreqSweepLine2

Returns a string of data values of a stored frequency data line.

Syntax

str_array = *objData*.**GetFreqSweepLine(***group, channel, line,filter,conversion***)**
variant_array = *objData*.**GetFreqSweepLine2(***group, channel, line,filter,conversion***)**

Argument

Parameter	Type	Description
group	short	desired group index
channel	short	desired channel index
line	short	desired line index
filter	short	index of mathematical filter to be used
conversion	short	index of conversion type of results

Result

Result	Type	Description
str_array	String	Character string with comma separated values of all the values of the data line
variant_array	double array	numerical array of values of all the values of the data line

Remarks

This method returns a string of data values of a data line stored in the container. The signal will be extracted and the data values are processed with a filters as available for the user in the "Chart Toolbar". The result is in a comma separated string in different numerical formats.

The argument *line* is the number of the data line to extract. 0 is the bottom line and the value property **Lines** -1 the top most one.

The argument *filter* defines the data processing algorithm to be used.

Table of filter index:

Filter No.	Filter Name	Description
0	FilterRaw	No data processing
1	FilterMean	The mean value is subtracted
2	FilterPlane	The background plane is subtracted
3	FilterDerive	The derivative of the signal is calculated
4	FilterParabola	A second order fit is subtracted
5	FilterPolynominal	A forth order fill is subtracted

For more detailed description of the filter algorithm please refer to the Nanosurf Software Reference Manual.

The argument *conversion* defines the format of the resulting string array.

Table of conversion index:

Conversion No.	Conversion Name	Description
0	ConversionBinary16	Output as signed 16bit data values
1	ConversionPhysical	Output as floating point values in physical base unit
	ConversionBinary32	Output as signed 32bit data values

See also

Lines Property

7.9.2.5 OperatingMode::IsFreqSearchRunning

Returns a flag if frequency peak search is running or not.

Syntax

flag = *opmode*.**IsFreqSearchRunning**

Result

Argument	Type	Description
flag	Boolean	Returns `True` if frequency search is currently executing otherwise `False`.

Remarks

This method returns `True` if a frequency sweep is running. A sweep can be started automatically by an approach or method **SearchVibratingFreq**.

Example

```
' see example at method SearchVibratingFreq
```

See also

Property AutoVibratingFreq
Method SearchVibratingFreq

7.9.2.6 OperatingMode::IsPhaseSearchRunning

Returns a flag if phase calibration is running or not.

Syntax

flag = *opmode*.**IsPhaseSearchRunning**

Result

Argument	Type	Description
flag	Boolean	Returns `True` if phase calibration is currently executing otherwise `False`.

Remarks

This method returns `True` if a phase calibration is running. A calibration can be started automatically after an approach or method **SearchReferencePhase**.

This method is used in the operating mode "Phase Contrast".

Example

```
' see example at method SearchReferencePhase
```

See also

Property AutoReferencePhase, ReferencePhase
Method SearchReferencePhase

7.9.2.7 OperatingMode::SearchReferencePhase

Calibrates the reference phase to the actual input phase..

Syntax

opmode.**SearchReferencePhase**

Remarks

This method calibrates the reference phase according to the actual sensor signal phase that best sensitivity is reached. Best sensitivity is reached when the reference phase and the input phase are 90° phase shifted.

During the calibration method **IsPhaseSearchRunning** returns `True`. The result of the calibration can be read afterwards from property **ReferencePhase**.

Example

```
' recalibrate phase
objOpMode.SearchReferencePhase
Do While objOpMode.IsPhaseSearchRunning : Loop
MsgBox "New reference phase is " & (objOpMode.ReferencePhase / 3.1415 * 180.0) &
"°"
```

See also

Property ReferencePhase, AutoReferencePhase
Method IsPhaseSearchRunning

7.9.2.8 OperatingMode::SearchVibratingFreq

Searches the resonance peak of the cantilever and set the vibrating frequency.

Syntax

opmode.**SearchVibratingFreq**

Remarks

This method searches the resonance peak of a cantilever. It performs a excitation frequency sweep in a certain frequency range and detects frequency with amplitude maximum. In a second frequency sweep a more close observation of the found resonance frequency is performed and the property **VibratingFreq** is set according to the **Peak**... property.

During the search method **IsFreqSearchRunning** returns `True`. If a search was successful or not is returned by **FreqSearchResult**. The result of the frequency sweep the bode plot can be saved with **CaptureFreqSearchChart** or automatically if **ShowFreqSearchChart** is enabled.

A set of properties is defining the resonance search:

Property name	Purpose
FreqSearchStart	Defines the lower frequency of the vibrating frequency search range
FreqSearchEnd	Defines the upper frequency of the vibrating frequency search range
FreqSearchStep	Defines the step resolution of the vibrating frequency search
AutoFreqSearchRange	Flags if the frequency search area is automatically calculated

PeakAmplReduction	Defines the shift of operating frequency point from peak maximum
PeakUpperSideBand	Flags if the shift is to the upper or lower side of the peak

Example

```
' manual search of resonance peak

' define search range
objOpMode.FreqSearchStart = 150000 'Hz
objOpMode.FreqSearchEnd   = 250000 'Hz

' execute search
objOpMode.ShowFreqSearchChart = True
objOpMode.SearchVibratingFreq
Do While objOpMode.IsFreqSearchRunning : Loop

' check if peak found and report result
If objOpMode.FreqSearchResult = 1 Then
  MsgBox "Resonance found at " & objOpMode.VibratingFreq & "Hz"
Else
  MsgBox "No resonance peak found"
End If
```

See also

Property VibratingFreq, ShowFreqSearchChart
Method IsFreqSearchRunning, FreqSearchResult

7.10 Scan

The Scan class handles the microscope's imaging subsystem.

Imaging is done by a line by line scanning process over the surface. During the scanning the z height information and other supplementary signals are recorded at data points along each scan line. These data points are stored in N*M Matrixes and are displayed on screen as charts.

A set of properties are defining the physical imaged area, the recorded signals and the number of data points. See the property table below. For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

A single image frame is measured by calling **StartFrameUp**, a continuous scan by calling **Start**. Is a complete image frame measured the stored results can by copied in a image document by method **StartCapture**. If a script is interested in numeric values of a scan line in the matrix use **GetLine** method.

A object pointer to this class is provided by the Application.Scan object property.

Table of properties for Scan class:

Property name	Purpose
AutoCapture	Get or set the flag if auto capture is active
AutoDeleteBuffer	Get or set the auto delete buffer function
AutoSlopeCorrection	Enable or disable the auto slope correction function
ImageWidth	Physical width of a image frame
ImageHeight	Physical height of a image frame
Points	Number of data points measured per scan line
Lines	Number of scan lines per image frame
Scantime	Speed of scan movement per scan line
Rotation	Z-Rotation of the image frame regarding the physical coordinate system
SlopeX	X-Axis rotation of the image plane regarding the physical coordinate system
SlopeY	Y-Axis rotation of the image plane regarding the physical coordinate system
CenterPosX	Offset of the image center regarding the X-Axis of the physical coordinate system
CenterPosY	Offset of the image center regarding the Y-Axis of the physical coordinate system
Overscan	Relation between the physical scan line length and the image width
ZPlane	Offset of the image plane regarding the Z-Axis of the physical coordinate system
Scanmode	Mode of scanning if in scan was started with method Start
Measuremode	Mode of scan line measurement
LineMode	Mode how a scan line is scanned
LineScanning	Mode how a scan line is scanned
RelTipPos	Offset of tip in ConstHeight LineMode
SyncOutMode	Returns or selects the mode of the synchronization output
FirstScanlineRep	Returns or set the number of repetitions of the first scan line per frame
ContourEnabled	Return or set ContourEnabled
AutoReadjustProbeEnabled	Return or set AutoReadjustProbeEnabled
ReadjustLiftHeight	Return or set ReadjustLiftHeight
SndScanDynamicAmplitude	Return or set SndScanDynamicAmplitude

SndScanDynamicAmplitudeEnabled	Return or set SndScanDynamicAmplitudeEnabled
SndScanForceModulationAmplitude	Return or set SndScanForceModulationAmplitude
SndScanForceModulationAmplitudeEnabled	Return or set SndScanForceModulationAmplitudeEnabled
SndScanEnableDarkMode	Return or set SndScanEnableDarkMode
SndScanEnableKPFM	Return or set SndScanEnableKPFM
SndScanSndLockInExcitationAmplitude	Return or set SndScanSndLockInExcitationAmplitude
SndScanSndLockInExcitationAmplitudeEnabled	Return or set SndScanSndLockInExcitationAmplitudeEnabled
PrescanSpeedup	Return or set the speedup of the Prescan scan mode

Table of methods for Scan class:

Method name	**Purpose**
DeleteBuffer	Deletes the content of the chart buffer
ShowWindow	Controls the visibility of the imaging window
Start	Starts image scanning.
Stop	Stops image scanning
Pause	Pauses the scanning
StartFrameUp	Start a single scan frame direction upward
StartFrameDown	Start a single scan frame direction downward
StartPrescan	Start a single QuickPrescan (direction upward)
StopPrescan	Stops a running QuickPrescan
Currentline	Get number of the current measured scan line
IsScanning	Retrieve the information whether a scanning is in process or not
IsScanningPrescan	Retrieve the information whether a QuickPrescan is in process or not
IsPaused	Return true if the current imaging process is paused
StartCapture	Prepare a image capture if scanning or do it immediately
StopCapture	Clear a prepared image capture
IsCapturing	Retrieve the information whether a capture is prepared or not
StartSlopeCorrection	Starts the slope correction

IsSlopeCorrectionRunning	Returns if a slope correction process is running or not
GetLine / GetLine2	Retrieve the data point values of a scan line. Returns the value as string or variant.
ImageSize	Set the physical size of a image
GetFrameDir	Retrieve the current scan direction

7.10.1 Properties

7.10.1.1 Scan::AutoCapture

Returns or set a flag if AutoCapture is activated.

Syntax

*scan.***AutoCapture** [= flag]

Setting

Argument	Type	Description
flag	boolean	Set to True AutoCapture is activated and set to False AutoCapture is deactivated.

Remarks

none

See also

7.10.1.2 Scan::AutoDeleteBuffer

Get or set the auto delete buffer function.

Syntax

*scan.***AutoDeleteBuffer** [= state]

Setting

Argument	Type	Description

state	bool	If set to TRUE the function is enabled. If set to FALSE the function is disabled

Remarks

If the AutoDeleteBuffer function is active the chart buffer will be automatically delete every time the Scan restarts.

For more information about this function please refer to the Nanosurf Software Reference Manual.

See also

Method DeleteBuffer

7.10.1.3 Scan::AutoReadjustProbeEnabled

Returns or set the AutoReadjustProbeEnabled.

Syntax

*scan.***AutoreadjustProbeEnabled** [= state]

Setting

Argument	Type	Description
state	bool	Defines the AutoReadjustProbeEnabled state

Remarks

See also

Property ReadjustLiftHeight

Version info

Software v3.6.0.0 or later

7.10.1.4 Scan::AutoSlopeCorrection

Enable or disable the auto slope correction function.

Syntax

*scan.***AutoSlopeCorrection** [= state]

Setting

Argument	Type	Description
state	bool	If set to TRUE the function is enabled. If set to FALSE the function is disabled

Remarks

If the AutoSlopeCorrection function is active the X/Y slopes will be corrected after every approach.

For more information about this function please refer to the Nanosurf Software Reference Manual.

See also

Property

7.10.1.5 Scan::CenterPosX

Returns or set the image X-Axis center position.

Syntax

*scan.***CenterPosX** [= pos]

Setting

Argument	Type	Description
pos	double	Defines the X-Axis position of the image center in meter

Remarks

The image can be place anywhere inside the maximal scan area defined by the scan head. To place a image not in the center of the scan area a displacement vector composed of **CenterPosX** and **CenterPosY** can be used. The maximal X-Axis displacement can be calculated by *(MaxScanrange -* ***ImageWidth*** */ 2)* if **Overscan** and **Rotation** are both zero.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' place a 30um image off center to (20um,-50um)
```

```
objScan.ImageSize 30e-6,30e-6
objScan.CenterPosX  = 20e-6
objScan.CenterPosY  = -50e-6
```

See also

Property CenterPosY**,** ImageWidth, Overscan, Rotation

7.10.1.6 Scan::CenterPosY

Returns or set the image Y-Axis center position.

Syntax

*scan.***CenterPosY** [= pos]

Setting

Argument	Type	Description
pos	double	Defines the Y-Axis position of the image center in meter

Remarks

The image can be place anywhere inside the maximal scan area defined by the scan head. To place a image not in the center of the scan area a displacement vector composed of **CenterPosX** and **CenterPosY** can be used. The maximal Y-Axis displacement can be calculated by *(MaxScanrange - **ImageHeight** / 2)* if **Overscan** and **Rotation** are both zero.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' place a 30um image off center to (20um,-50um)
objScan.ImageSize 30e-6,30e-6
objScan.CenterPosX  = 20e-6
objScan.CenterPosY  = -50e-6
```

See also

Property CenterPosX**,** ImageHeight, Overscan, Rotation

7.10.1.7 Scan::ContourEnabled

Returns or set the ContourEnabled.

Syntax

scan.**ContourEnabled** [= state]

Setting

Argument	Type	Description
state	bool	Defines the ContourEnabled state

Remarks

Version info

Software v3.6.0.0 or later

7.10.1.8 Scan::FirstScanlineRep

Returns or set the number of repetitions of the first scan line per frame.

Syntax

scan.**FirstScanlineRep** [= val]

Setting

Argument	Type	Description
val	long	Defines the number of repetitions of first scan line at the beginning of a image frame

Remarks

At a start of a new image measurement it can happen that the scan head signals are not stable after the movement to the first scan line to start an image frame. Therefore the program measure the first scaned line twice to get a nice first scan line.
In some cases this repetition of the first scan line is not of interest and should be switched off. in other cases on repetition is not enough to stabilize the signal and more repetitions are desired.

This property is controlling this repetitions. A Zero value means no repetition, a value of one means one repetition and so on.

Example

```
' prepare a single profile
objImageSize 1e-6,0     ' 1um length
objScan.Points = 1024
objScan.Lines = 1
objScan.FirstScanlineRep  = 0

' measure profile
objScan.StartFrameUp
Do While objScan.IsScanning : Loop

' read profile
scanline = objScan.GetLine(0,1,0,2,1)
```

See also

Property Lines

Version info

Software v1.4.0 or later

7.10.1.9 Scan::ImageHeight

Returns or set the physical width of a image frame.

Syntax

scan.**ImageHeight** [= height]

Setting

Argument	Type	Description
height	double	Defines the height of a image frame in meter

Remarks

The physical height of a image frame is defined by this property. The number of scan lines per image frame is defined in property **Lines**. A ImageHeight of zero is allowed and means that all scan lines are measured at the same position.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.
Proper scan head calibration is necessary to provide accurate image information.

Example

```
' define a 20um image frame size
```

```
objScan.ImageWidth  = 20e-6
objScan.ImageHeight = 20e-6
```

See also

Property ImageWidth
Method ImageSize

7.10.1.10 Scan::ImageWidth

Returns or set the physical width of a image frame.

Syntax

scan.**ImageWidth** [= width]

Setting

Argument	Type	Description
width	double	Defines the width of a image frame in meter

Remarks

The physical length of each scan line is defined by this property. The number of data points per scan line defined in property **Points** are spread continuously along the width of a scan line. The time to measure along a scan line is defined in the property **Scantime**. A width of zero is allowed and means that all data points are measured at the same position.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.
Proper scan head calibration is necessary to provide accurate image information.

Example

```
' define a 20um image frame size
objScan.ImageWidth  = 20e-6
objScan.ImageHeight = 20e-6
```

See also

Property ImageHeight, Points, Scantime
Method ImageSize

7.10.1.11 Scan::LineMode

Returns or set the mode a scan line is scanned.

Syntax

scan.**LineMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the how a scan line is scanned. See modes in the table below.

Remarks

Scan lines can be measured differently. This property defines this.

Table of scan line mode values and description:

State No.	Name	Description
0	LineMode_Standard	The tip is scanned over the surface with Z-Controller settings and topography information is recorded
1	LineMode_ConstHeight	The tip hovers above the surface at a defined distance. The distance is defined by scan.RelTipPos property. Topography height is only recorded at the beginning and end of each scan line.

See also

Property RelTipPos

Version info

Available since Software v1.5.0 (No longer available v3.6 and up)

7.10.1.12 Scan::Lines

Returns or set the scan lines per image frame.

Syntax

scan.**Lines** [= lines]

Setting

Argument	Type	Description
lines	long	Defines the number of scan lines per scan frame

Remarks

A image frame is composed by individual scan lines. The number of scan lines is defined by this property.

To scan a frame a minimal value of two scan lines have to be set which are placed at the bottom and the top of the image height. More scan lines are spread continuously along the height of a scan frame defined by property **ImageHeight**.

A line value of one set the image height to zero and single profile line can be measured.

The movement from one scan line to the next is done at the left side of a image frame and is always as smooth as possible by the resolution of the electronics and not related to the number of scan lines.

Example

```
' define a image
objScan.ImageSize 20e-6,20e-6
objScan.Points = 256
objScan.Lines  = 256
```

See also

Property ImageHeight, Points
Method ImageSize

7.10.1.13 Scan::LineScanning

Returns or set the mode a scan line is scanned.

Syntax

scan.**LineScanning** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the how a scan line is scanned. See modes in the table

below.

Remarks

Scan lines can be measured differently. This property defines this.

Table of scan line mode values and description:

State No.	Name	Description
0	Standard	The tip is scanned over the surface with Z-Controller settings and topography information is recorded
1	Dual scan	In Dual-Pass Imaging Mode each scan line is measured first with z-controller on and then a second time with lifted tip and Z-Controller off, both as Forward Scan and Backward Scan. The parameter “Contour” enables the contour reproduction, otherwise only the slope is corrected. The parameter “Lift Height” defines the lift up distance used for the second pass. The z reference position is taken at the tip z position at the start x/y-position of the second pass.
2	Interlaced	In Interlaced Dual-Pass Imaging Mode each scan line is measured first as Forward scan line with z-controller on and then during the backward scan line with lifted tip and Z-Controller off . The parameter “Contour” enables the contour reproduction, otherwise only the slope is corrected. The parameter “Lift Height” defines the lift up distance used for the second pass. The z reference position is taken at the tip z position at the start x/y-position of the second pass.
3	Second scan only	In Second-Pass Only Imaging Mode each scan line is measured only with z-controller off. At the beginning of the scan line the Z-Controller is switched off and the tip is lifted. Then only Slope corrected Second-Pass is possible. Optional the Surface reference is probed again prior the backward scan line. The parameter “Dual Probing” activates baseline probing of surface also for the backward scan. Otherwise baseline probing is only done for forward scan. The parameter “Lift Height” defines the lift up distance used for the second pass. The z reference position is taken at the tip z position at the start x/y-position of the second pass.

See also

Property RelTipPos

Version info

Software v3.6.0.0 or later

7.10.1.14 Scan::Measuremode

Returns or set the mode of measure a scan line.

Syntax

scan.**Measuremode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode of measure a scan line. See modes in the table below.

Remarks

Each scan line is divided in a forward scan and a backward scan. Which direction is stored in the image matrix is defined by this property.

Table of measure mode values and description:

State No.	Name	Description
0	Measure_None	not allowed
1	Measure_Forward	Record forward scan data only
2	Measure_Backward	Record backward scan data only
3	Measure_FwBw	Record forward and backward scan data

See also

Method Start

7.10.1.15 Scan::Overscan

Returns or set the over scan factor per scan line.

Syntax

scan.**Overscan** [= overscan]

Setting

Argument	Type	Description
overscan	double	Defines the over scan factor per scan line in percentage

Remarks

Each scan line can be set larger that the measured and displayed part of it. This can help on bad samples reducing start of scan line signal distortions. If Overscan is a none zero vale then the physical scanning of a scan line is larger than defined in ImageWidth. On both sides of the scan line a part of the movement is suppressed during data accusation. The length of the suppressed part is

*Overscan size = ImageWidth * Overscan / 100*

therefore the real physical movement is

*scan line length = ImageWidth * (1+2*Overscan/100)*

Example

```
' activate an over scan of 10%
objScan.ImageWidth = 30e-6
objScan.Overscan   = 10.0
```

See also

Property ImageWidth

7.10.1.16 Scan::Points

Returns or set the data points per scan line.

Syntax

scan.**Points** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of data points per scan line

Remarks

During the movement along each scan line a number of data points are taken and stored in a matrix in memory. The signal channels which are measured at each data point is related to the active operating mode but normally at least the z height information signal is measured.

A minimal value of two data points have to be set which are placed at the start and the end of each scan line. More data points are spread continuously along the width of a scan line defined by property **ImageWidth**.

The scan movement is always as smooth as possible by the resolution of the electronics and not related to the number of data points. Depending on the used Z Controller algorithm a filtering of the measurement can be enabled to suppress noise.

Example

```
' define a image
objScan.ImageSize 20e-6,20e-6
objScan.Points = 256
objScan.Lines  = 256
```

See also

Property ImageWidth, Lines, Scantime
Method ImageSize
Class ZController

7.10.1.17 Scan::ReadjustLiftHeight

Returns or set the ReadjustLiftHeight.

Syntax

scan.**ReadjustLiftHeight** [= distance]

Setting

Argument	Type	Description
distance	double	Defines the ReadjustLiftHeight in meter

Remarks

See also

Property AutoReadjustProbeEnabled

Version info

Software v3.6.0.0 or later

7.10.1.18 Scan::RelTipPos

Returns or set the offset of the tip in ConstHeight mode.

Syntax

scan.**RelTipPos** [= offset]

Setting

Argument	Type	Description
offset	double	Defines the position of the tip relative to surface height in ConstHeight Linemode.

Remarks

If LineMode is set to "ConstHeight" this property defines the position of the tip during the measurement of a scan line. It is a relative position to the current surface height at the beginning of a scan line. The surface height is sensed at each start of a movement (e.g Forward_Scan and Backward_Scan)

Example

```
' Measure an image in ConstHeight Mode
objScan.LineScanning = 3
objScan.RelTipPos = -500e-9  'nm
objScan.Start
```

See also

LineMode Property, Scan::LineScanning

7.10.1.19 Scan::Rotation

Returns or set the image rotation.

Syntax

scan.**Rotation** [= angle]

Setting

Argument	Type	Description
angle	double	Defines the rotation angle of the image in degree

Remarks

The image can be rotated around its center point by any angel according to the physical reference coordinate system. A positive angle defines a rotation of the scan line in positive scientific notation. The center point of the image is defined by the **CenterPosX** and **CenterPosY** properties.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' rotate a image by 45 degree
objScan.Rotation  = 45.0
```

See also

Property CenterPosX, CenterPosY

7.10.1.20 Scan::Scanmode

Returns or set the mode of scanning a image frame.

Syntax

scan.**Scanmode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode of scanning. See modes in the table below.

Remarks

The continuous imaging process can be controlled with this property. The following modes are available. The **Scanmode** value has only an effect if scan process is started with method **Start**.

Table of scan mode values and description:

State No.	Name	Description
0	Scanmode_Continuous	Switch scan direction after each scan frame
1	Scanmode_ContUp	scan direction always upward (scan line 0 to max)
2	Scanmode_ContDown	scan direction always downward (scan line max to 0)

See also

Method Start

7.10.1.21 Scan::Scantime

Returns or set the time used for scanning one scan line.

Syntax

scan.**Scantime** [= time]

Setting

Argument	Type	Description
time	double	Defines the time used to scan one scan line in seconds.

Remarks

This property is defining the time for one scan line in one direction. A scan line needs twice the time defined with Scantime for scanning a scan line in both forward and backward direction.

Normally the time to move a length of **ImageWidth** is equal **Scantime**. But if **Overscan** is none zero the time has to calculated as:

*Time for ImageWidth = Scantime / (1+2*Overscan/100)*

Example

```
' set scantime to get a scan frame in 5min
objScan.Scantime  = 5 * 60 / objScan.Lines / 2
```

See also

Property ImageWidth, Lines, Overscan

7.10.1.22 Scan::SlopeX

Returns or set the X-Axis slope compensation angle.

Syntax

scan.**SlopeX** [= angle]

Setting

Argument	Type	Description
angle	double	Defines the X-Axis slope angle in degree

Remarks

The image plane can be tilted to compensate a sample slope. A positive angle defines a rotation of the scan line in positive scientific notation around the X-Axis.

If **Rotation** is zero the **SlopeX** is used to compensate the slope along the **ImageWidth** and **SlopeY** the slope along **ImageHeight**!

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' compensate a slope of 3 degree
objScan.SlopeX  = 3.0
```

See also

Property SlopeY**,** Rotation

7.10.1.23 Scan::SlopeY

Returns or set the Y-Axis slope compensation angle.

Syntax

scan.**SlopeY** [= angle]

Setting

Argument	Type	Description
angle	double	Defines the Y-Axis slope angle in degree

Remarks

The image plane can be tilted to compensate a sample slope. A positive angle defines a rotation of the scan line in positive scientific notation around the Y-Axis.

If **Rotation** is 90° the **SlopeY** is used to compensate the slope along the **ImageWidth** and **Slopex** the slope along **ImageHeight**!

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' compensate a slope of -0.5 degree
objScan.SlopeY  = -0.5
```

See also

Property SlopeX, Rotation

7.10.1.24 Scan::SndScanDynamicAmplitude

Returns or set the SndScanDynamicAmplitude.

Syntax

*scan.***SndScanDynamicAmplitude** [= amplitude]

Setting

Argument	Type	Description
amplitude	double	Defines the SndScanDynamicAmplitude in volt

Remarks

See also

Property SndScanDynamicAmplitudeEnabled

Version info

Software v3.6.0.0 or later

7.10.1.25 Scan::SndScanDynamicAmplitudeEnabled

Returns or set the SndScanDynamicAmplitudeEnabled.

Syntax

scan.**SndScanDynamicAmplitudeEnabled** [= state]

Setting

Argument	Type	Description
state	bool	Defines the SndScanDynamicAmplitudeEnabled state

Remarks

See also

Property SndScanDynamicAmplitude

Version info

Software v3.6.0.0 or later

7.10.1.26 Scan::SndScanEnableDarkMode

Returns or set the SndScanEnableDarkMode.

Syntax

scan.**SndScanEnableDarkMode** [= state]

Setting

Argument	Type	Description
state	bool	Defines the SndScanEnableDarkMode state

Remarks

Version info

Software v3.6.0.0 or later

7.10.1.27 Scan::SndScanEnableKPFM

Returns or set the SndScanEnableKPFM.

Syntax

scan.**SndScanEnableKPFM** [= state]

Setting

Argument	Type	Description
state	bool	Defines the SndScanEnableKPFM state

Remarks

Version info

Software v3.6.0.0 or later

7.10.1.28 Scan::SndScanForceModulationAmplitude

Returns or set the SndScanForceModulationAmplitude.

Syntax

scan.**SndScanForceModulationAmplitude** [= amplitude]

Setting

Argument	Type	Description
amplitude	double	Defines the SndScanForceModulationAmplitude in volt

Remarks

See also

Property SndScanForceModulationAmplitudeEnabled

Version info

Software v3.6.0.0 or later

7.10.1.29 Scan::SndScanForceModulationAmplitudeEnabled

Returns or set the SndScanForceModulationAmplitudeEnabled.

Syntax

scan.**SndScanForceModulationAmplitudeEnabled** [= state]

Setting

Argument	Type	Description
state	bool	Defines the SndScanForceModulationAmplitudeEnabled state

Remarks

See also

Property SndScanForceModulationAmplitude

Version info

Software v3.6.0.0 or later

7.10.1.30 Scan::SndScanSndLockInExcitationAmplitude

Returns or set the SndScanSndLockInExcitationAmplitude.

Syntax

scan.**SndScanSndLockInExcitationAmplitude** [= amplitude]

Setting

Argument	Type	Description
amplitude	double	Defines the SndScanSndLockInExcitationAmplitude in volt

Remarks

See also

Property SndScanSndLockInExcitationAmplitudeEnabled

Version info

Software v3.6.0.0 or later

7.10.1.31 Scan::SndScanSndLockInExcitationAmplitudeEnabled

Returns or set the SndScanSndLockInExcitationAmplitudeEnabled.

Syntax

scan.**SndScanSndLockInExcitationAmplitudeEnabled** [= state]

Setting

Argument	Type	Description
state	bool	Defines the SndScanSndLockInExcitationAmplitudeEnabled state

Remarks

See also

Property SndScanSndLockInExcitationAmplitude

Version info

Software v3.6.0.0 or later

7.10.1.32 Scan::SyncOutMode

Returns or selects the mode of the synchronization output.

Syntax

scan.**SyncOutMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the signal generated at the synchronization output during a spectroscopy. See mode numbers in the table below.

Remarks

During a spectroscopy modulation different synchronisation signal can be generated at the sync output.
The sync pulse durations is about 4us.

Table of possible modes:

State No.	Name	Description
0	SyncOut_NoSync	No sync pulses are generated output is at Low-Lever.
1	SyncOut_PulsSample	At each spectroscopy sample position a High-Pulse is generated
2	SyncOut_PulsBegin	At the beginning of spectroscopy measurement a High-Pulse is generated
3	SyncOut_PulsEnd	At the end of spectroscopy measurement a High-Pulse is generated
4	SyncOut_PulsBeginAndEnd	At the beginning and the end of spectroscopy measurement a High-Pulse is generated
5	SyncOut_LevelBeginToEnd	A High level is generated during the spectroscopy measurement.

See also

Description of Sync-Output in the Operating Manual

Version info

Software v1.4.0 or later

7.10.1.33 Scan::ZPlane

Returns or set the Z-Axis center position.

Syntax

*scan.***ZPlane** [= pos]

Setting

Argument	Type	Description
pos	double	Defines the Z-Axis center in meter

Remarks

The z axis position of the tip can be predefined with this property. Any Z-Controller feedback position signal is added to this reference plane value.

If the Z-Controller is switch off or is very slow the tip position can be controlled by these property. During scanning the slope compensation plane is added to the tip position too. Therefore the tip is moving during scanning along a 3D plane defined by the position vector CenterPosX, CenterPosY, ZPlane and the rotation vectors SlopeX, SlopeY and Rotation.

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

See also

Property CenterPosX, CenterPosY, SlopeX, SlopeY, Rotation

7.10.1.34 Scan::PrescanSpeedup

Returns or sets the speedup of the Prescan scan mode. The speedup doubles with each integer increase of the PrescanSpeedup value.

Syntax

scan.PrescanSpeedup [= val]

Setting

Argument	Type	Description
PrescanSpeedup	long	Defines the doubling of the speedup value of the Prescan scan mode

Remarks

Example

```
' define a speedup of 8
objScan.PrescanSpeedup = 3
```

See also
Method StartPrescan

7.10.2 Methods

7.10.2.1 Scan::Currentline

Returns the number of the last measured scan line.

Syntax

line = *scan*.**Currentline**

Result

Result	Type	Description
line	long	The last measured scan line number.

Remarks

This method is returning the number of the last measured scan line.
A scan frame is composed of scan lines. Scan line zero is the bottom one and the top most is number **Lines** - 1.

This method can be used to monitor which scan lines are already measured during a imaging process of a scan frame.

Example

```
' keep track of measured scan lines and save topography to file

Dim objApp  : Set objApp  = CreateObject("Nanosurf.Application")
Dim objScan : Set objScan = objApp.Scan
Dim objFS   : Set objFS   = CreateObject("Scripting.FileSystemObject")
Dim objFile : Set objFile = objFS.CreateTextFile("c:\Image.csv")
Dim curline
Dim scanline

' start scan
objScan.StartFrameUp

' process all scan lines
curline = 0
Do While objScan.IsScanning
  If objScan.Currentline > curline Then

    ' save scanline
     scanline = objScan.GetLine(0,1,curline,0,0)
     objFile.WriteLine scanline

    ' wait for next
    curline = curline + 1
  End If
Loop

' process last line
scanline = objScan.GetLine(0,1,curline,0,0)
objFile.WriteLine scanline

' clean up
objFile.Close
Set objFile = Nothing
Set objFS   = Nothing
Set objScan = Nothing
Set objApp  = Nothing
```

See also

Property Lines

Method StartFrameUp, GetLine, IsScanning

7.10.2.2 Scan::DeleteBuffer

Deletes the content of the chart buffer.

Syntax

scan.**DeleteBuffer**

Remarks

This method deletes the content of the chart buffer.

Example

```
' delete chart buffer
objScan.DeleteBuffer
```

See also

Property AutoDeleteBuffer

7.10.2.3 Scan::GetFrameDir

Returns the current scan direction.

Syntax

dir = *scan*.**GetFrameDir**

Result

Result	Type	Description
dir	long	Returns the current scan direction. Valid direction number see table below.

Remarks

This method is returning the number of scan direction.

Table of direction number:

State No.	Name	Description
0	ScanDir_None	Not scanning
1	ScanDir_Up	Currently scanning upward
2	ScanDir_Down	Currently scanning downward

Example

```
objScan.Start
objApp.Sleep(30)
If objScan.GetFrameDir <> 0 Then
  objApp.PrintStatusMsg "Scanning"
Else
  objApp.PrintStatusMsg "No scanning"
End If
```

See also

Method Start

7.10.2.4 Scan::GetLine

Returns a string of data values of a scan line.

Syntax

array = *scan*.**GetLine(***group,channel,scanline,filter,conversion***)**

Argument

Parameter	Type	Description
group	long	number of group
channel	long	number of channel
scanline	long	scan line number
filter	long	index of mathematical filter to be used
conversion	long	index of conversion type of results

Result

Result	Type	Description
array	String	Character string with comma separated values of all the values of the scan line

Remarks

This method returns a string of data values of a scan line. Any signal of a measured image frame can be extracted and the data values can be processed with the same filters as available for the user in the "Chart Toolbar". The result is in a comma separated string in different numerical formats.

The first two arguments *group* and *channel* selects the matrix of a specific signal.

The group number for scanned image frames depends on the measure mode.

Table of group numbers:

Measure mode	Group No.	Group Name	Description
Measure_Forward	0	Group_ForwardScan	Groupf of image data for forwards scan lines
Measure_Backward	0	Group_BackwardScan	Groupf of image data for backward scan lines
Measure_FwBw	0	Group_ForwardScan	Groupf of image data for forwards scan lines
	1	Group_BackwardScan	Groupf of image data for backward scan lines

In each group there are different channels. To get the values of a specific signal one has to know the channel number. If a certain channel is available in a measurement depends on the active operating mode during the measurement.

Table of channel numbers:

Channel No.	Signal Name	Description
0	SigDeflection	Static cantilever deflection signal
1	SigTopography	Z-Topography signal
2	SigAmplitude	Cantilever vibrating amplitude signal
3	SigPhase	Cantilever phase shift signal
4	SigUser	User's defined ADC input signal

The argument *scanline* is the number of the scan line to extract. 0 is the bottom line and property **Lines** -1 the top most one.

The argument *filter* and *conversion* defines the data processing algorithm and formating to be used.
See parameter tables at Data.GetLine Method.

Example

```
' get topography of scan line 5 with plane fit filter active and in [m]
scanline = objScan.GetLine(0,1,5,2,1)

' get user input signal of current scan line, no filter as 16bit values
scanline = objScan.GetLine(0,5,objScan.Currentline,0,0)
```

See also

Property Lines
Method Start, Currentline

7.10.2.5 Scan::ImageSize

Sets width and height of a scan frame.

Syntax

scan.**ImageSize(***width*,*height***)**

Argument

Parameter	Type	Description
width	double	Width of the image frame in meter
height	double	Height of the image frame in meter

Remarks

This method sets the width and height of a scan frame with one call. The difference to setting the size by the properties **ImageWidth** and **ImageHeight** is that no intermediate tip movement is performed between the two property call and value rounding problems are avoided better for small scan frame sizes.

For more detailed description of the arguments see ImageWidth and ImageHeight property.

Example

```
' set scan frame to 100nm
objScan.ImageWidth  = 100e-9
objScan.ImageHeight = 100e-9

' better is using ImageSize method
objScan.ImageSize 100e-9,100e-9
```

See also

Property ImageWidth, ImageHeight

7.10.2.6 Scan::IsCapturing

Returns if a capture is pending or not.

Syntax

flag = *scan*.**IsCapturing**

Result

Result	Type	Description
flag	Boolean	Returns `True` if a capture is pending

Remarks

This method is returing `True` if a capture is pending.

Example

```
If  objScan.IsCapturing Then
  objScan.StopCapture
End If
```

See also

Method StartCapture, StopCapture

7.10.2.7 Scan::IsPaused

Returns if a scan is in paused or not.

Syntax

flag = *scan*.**IsPaused**

Result

Result	Type	Description
flag	Boolean	Returns `True` if imaging is in process

Remarks

This method is returning `True` if a scan is currently paused.

Example

```
' measure a frame
objScan.StartFrameUp

' pause process
objScan.Pause

' do something

' measure a frame
objScan.StartFrameUp
```

See also

Method Pause, StartFrameUp, StartFrameDown, Start

7.10.2.8 Scan::IsScanning

Returns if a scan is in process or not.

Syntax

flag = *scan*.**IsScanning**

Result

Result	Type	Description
flag	Boolean	Returns `True` if imaging is in process

Remarks

This method is returning `True` if a scan is currently running.

Example

```
' measure image
objScan.StartFrameUp
Do While objScan.IsScanning : Loop

' copy image date
objScan.StartCapture
```

See also

Method StartFrameUp, StartFrameDown, Start

7.10.2.9 Scan::IsScanningPrescan

Returns if a Prescan is in process or not.

Syntax

flag = *scan*.**IsScanningPrescan**

Result

Result	Type	Description
flag	Boolean	Returns `True` if imaging is in process

Remarks

This method is returning `True` if a Prescan is currently running.

Example

```
' measure image
objScan.StartPrescan
Do While objScan.IsScanningPrescan : Loop

' copy image data
objScan.StartCapture
```

See also

Method StartPrescan, StartFrameDown, Start

7.10.2.10 Scan::IsSlopeCorrectionRunning

Returns if a slope correction process is running or not.

Syntax

flag = *scan*.**IsSlopeCorrectionRunning**

Result

Result	Type	Description
flag	Boolean	Returns `True` if a slope correction is running

Remarks

This method is returning `True` if a scan is currently running.

Example

```
' slope correction
objScan.StartSlopeCorrection
Do While objScan.IsSlopeCorrectionRunning : Loop
```

See also

Method StartSlopeCorrection

7.10.2.11 Scan::Pause

Pause continuous imaging of scan frames or just single scan frames.

Syntax

scan.Pause

Remarks

This method pauses the continuous imaging process of scan frames as well as of single frames (up and down).

A paused imaging process can be resumed by calling Start or the corresponding StartFrameUp or StartFrameDown functions.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7

' start scan
objScan.Start

' pause immediately
objScan.Pause

' restart
objScan.Start
```

See also

Method IsPaused, Start, StartFrameUp, StartFrameDown

7.10.2.12 Scan::ShowWindow

Defines the display style of the imaging window.

Syntax

scan.**ShowWindow(**style**)**

Arguments

Argument	Type	Description
style	short	Visibility style number

Result

None.

Remarks

The **ShowWindow** method sets the visibility state of the window.

Available styles see Doc.ShowWindow Method

Example

```
objScan.ShowWindow(0) ' hide the imaging window
```

See also

None.

Version info

Software v1.4.0 or later

7.10.2.13 Scan::Start

Starts continuous imaging of scan frames.

Syntax

scan.**Start**

Remarks

This method is starting the continuous imaging process of scan frames. Scanning is only finished by the method **Stop**.

The size and other properties of a scan frame should be predefined prior the start but can be changed anytime also during scanning. Scan frame Property **Scanmode** defines how to proceed after a completed scan frame. A call to **StartCapture** creates a new document after the current frame is finished.

Operating Mode settings and Z Feedback controller settings should be set to reasonable values prior imaging but can be adjusted also at any time during the imaging. Prior to be able to scan an z approach should be performed successfully.

To scan single frames use method **StartFrameUp** or **StartFrameDown**.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7

' start scan
objScan.Start

' do something else ...

' finish immediately
objScan.Stop
```

See also

Property Scanmode
Method Stop, StartFrameUp, StartFrameDown
Class Approach, OperatingMode, ZController

7.10.2.14 Scan::StartCapture

Create a new image document.

Syntax

scan.**StartCapture**

Remarks

This method copies the measured data to a new image document. If a scanning process is running at the time **StartCapture** is called a new image document is created each time a frame is measured.

A pending capture can be canceled with **StopCapture**. If a capture is pending read method **IsCapturing**.

Example

```
' start imaging
objScan.StartFrameUp

' prepare image copy
objScan.StartCapture

' wait until copy is taken at end of frame
Do While objScan.IsCapturing : Loop
```

See also

Method StopCapture, IsCapturing
Method Application.SaveDocument

7.10.2.15 Scan::StartFrameDown

Starts a single down frame image

Syntax

scan.**StartFrameDown**

Remarks

This method is starting a single image starting from the top to the bottom. During the scan process **IsScanning** is `True` and if StartCapturing is called during the frame a new document is created after the scan frame is finished. At the end the tip is moved to the center of the image.

The size and other properties of a scan frame should be predefined prior the start but can be changed anytime also during scanning.

Prior to be able to scan a z-approach should be performed successfully.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7

' measure image
objScan.StartFrameDown
Do While objScan.IsScanning : Loop

' copy image date
objScan.StartCapture
```

See also

Method IsScanning, StartFrameUp
Class Approach

7.10.2.16 Scan::StartFrameUp

Starts a single up frame image

Syntax

scan.**StartFrameUp**

Remarks

This method is starting a single image starting from the bottom to the top. During the scan process **IsScanning** is `True` and if StartCapturing is called during the frame a new document is created after the scan frame is finished. At the end the tip is moved to the center of the image.

The size and other properties of a scan frame should be predefined prior the start but can be changed anytime also during scanning.

Prior to be able to scan a z-approach should be performed successfully.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7

' measure image
objScan.StartFrameUp
Do While objScan.IsScanning : Loop

' copy image date
objScan.StartCapture
```

See also

Method IsScanning, StartFrameDown
Class Approach

7.10.2.17 Scan::StartPrescan

Starts a single up frame image

Syntax

scan.**StartScanPrescan**

Remarks

This method is starting a single Prescan image starting from the bottom to the top. During the scan process **IsScanningPrescan** is `True` and if StartCapturing is called during the frame a new document is created after the scan frame is finished. At the end the tip is moved to the center of the image.

The size and other properties of a scan frame should be predefined prior the start.

Prior to be able to scan a z-approach should be performed successfully.

Example

```
' prepare scan
objScan.ImageSize 2e-6,2e-6
objScan.Scantime = 0.7
objScan.PrescanSpeedup = 3 ' every 8th line is scanned

' measure image
objScan.StartScanPrescan
Do While objScan.IsScanningPrescan : Loop

' copy image date
objScan.StartCapture
```

See also

Method IsScanningPrescan

7.10.2.18 Scan::StopPrescan

Stops Prescan imaging immediately.

Syntax

scan.**StopPrescan**

Remarks

This method stops any Prescan process immediately after the current scan line is finished. The tip is moved to the center of the image.

A possible pending capture flag is also aborted and no document is created.

Example

```
' start scan
objScan.StartPrescan

' do something else ...

' finish immediately
objScan.StopPrescan
```

See also

Method Start, StartPrescan

7.10.2.19 Scan::StartSlopeCorrection

Starts the slope correction

Syntax

scan.**StartSlopeCorrection**

Remarks

This method is starting the X / Y slope correction. During the slope correction process **IsSlopeCorrectionRunning** is `True.`

Example

```
' slope correction
objScan.StartSlopeCorrection
Do While objScan.IsSlopeCorrectionRunning : Loop
```

See also

Method IsSlopeCorrectionRunning

7.10.2.20 Scan::Stop

Stops imaging immediately.

Syntax

scan.**Stop**

Remarks

This method stops any scan process immediately after the current scan line is finished. The tip is moved to the center of the image.

A possible pending capture flag is also aborted and no document is created.

Example

```
' start scan
objScan.Start

' do something else ...

' finish immediately
objScan.Stop
```

See also

Method Start, StartFrameUp, StartFrameDown

7.10.2.21 Scan::StopCapture

Cancel a pending capture

Syntax

scan.**StopCapture**

Remarks

This method cancel a pending capture. If a capture is pending read method **IsCapturing**.

Example

```
' start imaging
objScan.StartFrameUp

' prepare image copy
objScan.StartCapture

' do something

If  objScan.IsCapturing Then
  objScan.StopCapture
End If
```

See also

Method StartCapture, IsCapturing

7.11 ScanHead

The ScanHead class handles the scan head subsystem.

A object pointer to this class is provided by the Application.ScanHead object property.

Table of properties for ScanHead class:

Property name	**Purpose**
HeadName	Get the name of the current attached scan head
HeadID	Get the ID number of the current attached scan head
AFMSensorStatus	Get the AFM sensor status
ApproachMotorStatus	Get the approach motor status
DetectorLateralPos	Get the detectors lateral position
DetectorNormalPos	Get the detectors normal position
LaserPowerMode	Get the mode of the laser power measurement
LaserPower	Get the laser power normalized
LaserPowerAbsolute	Get the laser power as absolute value in [W]

LaserPowerCurrent	Get the laser power as absolute value in [A]
ScanHead::IsLaserControlable	Get the information wether the laser in controlable(On/Off) or not
ScanHead::LaserOn	Get and set the readout laser
ScanHead::LaserSetpoint	Get and set the readout laser setpoint
ScanHead::IsExcitationLaserControllable	Get the information wether the excitation laser is available and controlable(On/Off) or not
ScanHead::ExcitationLaserOn	Get and set the excitation laser setpoint
ScanHead::ExcitationLaserSetpoint	Get and set the excitation laser setpoint
STMSensorStatus	Get the STM sensor status
DeflectionUnitMode	Defines the unit of deflection signal
Cantilever	Defines the selected cantilever by Index position
CantileverByGUID	Defines the selected cantilever by its GUID number
CurrentDeflectionZCompensation	Defines the current Z compensation.
CurrentDeflection	Defines the current active deflection sensitivity
CurrentSpringConst	Defines the current active spring constant used to calculate the deflection force
InvertedUserOutput1	Defines the output polarity of the user output1
InvertedUserOutput2	Defines the output polarity of the user output1
ApproachMotorMode	Get the type of the approach motor
AppraochMotorPosition	Get the position of the approach motor
DeflectionCalibration	Retrieve a object pointer to the Deflection calibration wizard class
ThermalTuning	Retrieve a object pointer to the Thermal Tuning class

Table of methods for ScanHead class:

Method name	Purpose
AdjustDetectorNormalOffset	Readjust the cantilever deflection offset
IsApproachMotorStatusDataValid	Returns "TRUE" if a data request is valid
IsDetectorStatusDataValid	Returns "TRUE" if a data request is valid
IsSensorStatusDataValid	Returns "TRUE" if a data request is valid
TriggerApproachMotorStatus	Request asynchronous data
TriggerDetectorStatus	Request asynchronous data

TriggerSensorStatus	Request asynchronous data
GetCantileverProperty	Get a property value of the active cantilever
SetCantileverProperty	Set a property value of the active cantilever
GetCalibrationSignalMax	Get the maximal calibration value of a signal
SetCalibrationSignalMax	Set a new value to a signal calibration
GetCalibrationSignalName	Get the name of a signal
SetCalibrationSignalName	Set a new name to a signal
GetCalibrationSignalUnit	Get the the unit a signal
SetCalibrationSignalUnit	Set a new unit to a signal
GetAFMSensorStatusMeterRange	Read the various sensor signal status meter range values
GetApproachMotorStatusMeterRange	Read the various approach motor status meter range values

7.11.1 Properties

7.11.1.1 ScanHead::STMSensorStatus

Get the STM sensor status.

Syntax

scanhead.**STMSensorStatus** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Sensor status in [A]

Remarks

None

See also

None

7.11.1.2 ScanHead::LaserPowerMode

Get the laser power.

Syntax

scanhead.**LaserPowerMode** [read only]

Argument

Parameter	Type	Description

value	LONG	`LaserPowerMode_Undefined  = 0,`
		`LaserPowerMode_LaserDrive = 1,`
		`LaserPowerMode_LaserPower = 2,`
		`LaserPowerMode_DetectorIndensity = 3,`

Remarks

This property returns the mode of the laser power measurement unit in the scan head currently attached.

In the LaserDrive mode the laser power monitors the laser's electrical drive.

In the LaserPower mode the laser power monitors the real laser optical power.

In the DetectorSensitivity mode the laser power monitors the sum signal of the detector.

See also

LaserPower, LaserPowerAbsolute, LaserPowerCurrent

7.11.1.3 ScanHead::LaserPowerCurrent

Get the laser power.

Syntax

scanhead.**LaserPowerCurrent** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Laser power in [A]

Remarks

The current optical laser power can be monitored on some scan heads. If this is possible this property returns the energy currently the laser is emitting in [A].

If the laser power readout is not supported then the returned value is negative.

See also

LaserPowerMode, LaserPower, LaserPowerAbsolute

7.11.1.4 ScanHead::LaserPowerAbsolute

Get the laser power.

Syntax

scanhead.**LaserPowerAbsolute** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Laser power in [W]

Remarks

The current optical laser power can be monitored on some scan heads. If this is possible this property returns the energy currently the laser is emitting in [W].

If the laser power readout is not supported then the returned value is negative.

See also

LaserPowerMode, LaserPower, LaserPowerCurrent

7.11.1.5 ScanHead::LaserPower

Get the laser power.

Syntax

scanhead.**LaserPower** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Laser power [0.0 .. +1.0]

Remarks

This property monitors the laser power in the scan head and reports it as a normalized value.

Depending on the scan head the laser power monitors the laser electrical drive power or the laser optical power.

A small value means that the electronics reduce the laser energy to get a fix amount of light onto the detector.

If the laser power signal is high the laser has to be driven with large power in order to get a fix amount of light onto the detector.

See also

LaserPowerMode, LaserPowerAbsolute, LaserPowerCurrent

7.11.1.6 ScanHead::IsLaserControlable

Says if the laser is controlable.

Syntax

scanhead.**IsLaserControlable** [read only]

Argument

Parameter	Type	Description
value	BOOL	Says if the laser can be truned on and off

Remarks

This property tells the user if the connected device/scanhead allows to turn on or off the laser.

See also

ScanHead::LaserOn

7.11.1.7 ScanHead::LaserOn

Tells if the readout laser is ON or OFF.
Turns the readout laser ON or OFF.

Syntax

objScanhead.LaserOn = TRUE or FALSE

value = *objScanhead*.LaserOn

Argument

Parameter	Type	Description
value	BOOL	TRUE if ON FALSE of OFF

Remarks

none

See also

[ScanHead::IsLaserControlable]

7.11.1.8 ScanHead::LaserSetpoint

Get or set the readout laser setpoint.

Syntax

value = *objScanhead*.LaserSetpoint

objScanhead.LaserSetpoint = newValue

Argument

Parameter	Type	Description
value	DOUBLE	Defines the setpoint of the readout laser in watt [W]

Remarks

This property is only available on Drive-Products (DriveAFM, DriveMount, DriveNMA)

See also

None

7.11.1.9 ScanHead::IsExcitationLaserControllable

Tells if the excitation laser is available and controllable.

Syntax

value = *objScanhead*.IsExcitationLaserControllable

Argument

Parameter	Type	Description
value	BOOL	Tells if the excitation laser is available and controllable(ON/OFF and setpoint)

Remarks

This property is only available on Drive-Products (DriveAFM, DriveMount, DriveNMA)

See also

None

7.11.1.10 ScanHead::ExcitationLaserOn

Tells if the excitation laser is ON or OFF.
Turns the excitation laser ON or OFF.

Syntax

objScanhead.ExcitationLaserOn = TRUE or FALSE

value = *objScanhead*.ExcitationLaserOn

Argument

Parameter	Type	Description
value	BOOL	TRUE if ON FALSE of OFF

Remarks

This property is only available on Drive-Products (DriveAFM, DriveMount, DriveNMA)

See also

None

7.11.1.11 ScanHead::ExcitationLaserSetpoint

Get or set the excitation laser setpoint.

Syntax

value = *objScanhead*.ExcitationLaserSetpoint

objScanhead.ExcitationLaserSetpoint = newValue

Argument

Parameter	Type	Description
Setpoint value	DOUBLE	Defines the setpoint of the excitation laser in watt [W]

Remarks

This property is only available on Drive-Products (DriveAFM, DriveMount, DriveNMA)

See also

None

7.11.1.12 ScanHead::HeadName

Get the name of the current attached scan head

Syntax

scanhead.**HeadName** [read only]

Argument

Parameter	Type	Description
value	String	`Name of the scan head`

Remarks

The controller detects the attached scan head and assign to it a name. This name can be read out by this property.

If no scan head or a unknown scan head is attached it returns "undefined".

See also

HeadID

7.11.1.13 ScanHead::HeadID

Get the ID number of the current attached scan head

Syntax

scanhead.**HeadID** [read only]

Argument

Parameter	Type	Description

value	LONG	`Head_NC   = 0,`
		`Head_Unknown = 1`
		`Head_EasyscanSTM  = 2,`
		`Head_EasyscanAFM  = 9,`
		`Head_NaniteAFM    = 12,`
		`Head_FlexAFM      = 14,`
		`Head_LensAFM      = 15,`

Remarks

The controller detects the attached scan head and assign to it ID number. This ID number can be read out by this property.

See also

HeadName

7.11.1.14 ScanHead::DetectorNormalPos

Get the detectors normal position.

Syntax

scanhead.**DetectorNormalPos** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Detector normal position [-1.0 .. +1.0]

Remarks

None

See also

None

7.11.1.15 ScanHead::DetectorLateralPos

Get the detectors lateral position.

Syntax

scanhead.**DetectorLateralPos** [read only]

Argument

Parameter	Type	Description
value	DOUBLE	Detector lateral position [-1.0 .. +1.0]

Remarks

None

See also

None

7.11.1.16 ScanHead::DeflectionUnitMode

Defines used unit for the deflection signal

Syntax

scanhead.**DeflectionUnitMode** [= index]

Argument

Argument	Type	Description
index	long	Defines the unit of the deflection signal.

Remarks

For Static Force Mode AFM different signal units could be of interest. How the deflection signal is displayed in the charts are defined by this property.

The following mode indexes are defined:

```
DefUnitMode_V     = 0,
DefUnitMode_m     = 1,
DefUnitMode_N     = 2,
```

See also

objZCtrl.SetPointForceUnitMode

7.11.1.17 ScanHead::CurrentSpringConst

Get/Set the currently used spring const value

Syntax

scanhead.**CurrentSpringConst**

Argument

Parameter	Type	Description
value	DOUBLE	Spring constant in [N/m]

Remarks

This property handles the actual spring constant calibration value used by the software to calculate the deflection force in [N].

The spring constant of the actual cantilever is predefined by the cantilever browser database. For high precision measurements this calibration is not accurate enough

because the manufacturing tolerances of cantilevers are very large.

Therefore the software provides in the SPM Parameter section Tip/Probe dialog a input field where a more accurate value can be entered.

The actual spring constant measurement can be done by the ThermalTuning dialog with a C3000.

See also

CurrentDeflection

7.11.1.18 ScanHead::CurrentDeflectionZCompensation

Get/Set the current deflection sensitivity value

Syntax

scanhead.**CurrentDeflectionZCompensation**

Argument

Parameter	Type	Description
value	DOUBLE	Deflection Z compensation [- 2.0 .. +2.0]

Remarks

This property changes the actual compensation value used for Z-Axis position coupling suppression of the Deflection signal.

If a scan head calibration file is loaded this value is set to the files default value.

See also

7.11.1.19 ScanHead::CurrentDeflection

Get/Set the current deflection sensitivity value

Syntax

scanhead.**CurrentDeflection**

Argument

Parameter	Type	Description

value	DOUBLE	Deflection sensitivity in [m/V]

Remarks

This property handles the actual deflection sensitivity calibration value used by the software to calculate the deflection in [m].

The deflection sensitivity is predefined by the scan head calibration file. For high precision measurements this calibration is not accurate enough

because the deflection sensitivity is also defined by the actual mounted cantilever and the actual set laser position on the cantilever.

Therefore the software provides in the SPM Parameter section Tip/Probe dialog a input field where a more accurate value can be entered.

The deflection sensitivity is measured by a F/z-Spectroscopy on a hard surface and by analyzing the deflection slope.

The deflection sensitivity calibration wizard in the software can be used to automate this step.

See also

CurrentSpringConst

7.11.1.20 ScanHead::CantileverByGUID

Returns or set the cantilever type mounted in the scan head.

Syntax

scanhead.**CantileverByGUID** [= index]

Setting

Argument	Type	Description
index	long	Defines which cantilever is mounted in the scan head.

Remarks

For AFM different type of cantilevers can be used with different mechanical properties as stiffness or resonance frequency. This property tells the software which cantilever the user has mounted.

The application stores each cantilever definition in a database. It is referenced by a global unique ID number the GUID. If the script knows this GUID it can be used to select a specific cantilever without knowing its index position in the list of cantilevers as it is with the **Cantilever** Property.

Some cantilever are Known Cantilever and others are User Defined Cantilever. Known Cantilever are has fixed predefined GUIDs defined by Nanosurf. User Defined Cantilevers get their GUID at the time a user create a new Cantilever entry in the database.

Here's a list of predefined Known Cantilever and their GUID:

```
 Manufacturer: Anasys Instruments
Name:              GUID:
 AN2-200           {BD61D124-8350-4464-BFE4-1D8A156E4913}
 GLA-1             {9E2BA28D-D843-41bf-8F62-05502B3EDB18}

Manufacturer: AppNano
 ACL-A             {ABB75273-9543-431a-B681-C79B533DD9E6}
 ANSCM             {40AEA787-942C-4d48-A389-DA81571F009C}
 SICON-A           {F7A339A7-E29F-42a9-B7AA-D69C54363B76}

Manufacturer: BudgetSensors
ContAl-G           {ED5A15E6-D3B0-4e64-8C50-809335D3E143}
 Multi75E-G        {9593403B-A476-49a9-AA1F-9C3AEDAC0178}
 Multi75M-G        {03D0715C-A520-4976-A5E2-4FC3078E3821}
 Multi75Al-G       {443A2EDC-5C9C-4d60-843F-C6688BEA1DEA}
 Tap190Al-G        {041FB80E-A179-4170-B5A4-A4EA1CC0A965}

Manufacturer: Nanosensors
CONTR              {89E92173-96FB-4ff9-94D8-42296D00D980}
 CONTSCPt          {1E95D12B-1DDB-4ace-B3AF-BE9C0D52D4FC}
 EFMR              {986305AC-64B5-462e-B37E-6BD5AE447BE3}
 LFMR              {C61FCA2C-6D5D-4105-9FDE-640D263E229F}
 MFMR              {9499F49F-920F-47ec-80B6-883F683FF056}
 NCLR              {62633FD4-0555-4cee-A8B4-B82F4CEFBB48}
 PPP-FMR           {EBA2B75C-AA94-4451-AD36-1388CDABF5E8}
 XYNCHR            {DD3DFE39-455E-40a1-801E-5D5B14CE4080}
```

Attention: For each cantilever type only some operating modes are useful. Set **OperatingMode** accordingly.

For more information please refer to the Nanosurf Software Reference Manual.

Example

```
' enable dynamic AFM and use NCLR Lever
objOpMode.OperatingMode = 3
objScanHead.CantileverByGUID = "{62633FD4-0555-4cee-A8B4-B82F4CEFBB48}"
```

See also

Property OperatingMode

7.11.1.21 ScanHead::Cantilever

Returns or set the cantilever type mounted in the scan head.

Syntax

scanhead.**Cantilever** [= index]

Setting

Argument	Type	Description
index	long	Defines which cantilever is mounted in the scan head.

Remarks

For AFM different type of cantilevers can be used with different mechanical properties as stiffness or resonance frequency. This property tells the software which cantilever the user has mounted.

The cantilevers are defined in a list by the dialog "Config Cantilevers types" in the menu "Options". From top down to the end of list each definition has its index number. Start with index 0. This index number is used with this property.

The software then handles the details about them and adjust the internal microscope electronics accordingly

Attention: For each cantilever type only some operating modes are useful. Set **OperatingMode** accordingly.

For more information please refer to the Nanosurf Software Reference Manual.

Example

```
' enable dynamic AFM and use NCLR Lever
objOpMode.OperatingMode = 3
objScanHead.Cantilever = 1
```

See also

Property OperatingMode

7.11.1.22 ScanHead::AprroachMotorMode

Get the type of the approach motor.

Syntax

value = scanhead.**ApproachMotorMode**

Argument

Parameter	Type	Description

value	LONG	`NotDefined = 0,`
		`LimitSwitches = 1`
		`PositionSensor = 2,`
		`NoApproachStatus = 3`

Remarks

none

See also

7.11.1.23 ScanHead::ApproachMotorStatus

Get the approach motor status.

Syntax

scanhead.**ApproachMotorStatus** [read only]

Argument

Parameter	Type	Description

state	LONG	`LimitStatus_FAIL = '6',`
		`LimitStatus_ERROR = '5',`
		`LimitStatus_NC = '4',`
		`LimitStatus_MAXOUT = '3',`
		`LimitStatus_MININ = '2',`
		`LimitStatus_INRANGE = '1',`
		`LimitStatus_NOTDEFINED = '0',`
		`LimitStatus_NOTDEFINED = -1,`

Remarks

None

See also

None

7.11.1.24 ScanHead::ApproachMotorPosition

Get the position of the approach motor.

Syntax

value = scanhead.**ApproachMotorPosition**

Argument

Parameter	Type	Description
value	DOUBLE	Position in meter

Remarks

none

See also

7.11.1.25 ScanHead::AFMSensorStatus

Get the AFM sensor status.

Syntax

scanhead.**AFMSensorStatus** [read only]

Argument

Parameter	Type	Description
value	LONG	SensorStatus_LASER_TOLOW = '7', SensorStatus_LASER_FAIL = '4', SensorStatus_LASER_TOHIGH = '3', SensorStatus_LASER_OK = '1', SensorStatus_NOTDEFINED = -1,

Remarks

None

See also

None

7.11.1.26 ScanHead::InvertedUserOutput1

Tells if the user output 1 is inverted or not.
Turns the inversion on user output 1 ON or OFF.

Syntax

objScanhead.InvertedUserOutput1 = TRUE or FALSE

value = *objScanhead*.InvertedUserOutput1

Argument

Parameter	Type	Description
value	BOOL	TRUE if inverted FALSE if not OFF

Remarks

none

See also

none

7.11.1.27 ScanHead::InvertedUserOutput2

Tells if the user output 2 is inverted or not.
Turns the inversion on user output 1 ON or OFF.

Syntax

objScanhead.InvertedUserOutput2 = TRUE or FALSE

value = *objScanhead*.InvertedUserOutput2

Argument

Parameter	Type	Description
value	BOOL	TRUE if inverted FALSE if not OFF

Remarks

none

See also

none

7.11.1.29 ScanHead::ThermalTuning

Returns a dispatch pointer to the sub class ThermalTuning. This property is read only.

Syntax

application.**ThermalTuning** [read only]

Result

The **ThermalTuning** property is returning a pointer to the IDispatch interface of the ThermalTuning object.

Remarks

Only one single instance exists of ThermalTuning object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create objects
Dim objApp  : Set objApp  = SPM.Application
Dim objScanhead : Set objScanhead = objApp.Scanhead
Dim objThermalTune : Set objThermalTune = objScanhead.ThermalTuning

' variables
Dim currentAverageData
Dim blockData
Dim frequencyList
Dim shoFitResult
Dim shoFitCurve
Dim numIterations
Dim maxIterations
Dim cantileverLength
Dim cantileverWidth
Dim envDensity
Dim envViscosity
Dim springConstant
numIterations = 0
maxIterations = 1000
cantileverLength = 0.000225
cantileverWidth = 0.000038
envDensity = 1.225
envViscosity = 0.0000185
springConstant = 0

' setup thermal tune
objThermalTune.FreqBandUpperBound 319000 ' Hz
```

```
objThermalTune.FreqResolution 45 ' Hz
objThermalTune.BlockCount 0 ' continuous
objThermalTune.AverageType 1 ' ProportionalWeight
objThermalTune.CantileverTemperature 21 ' degrees celsius
objThermalTune.FreqLowerBound 85000 ' Hz, fit lower bound
objThermalTune.FreqUpperBound 266000 ' Hz, fit upper bound

' start capture
objThermalTune.Start

' data acquisition and calculation loop
do while (numIterations < maxIterations) :
    if (objThermalTune.GetCurrentBlockCount > 0) then
           currentAverageData = objThermalTune.GetCurrentAverage
           blockData = objThermalTune.GetBlock(false)
           frequencyList = objThermalTune.GetFrequencyList
           shoFitResult =
objThermalTune.SimpleHarmonicOscFitOnCurrentAverageAndBounds
           ' calculate k
           springConstant =
objThermalTune.CalculateSpringConstant_Sader(cantileverLength, cantileverWidth,
shoFitResult(2), shoFitResult(3), envViscosity, envDensity)
    end if
    numIterations = numIterations + 1
loop

' stop capture
objThermalTune.stop

' display result
MsgBox springConstant

objThermalTune = nul : Set objThermalTune= Nothing
objApp  = nul : Set objApp  = Nothing
```

See also

class ThermalTuning

7.11.2 Methods

7.11.2.1 ScanHead::AdjustDetectorNormalOffset

Starts the offset calibration process for the normal deflection.

Syntax

flag = *scanhead*.**AdjustDetectorNormalOffset**

Result

Result	Type	Description
none	none	none

Remarks

This method starts the process to recalibrate the normal deflection offset to zero.

See also

Properties ScanHead::DetectorNormalPos

7.11.2.2 ScanHead::GetAFMSensorStatusMeterRange

Returns the normalized SignalMeter border value.

Syntax

value = objScanhead.**GetAFMSensorStatusMeter(***MeterID***)**

Argument

Parameter	Type	Description
MeterID	long	ID of the Status Meter range to read out

Result

Result	Type	Description
value	double	The normalized value of the selected StatusMeterRange

Remarks

The **GetAFMSensorStatusMeterRange()** method returns the normalized value of a selected signalmeter border value .

Available SignalMeter ID's are:

```
SignalMeter_MinRed    = 0,
SignalMeter_MinOrange = 1,
SignalMeter_MinGreen  = 2,
SignalMeter_MaxGreen  = 3,
SignalMeter_MaxOrange = 4,
SignalMeter_MaxRed    = 5,
```

See also

TechDoc "NSF SensorSignal Status Information Documentation"

7.11.2.3 ScanHead::GetCantileverProperty

Returns a property value of the current selected cantilever.

Syntax

value = *objScanhead*.**GetCantileverProperty(***propid***)**

Argument

Parameter	Type	Description
propID	long	ID of the property to read out

Result

Result	Type	Description
value	double	The value of the property

Remarks

The **GetCantileverProperty()** method returns a value of a selected cantilever property.

Available properties are:

```
CantileverProp_LeverLength        = 0,
CantileverProp_LeverWidth         = 1,
CantileverProp_SpringConst        = 2,
CantileverProp_AirResonanzeFrq    = 3,
CantileverProp_AirQFactor         = 4,
CantileverProp_LiquidResonanzeFrq = 5,
CantileverProp_LiquidQFactor      = 6,
```

Example

```
  MsgBox "Current cantilever's spring constant is " &
objScanhead.GetCantileverPropery(2) & "N/m"
```

See also

[ScanHead.SetCantileverProperty]

7.11.2.4 ScanHead::GetCalibrationSignalMax

Returns the calibration value of the selected signal.

Syntax

value = *objScanhead*.**GetCalibrationSignalMax(***sigID***)**

Argument

Paramete r	Type	Description
sigID	long	ID of the signal to read out

Result

Result	Type	Description
value	double	The maximal calibration value of the signal

Remarks

The **GetCalibrationSignalMax()** method returns the calibration value of a signal.

Available signal ID's are:

```
CalibSig_XAxis          = 0,
CalibSig_YAxis          = 1,
CalibSig_ZAxis          = 2,
CalibSig_TipCurrent     = 3,
CalibSig_TipVoltage     = 4,
CalibSig_Ch0_Deflection = 5,
CalibSig_Ch0_Amp        = 6,
CalibSig_Ch0_Phase      = 7,
CalibSig_Ch0_Excitation = 8,
```

Example

```
  MsgBox "Current Z-Axis Range is " &
2.0*objScanhead.GetCalibrationSignalMax(2)*1.0e6 & "um"
```

See also

[ScanHead.SetCalibrationSignalMax]

7.11.2.5 ScanHead::SetCalibrationSignalMax

Sets the calibration value of the selected signal.

Syntax

ok = *objScanhead*.**SetCalibrationSignalMax(***sigID*, *value***)**

Argument

Parameter	Type	Description
sigID	long	ID of the signal to read out
value	double	maximal signal value in it native unit

Result

Result	Type	Description
ok	bool	TRUE if the signal value could be set

Remarks

The S**etCalibrationSignalMax()** method sets the calibration value of a signal.

Available signal ID's are:

```
CalibSig_XAxis          = 0,
CalibSig_YAxis          = 1,
CalibSig_ZAxis          = 2,
CalibSig_TipCurrent     = 3,
CalibSig_TipVoltage     = 4,
CalibSig_Ch0_Deflection = 5,
CalibSig_Ch0_Amp        = 6,
CalibSig_Ch0_Phase      = 7,
CalibSig_Ch0_Excitation = 8,
```

Example

```
objScanhead.SetCalibrationSignalMax(5) = 2.5e-6 '[m]
```

See also

ScanHead.GetCalibrationSignalMax

7.11.2.6 ScanHead::GetCalibrationSignalName

Returns the name of the selected signal.

Syntax

value = *objScanhead*.**GetCalibrationSignalName(***sigID***)**

Argument

Parameter	Type	Description
sigID	long	ID of the signal to read out

Result

Result	Type	Description
name	string	The name of the signal

Remarks

The **GetCalibrationSignalName()** method returns the name of a signal.

Available signal ID's are defined at ScanHead::GetCalibrationSignalMax

See also

ScanHead.SetCalibrationSignalMax

7.11.2.7 ScanHead::SetCalibrationSignalName

Sets the name of the selected signal.

Syntax

ok = *objScanhead*.**SetCalibrationSignalName(***sigID, name***)**

Argument

Parameter	Type	Description

r		
sigID	long	ID of the signal to read out
name	string	new name of the signal

Result

Result	Type	Description
ok	bool	TRUE if the signal name could be set

Remarks

The **SetCalibrationSignalName()** method sets the name of a signal.

Available signal ID's please see at ScanHead.GetCalibrationSignalMax

See also

ScanHead.GetCalibrationSignalMax

7.11.2.8 ScanHead::GetCalibrationSignalUnit

Returns the unit of the selected signal.

Syntax

value = *objScanhead*.**GetCalibrationSignalUnit(***sigID***)**

Argument

Parameter	Type	Description
sigID	long	ID of the signal to read out

Result

Result	Type	Description
name	string	The unit of the signal

Remarks

The **GetCalibrationSignalUnit()** method returns the unit of a signal.

Available signal ID's are defined at ScanHead::GetCalibrationSignalMax

See also

ScanHead::GetCalibrationSignalName

7.11.2.9 ScanHead::SetCalibrationSignalUnit

Sets the name of the selected signal.

Syntax

ok = objScanhead.**SetCalibrationSignalUnit(***sigID, unitname***)**

Argument

Parameter	Type	Description
sigID	long	ID of the signal to read out
unitname	string	new unit of the signal

Result

Result	Type	Description
ok	bool	TRUE if the signal unit could be set

Remarks

The **SetCalibrationSignalUnit()** method sets the unit of a signal.

Available signal ID's please see at ScanHead.GetCalibrationSignalMax

See also

ScanHead.GetCalibrationSignalMax

7.11.2.10 ScanHead::IsApproachMotorStatusDataValid

Returns "TRUE" if a data request is valid.

Syntax

flag = *scanhead*.**IsApproachMotorStatusDataValid**

Result

Result	Type	Description
flag	Boolean	Returns `True` if the requested data is valid

Remarks

This method is returns `True` if the requested data is valid.

Example

```
' start trigger
objScanHead.TriggerApproachMotorStatus

' wait until async data is received
do while (objScanHead.IsApproachMotorStatusDataValid = false) : loop

MsgBox "" & objScanHead.ApproachMotorStatus
```

See also

Properties ScanHead::ApproachMotorStatus
Method ScanHead::TriggerApproachMotorStatusData

7.11.2.11 ScanHead::IsDetectorStatusDataValid

Returns "TRUE" if a data request is valid.

Syntax

flag = *scanhead*.**IsDetectorStatusDataValid**

Result

Result	Type	Description
flag	Boolean	Returns `True` if the requested data is valid

Remarks

This method is returns `True` if the requested data is valid.

Example

```
' start trigger
objScanHead.TriggerDetectorStatus

' wait until async data is received
do while (objScanHead.IsDetectorStatusDataValid = false) : loop

MsgBox "" & objScanHead.LaserPower & " " & objScanHead.DetectorLateralPos & " "
& objScanHead.DetectorNormalPos
```

See also

Properties ScanHead::DetectorLateralPos, ScanHead::DetectorNormalPos, ScanHead::LaserPower
Method ScanHead::TriggerDetectorStatus

7.11.2.12 ScanHead::IsSensorStatusDataValid

Returns "TRUE" if a data request is valid.

Syntax

flag = *scanhead*.**IsSensorStatusDataValid**

Result

Result	Type	Description
flag	Boolean	Returns `True` if the requested data is valid

Remarks

This method is returns `True` if the requested data is valid.

Example

```
' start trigger
objScanHead.TriggerSensorStatus

' wait until async data is received
do while (objScanHead.IsSensorStatusDataValid = false) : loop

' for STM use
MsgBox "" & objScanHead.STMSensorStatus

' for AFM use
```

```
MsgBox "" & objScanHead.AFMSensorStatus
```

See also

Properties ScanHead::AFMSensorStatus, ScanHead::STMSensorStatus
Method ScanHead::TriggerSensorStatus

7.11.2.13 ScanHead::SetCantileverProperty

Sets a property value of the current selected cantilever.

Syntax

bool = objScanhead.**SetCantileverProperty(***propid, value***)**

Argument

Parameter	Type	Description
propID	long	ID of the property to read out
value	double	new value for selected propID

Result

Result	Type	Description
ok	bool	TRUE is the property could be set.

Remarks

The **SetCantileverProperty()** method sets a value of a selected cantilever property.

Available properties are:

```
CantileverProp_LeverLength        = 0,
CantileverProp_LeverWidth         = 1,
CantileverProp_SpringConst        = 2,
CantileverProp_AirResonanzeFrq    = 3,
CantileverProp_AirQFactor         = 4,
CantileverProp_LiquidResonanzeFrq = 5,
CantileverProp_LiquidQFactor      = 6,
```

Example

```
ok = objScanhead.SetCantileverPropery(2, 0.1) ' [N/m]
```

See also

GetCantileverProperty

7.11.2.14 ScanHead::TriggerApproachMotorStatus

Request asynchronous data.

Syntax

scanhead.**TriggerApproachMotorStatus**

Remarks

This method triggers the request to receive approach motor status data.
The IsApproachMotorStatusDataValid flag will be cleared and set to true once the data has arrived.

Example

```
' start trigger
objScanHead.TriggerApproachMotorStatus

' wait until async data is received
do while (objScanHead.IsApproachMotorStatusDataValid = false) : loop

MsgBox "" & objScanHead.ApproachMotorStatus
```

See also

Properties ScanHead::ApproachMotorStatus
Method ScanHead::IsApproachMotorStatusDataValid

7.11.2.15 ScanHead::TriggerDetectorStatus

Request asynchronous data.

Syntax

scanhead.**TriggerDetectorStatus**

Remarks

This method triggers the request to receive detector status data.
The IsDetectorStatusDataValid flag will be cleared and set to true once the data has arrived.

Example

```
' start trigger
objScanHead.TriggerDetectorStatus

' wait until async data is received
do while (objScanHead.IsDetectorStatusDataValid = false) : loop

MsgBox "" & objScanHead.LaserPower & " " & objScanHead.DetectorLateralPos & " "
& objScanHead.DetectorNormalPos
```

See also

Properties ScanHead::DetectorLateralPos, ScanHead::DetectorNormalPos, ScanHead::LaserPower
Method ScanHead::IsDetectorStatusDataValid

7.11.2.16 ScanHead::TriggerSensorStatus

Request asynchronous data.

Syntax

scanhead.**TriggerSensorStatus**

Remarks

This method triggers the request to receive sensor status data.
The IsSensorStatusDataValid flag will be cleared and set to true once the data has arrived.

Example

```
' start trigger
objScanHead.TriggerSensorStatus

' wait until async data is received
do while (objScanHead.IsSensorStatusDataValid = false) : loop

' for STM use
MsgBox "" & objScanHead.STMSensorStatus

' for AFM use
MsgBox "" & objScanHead.AFMSensorStatus
```

See also

Properties ScanHead::AFMSensorStatus, ScanHead::STMSensorStatus
Method ScanHead::IsSensorStatusDataValid

7.12 SignalIO

The SignalIO class handles the microscope's IO subsystem.

A object pointer to this class is provided by the Application.SignalIO object property.

Table of properties for SignalIO class:

Property name	Purpose
EnableUserADC0	Enable User ADC0
EnableUserADC1	Enable User ADC1
UserADC0	Read the current ADC value
UserADC1	Read the current ADC value
ExcitationMode	Set the lever excitation modes
TipSignalMode	Set the tip signal modes
User0CtrlMode	Set the user0 control mode
User0IGain	Set the user0 I gain [0 .. oo]
User0InputPol	Set the user0 input pol
User0OutputFlag	Set the user0 output flag
User0SetPoint	Set the user0 set point
UserDAC0	User Output 0
UserDAC1	User Output 1
MonitorOut0	Defines the signal monitor on BNC Monitor 1 of the C3000
MonitorOut1	Defines the signal monitor on BNC Monitor 2 of the C3000
IsInstalled	Returns if the Advanced Signal Module is installed or not.

7.12.1 Properties

7.12.1.1 SignalIO::EnableUserADC0

Enable or disable the UserADC0.

Syntax

signalIO.**EnableUserADC0** [= state]

Argument

Parameter	Type	Description
state	BOOL	Enable or disable the UserADC0.

Remarks

None

See also

None

7.12.1.2 SignalIO::EnableUserADC1

Enable or disable the UserADC1.

Syntax

signalIO.**EnableUserADC1** [= state]

Argument

Parameter	Type	Description
state	BOOL	Enable or disable the UserADC1.

Remarks

None

See also

None

7.12.1.3 SignalIO::ExcitationMode

Get or set the lever excitation mode.

Syntax

signalIO.**ExcitationMode** [= mode]

Argument

Parameter	Type	Description
mode	LONG	Defines the lever excitation mode. See modes in the table below.

Remarks

Table of lever excitation mode values and description:

State No.	Name	Description
0	LeverMode_InternalSource	Cantilever excitation is controlled by the Nanosurf controller itself.
1	LeverMode_ExternalSource	Cantilever excitation is controlled by an external source

See also

None

7.12.1.5 SignalIO::MonitorOut0

Selects the channel mapped to monitor 0 output.

Syntax

signalIO.**MonitorOut0** [= channel]

Argument

Parameter	Type	Description
channel	Long	Get or set channel

Remarks

Channel table

Value	Name
0	Static Value Register
1	Test Dynamic
2	Reserved
3	Debug
4	Main Input 1
5	Main Input 2
6	Axis Position Input X
7	Axis Position Input Y
8	Axis Position Input Z
9	Extra Input 1
10	Extra Input 2
11	Extra Input 3 or 4

12	Approach
13	Position Output X
14	Position Output Y
15	Position Output Z
16	Mixed Output 3
17	Mixed Output 4
18	Tip Current Input
32	Z-Controller Output
33	Ramp Generator Approach
34	Ramp Generator Scan X
35	Ramp Generator Scan Y
36	Ramp Generator Scan Z
37	Ramp Generator Z-Controller
38	Ramp Generator Z-Direct
39	Ramp Generator Max-Z
40	Z-Controller Input Value
41	Z-Controller Error Value
42	Z-Controller PID Command
43	Z-Controller Sum Value
44	Z-Controller Limited Value
45	Axis Position Controller Output X
46	Axis Position Controller Output Y
47	Analyzer 1 Control Delta F
48	Analyzer 1 Control Amplitude
49	Analyzer 1 Phase
50	Analyzer 1 Amplitude
51	Analyzer 1 X
52	Analyzer 1 Y
53	Analyzer 2 Control Delta F
54	Analyzer 2 Control Amplitude
55	Analyzer 2 Phase
56	Analyzer 2 Amplitude
57	Analyzer 2 X
58	Analyzer 2 Y

See also

SignalIO::MonitorOut1

7.12.1.6 SignalIO::MonitorOut1

Selects the channel mapped to monitor 1 output.

Syntax

signalIO.**MonitorOut1** [= channel]

Argument

Parameter Type Description

channel	Long	Get or set channel

Remarks

Channel table

Value	Name
0	Static Value Register
1	Test Dynamic
2	Reserved
3	Debug
4	Main Input 1
5	Main Input 2
6	Axis Position Input X
7	Axis Position Input Y
8	Axis Position Input Z
9	Extra Input 1
10	Extra Input 2
11	Extra Input 3 or 4
12	Approach
13	Position Output X
14	Position Output Y
15	Position Output Z
16	Mixed Output 3
17	Mixed Output 4
18	Tip Current Input
32	Z-Controller Output
33	Ramp Generator Approach
34	Ramp Generator Scan X
35	Ramp Generator Scan Y
36	Ramp Generator Scan Z
37	Ramp Generator Z-Controller
38	Ramp Generator Z-Direct
39	Ramp Generator Max-Z
40	Z-Controller Input Value
41	Z-Controller Error Value
42	Z-Controller PID Command
43	Z-Controller Sum Value
44	Z-Controller Limited Value
45	Axis Position Controller Output X
46	Axis Position Controller Output Y
47	Analyzer 1 Control Delta F
48	Analyzer 1 Control Amplitude
49	Analyzer 1 Phase
50	Analyzer 1 Amplitude
51	Analyzer 1 X
52	Analyzer 1 Y
53	Analyzer 2 Control Delta F
54	Analyzer 2 Control Amplitude
55	Analyzer 2 Phase

56	Analyzer 2 Amplitude
57	Analyzer 2 X
58	Analyzer 2 Y

See also

SignalIO::MonitorOut0

7.12.1.7 SignalIO::TipSignalMode

Get or set the tip signal mode.

Syntax

signalIO.**TipSignalMode** [= mode]

Argument

Parameter	Type	Description
mode	LONG	Defines the operating mode for lithography. See modes in the table below.

Remarks

Table of tip signal mode values and description:

State No.	Name	Description
0	TipSig_CurrentSensInput	Sets the tip signal to the input current measurement level.
1	TipSig_VoltageOutput	Sets the tip signal to the measured output voltage.
2	TipSig_DirectFeedtrough	Establishes a direct connection between the “Tip-Voltage” Input BNC connector and the cantilever.

See also

None

7.12.1.8 SignalIO::User0CtrlMode

Get or set the user0 controller mode.

Syntax

signalIO.**User0CtrlMode** [= mode]

Argument

Parameter	Type	Description

mode	LONG	Defines the user 0 controller mode. See modes in the table below.

Remarks

Table of user 0 controller operation mode values and description:

State No.	Name	Description
0	User0Ctrl_Off	User 0 controller is off
1	User0Ctrl_On	User 0 controller is on

See also

None

7.12.1.9 SignalIO::User0IGain

Returns or set the integral gain of the user 0 controller.

Syntax

signalIO.**User0IGain** [= gain]

Setting

Argument	Type	Description
gain	double	Defines the amplification of the accumulating sum of the difference between input signal and set point value. Valid values are 0 .. 32767.

Remarks

The I-Gain is defining the amplification of sum of the difference between input signal and the set point value. A higher amplification generates a faster response to a input signal error. But a gain value too high can lead to oscillation of the z feedback loop and amplifies also noise from the input signal.

A value of zero switch of the integral gain completely.

Example

```
signalIO.User0IGain = 2000
```

See also

Property

7.12.1.10 SignalIO::User0InputPol

Get or set the user0 input polarity.

Syntax

signalIO.**User0InputPol** [= pol]

Argument

Parameter	Type	Description
pol	LONG	Defines the user0 input polarity. See modes in the table below.

Remarks

Table of user0 input polarities values and description:

State No.	Name	Description
0	User0InputPol_Pos	Polarity is positive
1	User0InputPol_Neg	Polarity is negative

See also

None

7.12.1.11 SignalIO::User0OutputFlag

Get or set the user0 output flag.

Syntax

signalIO.**User0OutputFlag** [= flag]

Argument

Parameter	Type	Description
flag	LONG	Defines the user0 output flag. See modes in the table below.

Remarks

Table of user0 output flag values and description:

State No.	Name	Description
0	User0OutFlag_	Undefined
1	User0OutFlag_AddToTipVoltage	

See also

None

7.12.1.12 SignalIO::User0SetPoint

Get or set the user0 set point.

Syntax

signalIO.**User0SetPoint** [= setpoint]

Argument

Parameter	Type	Description
setpoint	DOUBLE	Defines the user0 set point [-1.0 .. +1.0]

Remarks

None

See also

None

7.12.1.15 SignalIO::UserDAC0

Get or set the user DAC0.

Syntax

signalIO.**UserDAC0** [= value]

Argument

Parameter	Type	Description
value	DOUBLE	Defines the user DAC0 value.

Remarks

None

See also

None

7.12.1.16 SignalIO::UserDAC1

Get or set the user DAC1.

Syntax

signalIO.**UserDAC1** [= value]

Argument

Parameter	Type	Description
value	DOUBLE	Defines the user DAC1 value.

Remarks

With C3000 the DAC1 value can only be set if the system.**SystemStateIdleDAC1Mode** is set to **SysStateIdleZ_AbsolutPos**

See also

System.SystemStateIdleDAC1Mode

7.13 Spec

The Spec class handles the microscope's spectroscopy subsystem.

Spectroscopy is a very powerful function to get physical sample properties. Also sample modification is possible on certain material.

The basic principle of spectroscopy is to modulate a output signal and measure the reaction of another signal. This results in a 2D line chart.

This is done at one position aver the surface or at different points along a line, then a 3D chart is the result.

A set of properties are defining the modulation output, the start and end point of the modulation, the modulation time and may more.

For more information about spectroscopy please refer to the Nanosurf Software Reference Manual.

A spectroscopy is first prepared by defining all the properties and the call Start. IsMeasuring is reporting if the measurement is in process. After the measurement StartCapture can copy the result into a image document or GetLine extract the data values.

Lithography or any other free tip movement can be done with StartMoveTipTo and IsMoving.

A object pointer to this class is provided by the Application.Spec object property.

Table of properties for Spec class:

Property name	Purpose
ActiveZController	Flag to select if the Z-Controller is stopped during a spectroscopy measurement
AddUserOutCToZStartPosition	Returns or set a flag if AddUserOutCToZStartPosition is activated
AutoCapture	Get or set the flag if auto capture is active
AutoRecalibrateProbe	Obsolete: Use **AutoRecalibrateProbeInterval** instead
AutoRecalibrateProbeInterval	Get or set the interval of the auto recalibration
BwdModDataPoints	Number of data points taken during a backward measurement
BwdModulationMode	Backward modulation mode
BwdModulationRange	Backward modulation range
BwdModulationStopMode	Backward modulation stop mode
BwdModulationStopValue	Backward modulation stop value
BwdModulationTime	Speed of the backward measurement
BwdMoveSpeed	Speed of the backward measurement
BwdPauseDatapoints	Number of data points taken during a backward pause
BwdPauseMode	Z-Controller state during backward pause
BwdPauseTime	Backward pause time
BwdSamplingRate	Sampling rate of the backward measurement
CurrentModulationPhase	The current modulation phase within a spectroscopy
EnableRelative	In relative mode the modulation values are added to the current output value
FwdModDatapoints	Number of data points taken during a forward measurement
FwdModulationMode	Forward modulation mode
FwdModulationRange	Forward modulation range
FwdModulationStopMode	Forward modulation stop mode
FwdModulationStopValue	Forward modulation stop value
FwdModulationTime	Speed of the forward measurement
FwdMoveSpeed	Speed of the forward measurement
FwdPauseDatapoints	Number of data points taken during a forward pause
FwdPauseMode	Z-Controller state during forward pause
FwdPauseTime	Forward pause time
FwdSamplingRate	Sampling rate of the forward measurement

Min	Min of dim N
Range	Range of dim N
LineMin	Min of line N
LinePoints	Number of points of line N
LineRange	Range of line N
ModulatedOutput	Defines the output which is modulated during spectroscopy
ModuleLevel	0 = Standard, 1 = Advanced
PositionListCount	Number of spectroscopy positions
Repetition	Repetition of measurement at each modulation point
RepetitionMode	Select repetition mode
Sequence	Number of modulation points between From and To position
SpecEndMode	Select whether the Z-Controller goes back active keeps its last Z - position after a spectroscopy.
StartOffset	Start value of the measurement
SyncOutMode	Returns or selects the mode of the synchronization output
XYMoveSpeed	Defines the speed of tip movement between modulation points
StartOffsetMoveSpeed	Defines the speed of movement to the start offset position

Table of methods for Spec class:

Method name	Purpose
Currentline	Retrieve the current spectroscopy sequence number
GetLine	Retrieve the data point values of a spectroscopy line
GetLine2	Retrieve the data point values of a spectroscopy line
IsCapturing	Retrieve the information whether a capture is prepared or not
IsMeasuring	Return True if spectroscopy sequence is in process
IsMoving	Return True if a tip movement is in process
ShowWindow	Controls the visibility of the imaging window
Start	Starts spectroscopy sequence
StartCapture	Prepare a data capture if measuring or do it immediately
StartMoveTipTo	Starts a tip movement to a destination position
Stop	Stops spectroscopy sequence
StopCapture	Clear a prepared data capture
Pause	Pauses the spectroscopy.

IsPaused	Returns if the spec is paused
IsFwdModulation IsBwdModulation IsFwdPauseIs BwdPause	Returns if the spectroscopy process is in a certain state.
ResumeLastPoint	Continue the spectroscopy after pause at last measured point
ResumeNextPoint	Continue the spectroscopy after pause at next point
ClearPositionList	Clear the spectroscopy position list
AddPosition	Add a spectroscopy position to the list of positions
AddPosition2	Add a spectroscopy position to the list of positions
AddPositions	Add a list of spectroscopy position to the list of positions
ForceBaseLinePos	Set the base line to a defined value

7.13.1 Properties

7.13.1.1 Spec::ActiveZController

Returns or set a flag to select if the Z-Controller is stopped during a spectroscopy measurement

Syntax

spec.**ActiveZController** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` to keep Z-Controller active during a spectroscopy measurement

Remarks

This flag selects if the Z-Controller is active during a spectroscopy measurement or not.

During normal spectroscopy measurement the Z-Controller is stopped in order to keep the tip position fixed during the measurement. In special cases it could be of interest to keep the Z-Controller active an measure the influence of a modulation to the z-position.

ActiveZController can only be activated if **ModulatedOutput** is not set to *ModOut_Z.*

If **ActiveZController** is activated the spectroscopy is measuring the *SigTopography* 1 too.

See also

Property ModulatedOutput, GetLine Method

Version info

Software v1.4.0 or later

7.13.1.3 Spec::AutoCapture

Returns or set a flag if AutoCapture is activated.

Syntax

spec.**AutoCapture** [= flag]

Setting

Argument	Type	Description
flag	boolean	Set to True AutoCapture is activated and set to False AutoCapture is deactivated.

Remarks

none

See also

7.13.1.4 Spec::AutoRecalibrateProbe

(Deprecated) Returns or set a flag to select if the auto recalibrate probe process should be performed before every spec.

Syntax

spec.**AutoRecalibrateProbe** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` is activated

Remarks

None

See also

Property AutoRecalibrateProbeInterval

7.13.1.5 Spec::AutoRecalibrateProbeInterval

Returns or set a value to select in what interval the auto recalibrate probe process should be performed before specs.

Syntax

spec.**AutoRecalibrateProbeInterval** [= val]

Setting

Argument	Type	Description
val	long	0 = Deactivated 1 = Performed before every spec N = Performed before every nth spec

Remarks

None

See also

7.13.1.6 Spec::BwdModDatapoints

Returns or set the number of measurement points of a backward modulation

Syntax

spec.**BwdModDatapoints** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of data points stored during a backward modulation. Minimum value is 2.

Remarks

This property defines how many data points are measured during a backward spectroscopy measurement.

See also

Property
Method Start

7.13.1.7 Spec::BwdModulationMode

Returns or set the modulation mode of the spectroscopy.

Syntax

spec.**BwdModulationMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode during a spectroscopy. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecModMode_FixedLength	Stop if the end point is reached.

1	SpecModMode_StopByValue	Stop if the modulation mode criteria's are meet.

See also

Property BwdModulationStopMode BwdModulationStopValue
Method

7.13.1.8 Spec::BwdModulationRange

Returns or set the backward modulation range.

Syntax

spec.**BwdModulationRange** [= range]

Setting

Argument	Type	Description
range	double	Defines the range of the backward modulation. [= range] range in m if modulation output "Z-Axis"

Remarks

none

See also

7.13.1.9 Spec::BwdModulationStopMode

Returns or set the mode of the backward modulation stop.

Syntax

spec.**BwdModulationStopMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the stop mode during a spectroscopy. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecStopMode_IsLessThan	No sync pulses are generated output is at Low-Lever.
1	SpecStopMode_IsGreaterThan	At each spectroscopy sample position a High-Pulse is generated

See also

7.13.1.10 Spec::BwdModulationStopValue

Returns or set the value of the backward modulation stop.

Syntax

spec.**BwdModulationStopValue** [= value]

Setting

Argument	Type	Description
value	double	Defines the stop value during a spectroscopy. [= value] value in V, m or N

Remarks

none

See also

7.13.1.11 Spec::BwdModulationTime

Returns or set the backward modulation time.

Syntax

spec.**BwdModulationTime** [= time]

Setting

Argument	Type	Description
time	double	Defines the backward modulation time. [= time] time in second

Remarks

none

See also

7.13.1.12 Spec::BwdMoveSpeed

Returns or set the backward move speed.

Syntax

spec.**BwdMoveSpeed** [= speed]

Setting

Argument	Type	Description
speed	double	Defines the move speed. [= speed] speed in m/s if modulation output "Z-Axis"

Remarks

none

See also

7.13.1.13 Spec::BwdPauseDatapoints

Returns or set the number of measurement points of a backward pause

Syntax

spec.**BwdPauseDatapoints** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of data points stored during a backward pause. Minimum value is 2.

Remarks

This property defines how many data points are measured during a backward spectroscopy pause measurement.

See also

Property
Method Start

7.13.1.14 Spec::BwdPauseMode

Returns or set the backward pause mode.

Syntax

spec.**BwdPauseMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the backward pause mode. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecPauseMode_ZOff	Keep last Z-Pos.
1	SpecPauseMode_ZOn	Z-Controller active.

See also

7.13.1.15 Spec::BwdPauseTime

Returns or selects the backward pause time.

Syntax

spec.**BwdPauseTime** [= time]

Setting

Argument	Type	Description
time	double	Defines the backward pause time. [= time] time in second

Remarks

none

See also

7.13.1.16 Spec::BwdSamplingRate

Returns or selects the backward sampling rate.

Syntax

spec.**BwdSamplingRate** [= value]

Setting

Argument	Type	Description
value	double	Defines the backward sampling rate. [= value] value in Hz

Remarks

none

See also

7.13.1.17 Spec::CurrentModulationPhase

Returns the current modulation phase.

Syntax

spec.**CurrentModulationPhase** [= phase] [read only]

Setting

Argument	Type	Description
phase	long	Defines the current modulation phase. See phase numbers in the table below.

Remarks

Phases may be skipped either because they don't exist or the time between property calls sees to missed phases.

Table of possible phases:

State No.	Name	Description
0	No Phase	Not in a specific phase, spec might not be running or between phases right now
1	Forward Modulation	In Forward Modulation phase
2	Forward Pause	In Forward Pause phase
3	Backward Modulation	In Backward Modulation phase
4	Backward Pause	In Backward Modulation phase

See also

7.13.1.18 Spec::EnableRelative

Returns or set a flag to select if end and start values are relative values or not.

Syntax

spec.**EnableRelative** [= flag]

Setting

Argument	Type	Description
flag	Boolean	Set to `True` is **StartValue** and **EndValue** properties should be interpreted as relative shifts to the current value.

Remarks

This flag selects if the values in **StartValue** and **EndValue** properties are interpreted as relative values to the current output value. A current output value is the value which the output had prior to the spectroscopy measurment.

Relative mode is used mainly to modulate the Z-Axis because normally not the absolute z value is interesting but the relative z value to the z-position of the topography. (e.g sample is at 3um Z controller output position, EnableRelative = `True`, StartValue= -1um, EndValue = 5um, resulting measurement is done from 2um to 8um)

See also

Property StartValue, ModulatedOutput

7.13.1.19 Spec::FwdModDatapoints

Returns or set the number of measurement points of a forward modulation

Syntax

spec.**FwdModDatapoints** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of data points stored during a forward modulation. Minimum value is 2.

Remarks

This property defines how many data points are measured during a forward spectroscopy measurement.

See also

Property
Method Start

7.13.1.20 Spec::FwdModulationMode

Returns or set the modulation mode of the spectroscopy.

Syntax

spec.**FwdModulationMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode during a spectroscopy. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecModMode_FixedLength	Stop if the end point is reached.
1	SpecModMode_StopByValue	Stop if the modulation mode criteria's are meet.

See also

Property FwdModulationStopMode FwdModulationStopValue
Method

7.13.1.21 Spec::FwdModulationRange

Returns or set the forward modulation range.

Syntax

spec.**FwdModulationRange** [= range]

Setting

Argument	Type	Description

range	double	Defines the range of the forward modulation. [= range] range in m if modulation output "Z-Axis"

Remarks

none

See also

7.13.1.22 Spec::FwdModulationStopMode

Returns or set the mode of the forward modulation stop.

Syntax

spec.**FwdModulationStopMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the stop mode during a spectroscopy. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecStopMode_IsLessThan	No sync pulses are generated output is at Low-Lever.
1	SpecStopMode_IsGreaterThan	At each spectroscopy sample position a High-Pulse is generated

See also

7.13.1.23 Spec::FwdModulationStopValue

Returns or set the value of the forward modulation stop.

Syntax

spec.**FwdModulationStopValue** [= value]

Setting

Argument	Type	Description
value	double	Defines the stop value during a spectroscopy. [= value] value in V, m or N

Remarks

none

See also

7.13.1.24 Spec::FwdModulationTime

Returns or set the forward modulation time.

Syntax

spec.**FwdModulationTime** [= time]

Setting

Argument	Type	Description
time	double	Defines the forward modulation time. [= time] time in second

Remarks

none

See also

7.13.1.25 Spec::FwdMoveSpeed

Returns or set the forward move speed.

Syntax

spec.**FwdMoveSpeed** [= speed]

Setting

Argument	Type	Description
speed	double	Defines the move speed. [= speed] speed in m/s if modulation output "Z-Axis"

Remarks

none

See also

7.13.1.26 Spec::FwdPauseDatapoints

Returns or set the number of measurement points of a forward pause

Syntax

spec.**FwdPauseDatapoints** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of data points stored during a backward pause. Minimum value is 2.

Remarks

This property defines how many data points are measured during a forward spectroscopy measurement.

See also

Property
Method Start

7.13.1.27 Spec::FwdPauseMode

Returns or set the forward pause mode.

Syntax

spec.**FwdPauseMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the forward pause mode. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecPauseMode_ZOff	Keep last Z-Pos.
1	SpecPauseMode_ZOn	Z-Controller active.

See also

7.13.1.28 Spec::FwdPauseTime

Returns or selects the forward pause time.

Syntax

spec.**FwdPauseTime** [= time]

Setting

Argument	Type	Description
time	double	Defines the forward pause time. [= time] time in second

Remarks

none

See also

7.13.1.29 Spec::FwdSamplingRate

Returns or selects the forward sampling rate.

Syntax

spec.**FwdSamplingRate** [= value]

Setting

Argument	Type	Description
value	double	Defines the forward sampling rate. [= value] value in Hz

Remarks

none

See also

7.13.1.32 Spec::LineMin

Return or set the min value for the spectroscopy line

Syntax

spec.**LineMin(**group, channel, line**) [**= min**]**

Argument

Paramete r	Type	Description
group	long	number of group
channel	long	number of channel
line	long	line number

Remarks

none

See also

Property LinePoints, LineRange
Method GetLine, GetLine2

7.13.1.33 Spec::LinePoints

Return or set the points value for the spectroscopy line

Syntax

spec.**LinePoints(**group, channel, line**) [**= points**]**

Argument

Parameter	Type	Description
group	long	number of group
channel	long	number of channel
line	long	line number

Remarks

none

See also

Property LineMin, LineRange
Method GetLine, GetLine2

7.13.1.34 Spec::LineRange

Return or set the range value for the spectroscopy line

Syntax

spec.**LineRange(**group, channel, line**) [**= range**]**

Argument

Parameter	Type	Description
group	long	number of group

channel	long	number of channel
line	long	line number

Remarks

none

See also

Property LineMin, LinePoints
Method GetLine, GetLine2

7.13.1.35 Spec::ModulatedOutput

Returns or selects the output of modulation.

Syntax

spec.**ModulatedOutput** [= output]

Setting

Argument	Type	Description
output	long	Defines the output signal which is modulated. See outputs in the table below.

Remarks

The spectroscopy modulation can be at different signal output. Which output is used is defined by this property.

Table of outputs for spectroscopy modulation :

Output No.	Name	Description
0	ModOut_Z	Z-Axis is modulated
1	ModOut_TipVoltage	The Tip Voltage output is modulated
2	ModOut_UserOut1	The User Output 1 is modulated
3	ModOut_UserOut2	The User Output 2 is modulated

See also

Method Start

Version info

More outputs defined in software v1.4.0 or later

7.13.1.36 Spec::ModuleLevel

Returns or selects the mode of the synchronization output.

Syntax

spec.**ModuleLevel** [= Level]

Setting

Argument	Type	Description
level	long	Defines the spectroscopy level.

Remarks

Table of possible modes:

State No.	Name	Description
0	Standard mode	Standard set of spectroscopy functionality.
1	Advanced mode	Advanced set of spectroscopy functionality. To use the advanced mode a key has to be purchased.

See also

7.13.1.37 Spec::PositionListCount

Returns the PositionListCount.

Syntax

spec.**PositionListCount** [= count]

Setting

Argument	Type	Description
count	Long	read only

Remarks

none

See also

7.13.1.38 Spec::Repetition

Returns or set the number of modulation cycles during a measurement.

Syntax

spec.**Repetition** [= count]

Setting

Argument	Type	Description
count	long	Defines the cycles of modulation per measurement. Minimum value is 1.

Remarks

This property defines how many modulations are repeated per spectroscopy measurement.

See also

7.13.1.39 Spec::RepetitionMode

Returns or selects the repetition mode.

Syntax

spec.**RepetitionMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the mode that is active. See outputs in the table below.

Remarks

Table of modes:

Output No.	Name	Description
0	RepetitionMode_List	Repeat all N points X times. 1 file per list.
1	RepetitionMode_Position	Repeat each position X times. 1 file per position.

See also

Method Repetition

7.13.1.40 Spec::Sequence

Returns or set the number of xy-points per spectroscopy sequence.

Syntax

spec.**Sequence** [= points]

Setting

Argument	Type	Description
points	long	Defines the number of xy-positions per spectroscopy sequence. Minimum value is 1.

Remarks

A complete spectroscopy is a sequence of measurements at different position over the sample.
The measurement positions are spread continuously along a line defined by the four properties.

See also

Property
Method Spec::Start, Spec::AddPosition, Spec::ClearPositionList

7.13.1.41 Spec::SpecEndMode

Returns or set the spectroscopy end mode.

Syntax

spec.**SpecEndMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the spectroscopy end mode. See mode numbers in the table below.

Remarks

Table of possible modes:

State No.	Name	Description
0	SpecEndMode_StayLastZPos	Keep last Z-Pos.
1	SpecEndMode_Approached	Z-Controller active.

See also

7.13.1.42 Spec::StartOffset

Returns or set the start value of the measurement

Syntax

spec.**StartOffset** [= value]

Setting

Argument	Type	Description
value	double	Defines the start value of the spectroscopy modulation.

Remarks

none

See also

7.13.1.44 Spec::SyncOutMode

Returns or selects the mode of the synchronization output.

Syntax

spec.**SyncOutMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the signal generated at the synchronization output during a spectroscopy. See mode numbers in the table below.

Remarks

During a spectroscopy modulation different synchronisation signal can be generated at the sync output.
The sync pulse durations is about 4us.

Table of possible modes:

State No.	Name	Description
0	SyncOut_NoSync	No sync pulses are generated output is at Low-Lever.
1	SyncOut_PulsSample	At each spectroscopy sample position a High-Pulse is generated
2	SyncOut_PulsBegin	At the beginning of spectroscopy measurement a High-Pulse is generated
3	SyncOut_PulsEnd	At the end of spectroscopy measurement a High-Pulse is generated
4	SyncOut_PulsBeginAndEnd	At the beginning and the end of spectroscopy measurement a High-Pulse is generated
5	SyncOut_LevelBeginToEnd	A High level is generated during the spectroscopy measurement.

See also

Description of Sync-Output in the Operating Manual

Version info

Software v1.4.0 or later

7.13.2 Methods

7.13.2.1 Spec::AddPosition

Add a spectroscopy position to the list of positions.

Syntax

spec.**AddPosition(**x, y, z**)**

Result

Parameter	Type	Description
x	double	X-Axis component of the destination position. Unit in meter [m]
y	double	Y-Axis component of the destination position. Unit in meter [m]
z	double	Z-Axis component of the destination position. Unit in meter [m]

Remarks

This method adds a spectroscopy position to the position list. The coordinate system of the destination position is the scanner coordinate system. I.e. the position (0,0,0) is the center position of the scanner.

Example

```
' pos(x,y,z) = (1um,2um,0um)
objSpec.AddPosition 1e-6,2e-6,0
```

See also

Method ClearPositionList,

7.13.2.4 Spec::ClearPositionList

Clear the spectroscopy position list.

Syntax

spec.**ClearPositionList**

Result

none

Remarks

This method clears the spectroscopy position list.

Example

```
' clear position list
objSpec.ClearPositionList
```

See also

Method AddPosition

7.13.2.5 Spec::Currentline

Returns the number of the last measured spectroscopy line.

Syntax

line = *spec*.**Currentline**

Result

Result	Type	Description
line	long	The last measured spectroscopy line number.

Remarks

This method is returning the number of the last measured spectroscopy line.
A complete spectroscopy sequence is composed of spectroscopy data lines. At each Sequence point a spectroscopy data line is stored. A spectroscopy data line is composed of two spectroscopy modulation data array. One for ForwardSpectroscopy and one for BackwardSpectroscopy. Line zero is the first sequence data line and the last has number **Sequence** - 1.

This method can be used to monitor which spectroscopy lines is currently measured during a spectroscopy process.

See also

Property Sequence
Method Start, GetLine, IsMeasuring

7.13.2.7 Spec::GetLine

Returns a string of data values of a spectroscopy data line.

Syntax

array = *spec*.**GetLine(***group,channel,specine,filter,conversion***)**

Argument

Paramete r	Type	Description
group	long	number of group
channel	long	number of channel
specline	long	spec line number
filter	long	index of mathematical filter to be used
conversion	long	index of conversion type of results

Result

Result	Type	Description
array	String	Character string with comma separated values of all the values of the scan line

Remarks

This method returns a string of data values of a spectroscopy data line. Any signal of a measured spectroscopy sequence can be extracted and the data values can be processed with the same filters as available for the user in the "Chart Toolbar". The result is in a comma separated string in different numerical formats.

The first two arguments *group* and *channel* selects the matrix of a specific signal.

Table of group numbers:

Group No.	Group Name	Description
0	Group_ForwardSpec	Selects signal channels of forward spectroscopy modulation
1	Group_BackwardSpec	Selects signal channels of backward spectroscopy modulation
2	Group_ForwardSpecPaus e	Selects signal channels of forward pause spectroscopy
3	Group_BackwardSpecPa	Selects signal channels of backward pause spectroscopy

	use	

In each group there are different signal channels. To get the values of a specific signal one has to know the channel number. If a certain channel is available in a measurement depends on the active operating mode during the measurement.

Table of channel numbers:

Channel No.	Signal Name	Description
0	SigDeflection	Static cantilever deflection signal
1	SigTopography	Z-Topography signal
2	SigAmplitude	Cantilever vibrating amplitude signal
3	SigPhase	Cantilever phase shift signal
4	SigUser	User's defined ADC input signal

The argument *specline* is the number of the sequence data line to extract. 0 is the first sequnece line and property **Sequence** -1 the last one.

The argument *filter* and *conversion* defines the data processing algorithm and formating to be used.
See parameter tables at Data.GetLine Method.

Example

```
' get deflection of forward spec line of sequence 5 with plane fit filter active
and in [m]
specline = objSpec.GetLine(0,0,5,2,1)
datararray = Split(specline,",")

' get user input signal of current scan line, no filter as 16bit values
specline = objSpec.GetLine(0,5,objSpec.Currentline,0,0)
```

See also

Property Sequence
Method Start, Currentline

7.13.2.8 Spec::GetLine2

Returns a VARIANT array of data values of a spectroscopy data line.

Syntax

array = *spec*.**GetLine2(***group, channel, specline, filter, conversion***)**

Argument

Parameter	Type	Description
group	long	number of group
channel	long	number of channel
specline	long	spec line number
filter	long	index of mathematical filter to be used
conversion	long	index of conversion type of results

Result

Result	Type	Description
array	VARIANT	VARIANT array with values of all the values of the spec line

Remarks

This method returns a string of data values of a spectroscopy data line. Any signal of a measured spectroscopy sequence can be extracted and the data values can be processed with the same filters as available for the user in the "Chart Toolbar". The result is in a comma separated string in different numerical formats.

The first two arguments *group* and *channel* selects the matrix of a specific signal.

Table of group numbers:

Group No.	Group Name	Description
0	Group_ForwardSpec	Selects signal channels of forward spectroscopy modulation
1	Group_BackwardSpec	Selects signal channels of backward spectroscopy modulation
2	Group_ForwardSpecPause	Selects signal channels of forward pause spectroscopy
3	Group_BackwardSpecPause	Selects signal channels of backward pause spectroscopy

In each group there are different signal channels. To get the values of a specific signal one has to know the channel number. If a certain channel is available in a measurement depends on the active operating mode during the measurement.

Table of channel numbers:

Channel No.	Signal Name	Description
0	SigDeflection	Static cantilever deflection signal
1	SigTopography	Z-Topography signal
2	SigAmplitude	Cantilever vibrating amplitude signal
3	SigPhase	Cantilever phase shift signal
4	SigUser	User's defined ADC input signal

The argument *specline* is the number of the sequence data line to extract. 0 is the first sequnece line and property **Sequence** -1 the last one.

The argument *filter* and *conversion* defines the data processing algorithm and formating to be used.
See parameter tables at Data.GetLine Method.

Example

```
' get deflection of forward spec line of sequence 5 with plane fit filter active
and in [m]
specline = objSpec.GetLine(0,0,5,2,1)
datararray = Split(specline,",")

' get user input signal of current scan line, no filter as 16bit values
specline = objSpec.GetLine(0,5,objSpec.Currentline,0,0)
```

See also

Property Sequence
Method Start, Currentline

7.13.2.9 Spec::IsCapturing

Returns if a capture is pending or not.

Syntax

flag = *spec*.**IsCapturing**

Result

Result	Type	Description
flag	Boolean	Returns `True` if a capture is pending

Remarks

This method is returing `True` if a capture is pending.

Example

```
If  objSpec.IsCapturing Then
  objSpec.StopCapture
End If
```

See also

Method StartCapture, StopCapture

7.13.2.10 Spec::IsMeasuring

Returns if a spectroscopy measurement is in process or not.

Syntax

flag = *spec*.**IsMeasuring**

Result

Result	Type	Description
flag	Boolean	Returns `True` if a spectroscopy measurement is in process

Remarks

This method is returning `True` if a spectroscopy measurement is currently running.

Example

```
' measure
objSpec.Start
Do While objSpec.IsMeasuring : Loop

' copy image date
objSpec.StartCapture
```

See also

Method Start

7.13.2.12 Spec::IsMoving

Returns if a tip movement by **StartMoveTipTo** is in process or not.

Syntax

flag = *spec*.**IsMoving**

Result

Result	Type	Description
flag	Boolean	Returns `True` if a tip movement is in process

Remarks

This method is returning `True` if a tip movement started by StartMoveTipTo is currently running.

If fast tip movement is needed by script control please make sure that the StatusReadDelay property of the Application class is set to zero!

Example

```
' move tip to pos(x,y,z) = (1um,2um,0um)
objApp.StatusReadDelay = 0.0
objSpec.StartMoveTipTo 1e-6,2e-6,0
Do While objSpec.IsMoving : Loop
```

See also

Method StartMoveTipTo, Property objApp.StatusReadDelay

7.13.2.16 Spec::ShowWindow

Defines the display style of the Spectroscopy window.

Syntax

spec.**ShowWindow(**style**)**

Arguments

Argument	Type	Description
style	short	Visibility style number

Result

None.

Remarks

The **ShowWindow** method sets the visibility state of the window.

Available styles see Doc.ShowWindow Method

Example

```
objSpec.ShowWindow(0) ' hide the window
```

See also

None.

Version info

Software v1.4.0 or later

7.13.2.17 Spec::Start

Starts spectroscopy sequence.

Syntax

spec.**Start**

Remarks

This method is starting the spectroscopy sequence. It can be aborted at the end of a modulation by method **Stop**. If a spectroscopy measurement is running read method **IsMeasuring**.

The modulation output, the start and end values and all the other properties of spectroscopy class should be predefined prior the start but some can be changed also during spectroscopy. A call to **StartCapture** creates a new document after the spectroscopy measurement is finished.

During a spectroscopy modulation the z feedback controller is set to Loopmode_Freeze mode.

Please use the command **AddPosition** to add a position where a spectroscopy measurment should take place. Use the command **ClearPositionList** to clear the position list.

Example

```
' do spec
objSpec.Start
Do While objSpec.IsMeasuring : Loop
```

See also

Method IsMeasuring
Class OperatingMode, ZController

7.13.2.19 Spec::StartCapture

Create a new image document.

Syntax

spec.**StartCapture**

Remarks

This method copies the measured spectroscopy data to a new image document. If a spectroscopy measurment process is running at the time **StartCapture** is called the copy is delayed until the sequence is fully measured. A pending capture can be called with **StopCapture**. If a capture is pending read method **IsCapturing**.

Example

```
' start spec
objSpec.Start

' prepare image copy
objScan.StartCapture

' wait until copy is taken at end of sequnece
Do While objSpec.IsCapturing : Loop
objApp.SaveDocument("myspec.nid")
```

See also

Method StopCapture, IsCapturing
Method Application.SaveDocument

7.13.2.20 Spec::StartMoveTipTo

Move the tip from the current position to a destination coordinate.

Syntax

spec.**StartMoveTipTo(***x*,*y*,*z***)**

Argument

Parameter	Type	Description
x	double	X-Axis component of the destination position. Unit in meter [m]
y	double	Y-Axis component of the destination position. Unit in meter [m]
z	double	Z-Axis component of the destination position. Unit in meter [m]

Remarks

This method moves the tip from the current position to a target position. The position is defined in the scanners physical reference coordinate system. The move speed is approximately defined the factor

Move speed = objScan.ImageWidth / objScan.Scantime

Attention: If the Z controller is in Loopmode_Run then the Z-Position is never exactly the value of the z argument but a superimpose of the Z-Argument and the z-feedback output signal.

The method only starts the movement and return immediately. To wait until the movement is finished read **IsMoving** method.

If fast tip movement is needed by script control please make sure that the StatusReadDelay property of the Application class is set to zero!

For more information about the physical reference coordinate system please refer to the Nanosurf Software Reference Manual.

Example

```
' Simple lithography.
' -------------------
' Scratch a square and image afterward

normalforce  =  30e-9 'N
scratchforce = 200e-9 'N

 ' prepare operating mode
objApp.StatusReadDelay  = 0.0 ' No delay to get full writing speed
objOpMode.OperatingMode = 1   ' Static Force mode
objOpMode.Cantilever    = 0   ' CONTR Lever

' move to start point
objZCtrl.SetPoint = normalforce
objSpec.StartMoveTipTo -5e-6,-5e-6,0
Do While objSpec.IsMoving : Loop

' scratch the square
objZCtrl.SetPoint = scratchforce
```

```
objSpec.StartMoveTipTo 5e-6,-5e-6,0
Do While objSpec.IsMoving : Loop

objSpec.StartMoveTipTo 5e-6,5e-6,0
Do While objSpec.IsMoving : Loop

objSpec.StartMoveTipTo -5e-6,5e-6,0
Do While objSpec.IsMoving : Loop

objSpec.StartMoveTipTo -5e-6,-5e-6,0
Do While objSpec.IsMoving : Loop

' release scratch a square
objZCtrl.SetPoint = normalforce

' image
objScan.ImageSize 30e-6,30e-6
objScan.StartFrameUp
Do While objScan.IsScanning : Loop
```

See also

Method IsMoving, Property objApp.StatusReadDelay

7.13.2.21 Spec::Stop

Stops spectroscopy measurement immediately.

Syntax

spec.**Stop**

Remarks

This method stops any spectroscopy process immediately after the current modulation is finished.
A possible pending capture flag is also cleared and no document is created.

Example

```
' start scan
objSpec.Start

' do something else ...

' finish immediately
objSpec.Stop
```

See also

Method Start, StartCapture

7.13.2.22 Spec::StopCapture

Cancel a pending capture

Syntax

spec.**StopCapture**

Remarks

This method cancel a pending capture. If a capture is pending read method **IsCapturing**.

Example

```
' start sequence
objSpec.Start

' prepare data copy
objScan.StartCapture

' do something

If  objSpec.IsCapturing Then
  objSpec.StopCapture
End If
```

See also

Method StartCapture, IsCapturing

7.14 SPMCtrlDataStream

The SPM control data stream handles access to the SPM data stream subsystem.

A object pointer to this class is provided by the SPMCtrlManager.DataStream object property.

Table of properties for the SPMCtrlDataStream class:

Property name	Purpose

MonitoringChannelMap	Returns a object pointer to the single LogicalUnit class object
MonitoringChannelUnits	Returns a object pointer to the single DataBuffer class object

Table of methods for the SPMCtrlDataStream class:

Method name	Purpose
ActivateSocketStreamingInterface	Activates the socket streaming interface on given port number

7.14.1 Methods

7.14.1.1 SPMCtrlDataStream::ActivateSocketStreamingInterface

Activates the socket streaming interface.

Syntax

objSPMCtrlDataStream.**ActivateSocketStreamingInterface(***nPort***)**

Argument

Parameter	Type	Description
nPort	long	Socket port to use for socket server (10000<nPort<60000) or 0

Remarks

This method opens a socket. The port must be free for this to be successful.
0 will deactivate the socket streaming interface.

Example

```
' Activate the socket interface on port 30003
objSPMCtrlDataStream.ActivateSocketStreamingInterface = "30003"

' Deactivate the socket interface
objSPMCtrlDataStream.ActivateSocketStreamingInterface = "0"
```

Version info

Software v3.8.0.0 or later

7.14.2 Properties

7.14.2.1 SPMCtrlDataStream::MonitoringChannelMap

Returns or sets a variant array of integers with the channel id's in it.

Syntax

objSPMCtrlDataStream.**MonitoringChannelMap** [= flag]

Setting

Argument	Type	Description
flag	VARIANT long array	Array of integers of channel id's

Remarks

Channel Id's:

```
//  CI_Deflection = 0,
//  CI_Friction = 1, // Lateral
//  CI_UserIn3 = 2, // User In A / Tip Current
//  CI_UserIn2 = 3, // User In B
//  CI_UserIn1 = 4,
//  CI_Amplitude_Alyzr1 = 5,
//  CI_Phase_Alyzr1 = 6,
//  CI_LockInX_Alyzr1 = 7,
//  CI_LockInY_Alyzr1 = 8,
//  CI_AmplitudeCtrlOut_Alyzr1 = 9,
//  CI_PhaseCtrlOut_Alyzr1 = 10,
//  CI_Amplitude_Alyzr2 = 11,
//  CI_Phase_Alyzr2 = 12,
//  CI_LockInX_Alyzr2 = 13,
//  CI_LockInY_Alyzr2 = 14,
//  CI_AmplitudeCtrlOut_Alyzr2 = 15,
//  CI_PhaseCtrlOut_Alyzr2 = 16,
//  CI_ZAxisSensor = 17,
//  CI_XAxis = 18,
//  CI_YAxis = 19,
//  CI_ZAxis = 20,
//  CI_UserOutC = 21,
//  CI_TipVoltageOutput = 22,
//  CI_ApproachMotor = 23,
//  CI_XAxisSensor = 24,
//  CI_YAxisSensor = 25,
```

See also

Property MonitoringChannelUnits

Version info

Software v3.5.0.0 or later

7.14.2.2 SPMCtrlDataStream::MonitoringChannelUnits

Returns a variant array of strings with the unit names in it. This property is read only.

Syntax

objSPMCtrlDataStream.**MonitoringChannelUnits** [= flag] [read only]

Setting

Argument	Type	Description
flag	VARIANT string array	Array of strings of units names

Remarks

The monitoring channel map determines the layout of the units contained in the array.

See also

Property MonitoringChannelMap

Version info

Software v3.5.0.0 or later

7.15 SPMCtrlManager

The SPM control manager handles access to the SPM subsystem.

A object pointer to this class is provided by the Application.SPMCtrlManager object property.

Table of properties for the SPMCtrlManager class:

Property name	Purpose
LogicalUnit	Returns a object pointer to the single LogicalUnit class object

DataBuffer	Returns a object pointer to the single DataBuffer class object
DataStream	Returns a object pointer to the single DataStream class object
MacroCmd	Returns a object pointer to the single MacroCmd class object

7.15.1 Properties

7.15.1.1 SPMCtrlManager::DataStream

Returns a dispatch pointer to the sub class DataStream. This property is read only.

Syntax

application.**DataStream** [read only]

Result

The **DataStream** property is returning a pointer to the IDispatch interface of the SPMCtrlDataStream object.

Remarks

Only one single instance exists of the SPMCtrlDataStream object. All successive read of this property will return the same IDispatch pointer.

It is good practice to free the object reference after usage. See the example on how to do this.

Example

```
' create object

Dim objApp             : Set objApp            = Nanosurf_C3000.Application
Dim objSPMCtrlManager : Set objSPMCtrlManager = objApp.SPMCtrlManager
Dim objSPMDataStream   : Set objSPMDataStream   = objSPMCtrlManager.DataStream

' do something with the object

' clean up

objSPMDataStream  = nul : Set objSPMDataStream  = Nothing
objSPMCtrlManager = nul : Set objSPMCtrlManager = Nothing
objApp            = nul : Set objApp            = Nothing
```

See also

Class SPMCtrlDataStream

7.16 Stage

The Stage class handles the stage subsystem.

A object pointer to this class is provided by the Application.Stage object property.

Table of properties for the Stage class:

Property name	Purpose
HasInstance	Says if there is a stage instance
HasPositionReached	Says if the last move has reached its destination
IsReferenced	Says if the stage is referenced

Table of methods for the Stage class:

Method name	Purpose
AppendToMoveTransaction	Append move operation to transaction
ClearMoveTransaction	Clear everything from move transaction
CloseInstance	Close stage instance
CommitMoveTransaction	Commit move transaction
EmergencyStop	Stops all stage movement with emergency stop configuration
GetAxisName	Returns the name of given axis
GetAxisPosition	Returns the position orthogonal corrected of given axis
GetAxisPositionMonitoring	Returns the position orthogonal corrected & monitor inverted of given axis
GetAxisRange	Returns possible range of the axis
GetAxisUnit	Returns the unit of given axis
GetAxisValue	Returns the value (position) of given axis
GetCurrentAxisZeroPosition	Returns the given axis zero position
GetSpeedPercent	Returns the current speed percent value
GetState	Returns the current stage state
GetTransactionCommitCount	Returns the number of committed transactions
Lock	Locks stage if idle
ReferenceSearch	Performs a reference search
SetAxisZero	Sets the current position of axis zero (no move)
SetSpeedPercent	Sets the speed percent value

SetTransactionDependentApproachMove	Sets the transaction to apply the dependent approach move
SetTransactionNoOrthoCorrection	Sets the transaction to not apply orthogonal corrections
SetTransactionNoSecureMove	Sets the transaction to not perform secure moves
SetupInstance	Creates a stage instance from a configuration file
SetZero	Sets the current position zero (no move)
SpecialOperationAxis	Performs a special operation on an axis
SpecialOperationController	Performs a special operation on a controller
SpecialOperationView	Performs a special operation on the stage view
Stop	Stops all stage movement
Unlock	Unlocks the stage if locked

7.16.1 Properties

7.16.1.1 Stage::HasInstance

Returns a flag which says if there is an stage instance or not. This property is read only.

Syntax

objStage.**HasInstance** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if there is a stage instance

Remarks

This flag concerns the main stage sub system instance. There can only be one such instance. This flag must be true for most other other properties and methods to be used. If it is not, an instance can be setup with **SetupInstance**.

See also

Method SetupInstance, CloseInstance

Version info

Software v3.5.0.0 or later

7.16.1.2 Stage::HasPositionReached

Returns a flag which says if the last move action has reached the specified position or not. This property is read only.

Syntax

objStage.**HasPositionReached** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if the position was reached

Remarks

This flag says if the stage is referenced. This means that the absolute physical position is known. This flag must be `True` for most movement actions to work properly. This flag can only be checked after a move. During a move the value is undefined.

See also

Method CommitMoveTransaction, AppendToMoveTransaction, Stop, EmergencyStop

Version info

Software v3.5.0.0 or later

7.16.1.3 Stage::IsReferenced

Returns a flag which says if the stage is referenced or not. This property is read only.

Syntax

objStage.**IsReferenced** [= flag] [read only]

Setting

Argument	Type	Description
flag	Boolean	`True` if the stage is referenced

Remarks

This flag says if the stage is referenced. This means that the absolute physical position is known. This flag must be true for most movement actions to work properly.

See also

Method ReferenceSearch, GetStage

Version info

Software v3.5.0.0 or later

7.16.2 Methods

7.16.2.1 Stage::AppendToMoveTransaction

This method appends a move command to the move transaction.

Syntax

objStage.**AppendToMoveTransaction(nAxisId, fNewValue, bRelativeValue)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis identifier
fNewValue	double	Position to move to
bRelativeValue	boolean	Says if fNewValue is relative to current position or absolute

Result

None

Remarks

The **AppendToMoveTransaction** method adds a move command to the move transaction. If a move transaction has multiple move commands for the same axis, the last one counts (even if this should be avoided anyway). The transaction list can be cleared with **ClearMoveTransaction**.

See also

Method ClearMoveTransaction, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.2 Stage::ClearMoveTransaction

This method clears the current move transaction of all entries.

Syntax

objStage.**ClearMoveTransaction()**

Argument

None

Result

None

Remarks

The **ClearMoveTransaction** method clears the current move transaction of all entries. Everything added with **AppendToMoveTransaction** is lost.

See also

Method AppendToMoveTransaction, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.3 Stage::CloseInstance

This method closes down the stage sub system instance.

Syntax

objStage.**CloseInstance()**

Argument

None

Result

None

Remarks

The **CloseInstance** method closes down the stage sub system instance. If there is no instance this is noop.

See also

Method SetupInstance, Property HasInstance

Version info

Software v3.5.0.0 or later

7.16.2.4 Stage::CommitMoveTransaction

This method commits all appended move commands.

Syntax

objStage.**CommitMoveTransaction()**

Argument

None

Result

None

Remarks

The **CommitMoveTransaction** method commits all appended move commands. The appended move commands are not cleared automatically. Depending on the stage hardware setup and configuration, all move commands are started as concurrent as possible. If the move transaction is empty this is noop.

See also

Method AppendToMoveTransaction, ClearMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.5 Stage::EmergencyStop

This method stops all stage movement with emergency parameters.

Syntax

objStage.**EmergencyStop()**

Argument

None

Result

None

Remarks

The **EmergencyStop** method configures special parameters and stops all stage axis in their movement. Depending on the stage hardware and configuration a stop may take a long time. To stop as fast as possible an emergency stop configuration is applied before stopping.

See also

Method Stop, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.6 Stage::GetAxisName

This method returns the axis name of given axis.

Syntax

retval = *objStage*.**GetAxisName(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	String	Display name of axis

Remarks

The **GetAxisName** method returns the display name of the given virtual axis. This value is directly read from the configuration file.

See also

Method GetAxisUnit, GetAxisValue

Version info

Software v3.5.0.0 or later

7.16.2.7 Stage::GetAxisPosition

This method returns the axis position with orthogonal correction of given axis.

Syntax

retval = *objStage*.**GetAxisPosition(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Position of axis

Remarks

The **GetAxisPosition** method returns the value of the given virtual axis with the orthogonal correction calculated in. This value is read from the controller or cache. It is not monitor inverted. For this see the **GetAxisPositionMonitoring** method.

See also

Method GetAxisName, GetAxisUnit, GetAxisValue, GetAxisPositionMonitoring

Version info

Software v3.8.5.6 or later

7.16.2.8 Stage::GetAxisPositionMonitoring

This method returns the axis position with orthogonal correction and monitor inversion of given axis.

Syntax

retval = *objStage*.**GetAxisPositionMonitoring(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Position of axis

Remarks

The **GetAxisPositionMonitoring** method returns the value of the given virtual axis with the orthogonal correction calculated in and monitor inversion. This value is read from the controller or cache.

See also

Method GetAxisName, GetAxisUnit, GetAxisValue, GetAxisPosition

Version info

Software v3.8.5.6 or later

7.16.2.9 Stage::GetAxisRange

This method returns the axis travel range of given axis.

Syntax

retval = *objStage*.**GetAxisRange(nAxisId)**

Argument

Parameter	Type	Description

nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Range of axis

Remarks

The **GetAxisRange** method returns the range (upper limit - lower limit) of the given virtual axis. This value is read from the configuration (stagex).

See also

Method GetAxisValue, GetCurrentAxisZeroPosition

Version info

Software v3.8.2.0 or later

7.16.2.10 Stage::GetAxisUnit

This method returns the axis unit of given axis.

Syntax

retval = *objStage*.**GetAxisUnit(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	String	Display unit of axis

Remarks

The **GetAxisUnit** method returns the display unit of the given virtual axis. This value is derived from the axis type.

See also

Method GetAxisName, GetAxisValue

Version info

Software v3.5.0.0 or later

7.16.2.11 Stage::GetAxisValue

This method returns the axis value of given axis.

Syntax

retval = *objStage*.**GetAxisValue(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Value of axis

Remarks

The **GetAxisValue** method returns the value of the given virtual axis. This value is read from the controller or cache. It is not orthogonal corrected and not monitor inverted. For those see the **GetAxisPosition** & **GetAxisPositionMonitoring** methods.

See also

Method GetAxisName, GetAxisUnit, GetAxisPosition, GetAxisPositionMonitoring

Version info

Software v3.5.0.0 or later

7.16.2.12 Stage::GetCurrentAxisZeroPosition

This method returns the current axis zero position (offset) of given axis.

Syntax

retval = *objStage*.**GetCurrentAxisZeroPosition(nAxisId)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis id

Result

Result	Type	Description
retval	double	Zero position (offset) of axis

Remarks

The **GetCurrentAxisZeroPosition** method returns the zero position of the given virtual axis. This value is read from the virtual axis and can change every time the axis is referenced or the axes are zeroed.

See also

Method GetAxisValue, GetAxisRange

Version info

Software v3.8.2.0 or later

7.16.2.13 Stage::GetSpeedPercent

This method returns the global stage speed in percent.

Syntax

retval = *objStage*.**GetSpeedPercent()**

Argument

None

Result

Result	Type	Description
retval	int32	Global stage speed in percent

Remarks

The **GetSpeedPercent** method returns the global stage speed in percent.

See also

Method SetSpeedPercent, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.14 Stage::GetState

This method returns the stage state.

Syntax

retval = *objStage*.**GetState()**

Argument

None

Result

Result	Type	Description
retval	int32	Current global stage state

Remarks

The **GetAxisValue** method returns the current global stage state.
Table of possible states:

State #	Name	Description
1	IdleUnreferenced	In idle state without absolute physical reference.
2	Idle	In idle state with absolute physical reference.
3	Move	Stage is moving. Either a manual move, a "move to" or a reference search.

See also
-

Version info

Software v3.5.0.0 or later

7.16.2.15 Stage::GetTransactionCommitCount

This method returns the committed transaction count.

Syntax

retval = *objStage*.**GetTransactionCommitCount()**

Argument

None

Result

Result	Type	Description
retval	int32	Transaction commit count

Remarks

The **GetTransactionCommitCount** method returns the committed transaction count. This count increases with every move commit as soon as the state changes from idle to non idle. GetState and this count are atomic and thread safe.

See also

Method_CommitMoveTransaction, GetState

Version info

Software v3.8.2.0 or later

7.16.2.16 Stage::Lock

This method locks the stage when idle.

Syntax

objStage.**Lock()**

Argument

None

Result

None

Remarks

The **Lock** method locks the stage system when idle. Unlock is needed to use the stage system again. No other action is possible.

See also

Method Unlock

Version info

Software v3.8.8.3 or later

7.16.2.17 Stage::ReferenceSearch

This method starts a reference search.

Syntax

objStage.**ReferenceSearch()**

Argument

None

Result

None

Remarks

The **ReferenceSearch** method starts a reference search. The stage must be idle to perform a reference search. **GetState** can be used to see if the reference was found.

See also

Method Stop, GetState, Property IsReferenced

Version info

Software v3.5.0.0 or later

7.16.2.18 Stage::SetAxisZero

This method sets given axis coordinate to zero. There is no move, the internal coordinate offset is changed.

Syntax

objStage.**SetAxisZero()**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual axis identifier

Result

None

Remarks

The **SetAxisZero** method sets given axis coordinate to zero. There is no move, the internal coordinate offset is changed.

See also

Method GetAxisValue, GetCurrentAxisZeroPosition, SetZero

Version info

Software v3.8.7.0 or later

7.16.2.19 Stage::SetSpeedPercent

This method sets the global stage speed in percent.

Syntax

objStage.**SetSpeedPercent(nSpeedPercent)**

Argument

Parameter	Type	Description
nSpeedPercent	int32	New speed in percent from 1-100

Result

None

Remarks

The **SetSpeedPercent** method sets the new global stage speed in percent. This new speed is first used in the next move transaction commit.

See also

Method GetSpeedPercent, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.20 Stage::SetTransactionDependentApproachMove

This method configures the move transaction to add a dependent approach axis move if necessary.

Syntax

objStage.**SetTransactionDependentApproachMove()**

Argument

None

Result

None

Remarks

The **SetTransactionDependentApproachMove** method configure the move transaction to add a dependent approach axis move if necessary. This move is configured with the "DependentMoveFactor" attribute on the Approach node in the stagex configuration.

See also

Method AppendToMoveTransaction, ClearMoveTransaction, CommitMoveTransaction

Version info

Software v3.8.8.0 or later

7.16.2.21 Stage::SetTransactionNoOrthoCorrection

This method configures the move transaction to not apply orthogonal correction when.

Syntax

objStage.**SetTransactionNoOrthoCorrection()**

Argument

None

Result

None

Remarks

The **SetTransactionNoOrthoCorrection** method configure the move transaction to not apply orthogonal correction when moving. This only applies if an orthogonal relation is setup between two axes in the stagex configuration.

See also

Method AppendToMoveTransaction, ClearMoveTransaction, CommitMoveTransaction

Version info

Software v3.8.5.6 or later

7.16.2.22 Stage::SetTransactionNoSecureMove

This method configures the move transaction to not apply orthogonal correction when.

Syntax

objStage.**SetTransactionNoSecureMove()**

Argument

None

Result

None

Remarks

The **SetTransactionNoSecureMove** method configure the move transaction to not do secure moves configured in the stagex configuration.

See also

Method AppendToMoveTransaction, ClearMoveTransaction, CommitMoveTransaction

Version info

Software v3.8.5.6 or later

7.16.2.23 Stage::SetupInstance

This method creates the stage sub system instance with a given configuration.

Syntax

objStage.**SetupInstance(***strFilename***)**

Argument

Parameter	Type	Description
strFilename	String	Stage configuration filename to setup stage instance with

Result

None

Remarks

The **SetupInstance** method creates the stage sub system instance. If there is already an instance, it will be closed before creating the new one. The given file name supplies the configuration for the new instance.

See also

Method CloseInstance, Property HasInstance

Version info

Software v3.5.0.0 or later

7.16.2.24 Stage::SetZero

This method sets every axis coordinate to zero. There is no move, the internal coordinate offset is changed.

Syntax

objStage.**SetZero()**

Argument

None

Result

None

Remarks

The **SetZero** method sets every axis coordinate to zero. There is no move, the internal coordinate offset is changed.

See also

Method GetAxisValue, GetCurrentAxisZeroPosition, SetAxisZero

Version info

Software v3.8.2.0 or later

7.16.2.25 Stage::SpecialOperationAxis

This method performs a special operation on a stage axis.

Syntax

objStage.**SpecialOperationAxis(nAxisId, nId, fValue, nValue, strValue)**

Argument

Parameter	Type	Description
nAxisId	int32	Virtual Axis id
nId	int32	Special operation id
fValue	double [out]	Double value pointer
nValue	int32 [out]	Integral value pointer
strValue	String [out]	String value pointer

Result

None

Remarks

The **SpecialOperationAxis** method performs a special operation on a stage axis. Special operations are axis type dependent and are documented separately and customer specific.

See also

Method SpecialOperationView, SpecialOperationController

Version info

Software v3.5.0.0 or later

7.16.2.26 Stage::SpecialOperationController

This method performs a special operation on a stage controller.

Syntax

objStage.**SpecialOperationController(nControllerId, nId, fValue, nValue, strValue)**

Argument

Parameter	Type	Description
nControllerId	int32	Hardware Controller id
nId	int32	Special operation id
fValue	double [out]	Double value pointer
nValue	int32 [out]	Integral value pointer
strValue	String [out]	String value pointer

Result

None

Remarks

The **SpecialOperationController** method performs a special operation on a stage controller. Special operations are controller type dependent and are documented separately and customer specific.

See also

Method SpecialOperationView, SpecialOperationAxis

Version info

Software v3.5.0.0 or later

7.16.2.27 Stage::SpecialOperationView

This method performs a special operation on the stage view.

Syntax

objStage.**SpecialOperationView(nId, fValue, nValue, strValue)**

Argument

Parameter	Type	Description
nId	int32	Special operation id
fValue	double [out]	Double value pointer
nValue	int32 [out]	Integral value pointer
strValue	String [out]	String value pointer

Result

None

Remarks

The **SpecialOperationView** method performs a special operation on the stage view. Special operations are stage view type dependent and are documented separately and customer specific.

See also

Method SpecialOperationController, SpecialOperationAxis

Version info

Software v3.5.0.0 or later

7.16.2.28 Stage::Stop

This method stops all stage movement.

Syntax

objStage.**Stop()**

Argument

None

Result

None

Remarks

The **Stop** method stops all stage axis in their movement. This can be a reference search, a manual move or a "move to" operation. If the stage is idle this is noop.

See also

Method EmergencyStop, CommitMoveTransaction

Version info

Software v3.5.0.0 or later

7.16.2.29 Stage::Unlock

This method unlocks the stage when locked.

Syntax

objStage.**Unlock()**

Argument

None

Result

None

Remarks

The **Unlock** method unlocks the stage system when locked. No other action is possible on the stage system while it is locked.

See also

Method Lock

Version info

Software v3.8.8.3 or later

7.17 System

The System class is providing general online SPM specific properties and methods.

Table of properties of System class:

Property name	Purpose
SystemState	Defines the state the SPM Controller is in
SystemStateIdleZAxisMode	Defines the mode for the Z-Axis in the Idle-State
SystemStateIdleXYAxisMode	Defines the mode for the XY-Axis in the Idle-State
SystemStateIdleDAC1Mode	Defines the mode for the DAC1 channel in the Idle-State
SystemStateIdleZAxisValue	Defines the position for the Z-Axis in the Idle-State
MeasurementEnvironment	Defines the measurement environment the SPM is working in
SystemHealthState	Monitors the health state of the SPM software / hardware system

Table of methods of System class:

Methode name	Purpose
MotorMove	Performs a motor move
MotorStep	Performs a motor step
MotorStop	Stops any motor movement
ForceMotorPosUpdate	Requests an update of the motor positions
MotorSetPosZero	Sets current position of given motor to 0.0
LevelScanhead	Levels the scanhead
MotorReference	Reference given motor
MotorReferenceAndMoveBack	References given motor and goes back to the previous position
IsMotorReferenced	Checks whether motors are referenced
GetMotorPosition	Returns position of given motor

7.17.1 Properties

7.17.1.1 System::MeasurementEnvironment

Returns or set the sensor measurement environment mode.

Syntax

system.**MeasurementEnvironment** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the measurement environment mode of the sensor system. See valid mode index in the table below.

Remarks

Table of measurement environment mode values and description:

State No.	Name	Description
0	MeasEnv_Air	measure in air
1	MeasEnv_LIquid	measure in liquid

Example

```
' measure in liquid
objSysteme.MeasurementEnvironement = 1
```

See also

Property Cantilever.

7.17.1.2 System::SystemHealthState

Monitors the health state of the SPM software / hardware system

Syntax

system.**SystemHealthState**

Result

The system health state is a value encoded with information about various system states.
Those states are looked at in regard of healthiness. For instance, if the controller isn’t reachable this is unhealthy.
The system health state is a summary of such states and checks and returnes its status as a bit field of results.
A health state of 0 means everything should be ok.

Table of possible health flags:

Bit No.	Name	Description
0	HealthState_CtrlDoNotResponse	The controller gives no answer to a test communication package in a timeframe of of 5sec.
1	SysState_CtrlInSimulationMode	The controller is only simulated. This could be by desire or because it was not found during PC software startup.

Remarks

See also

none

7.17.1.3 System::SystemState

Defines the state the SPM Controller is in

Syntax

system.**SystemState**

Result

The SPM Controller is always in a so called system state. The following SystemStates are available:

Table of operating mode values and description:

State No.	Name	Description

0	SysState_Uncal	State during startup or error
1	SysState_Idle	State after startup and with no running activity
2	SysState_Approach	State during approaching
3	SysState_Scan	State during imagine
4	SysState_Spec	State during spectroscopy
5	SysState_Litho	State during lithography
6	SysState_MacroCmd	State of macro command engine usage

Remarks

The SPM Controller automatically enters states if a activity is started by the user or COM-API (e.g Start Imaging with "Start" button or calling the objScan.Start command). After a activity is finished the SPM Controller enters the **SysState_Idle**.

See also

Properties SystStateIdleZMode, SystemStateIdleXYMode

7.17.1.4 System::SystemStateIdleZAxisMode

Defines the mode for the Z-Axis in the Idle-State

Syntax

system.**SystemStateIdleZAxisMode**

Result

If the SPM Controller is in the **SysState_Idle** this property defines the mode the Z-Axis of the scan head is in.

The following ZIdleModes are available:

State No.	Name	Description
0	SysStateIdleZ_ZControllerActive	Allows the z feedback controller work on this axis
1	SysStateIdleZ_RetractTip	Retract the z-Axis to minimal position
2	SysStateIdleZ_KeepLastPos	Keep the z-Axis value
3	SysStateIdleZ_AbsolutPos	Set the Z-Axis to the defined absolute

		position

Remarks

None.

See also

Properties SystStateIdleZAxisValue

7.17.1.5 System::SystemStateIdleZAxisValue

Defines the position for the Z-Axis in the Idle-State

Syntax

system.**SystemStateIdleZAxisValue**

Result

If the SPM Controller is in the **SysState_Idle** and the ZIdleMode is set to **SysStateIdleZ_AbsolutPos** this property defines the absolute position the z-axis is set to.

Remarks

None.

See also

Properties System.SystemStateIdleZAxisMode

7.17.1.6 System::SystemStateIdleDAC1Mode

Defines the mode for the DAC1 signal channel in the Idle-State

Syntax

system.**SystemStateIdleDAC1Mode**

Result

If the SPM Controller is in the **SysState_Idle** this property defines the mode the DAc1 signal channel is in.

The following ZIdleModes are available:

State No.	Name	Description
0	SysStateIdleZ_ZControllerActive	Allows the z feedback controller work on this axis
1	SysStateIdleZ_RetractTip	Sets the DAC1 output to minimal value
2	SysStateIdleZ_KeepLastPos	Keep the DAC1 value
3	SysStateIdleZ_AbsolutPos	Set the DAC1 to a defined absolute position

Remarks

The DAC1 is mapped by the C3000 controller to the "User Output C" output channel. The User Output C is some times used for controlling a external long range z-actuator.

The Absolute position value of this channel is defined by the objSignalIO.UserDAC1 value.

See also

Properties objSignalIO.UserDAC1

7.17.1.7 System::SystemStateIdleXYAxisMode

Defines the mode for the XY-Axis in the Idle-State

Syntax

system.**SystemStateIdleXYAxisMode**

Result

If the SPM Controller is in the **SysState_Idle** this property defines the mode the Z-Axis of the scan head is in.

The following XYIdleModes are available:

State No.	Name	Description
0	SysStateIdleXY_ImageCenter	Go to the center position defined by the XYOffset of the scan image
1	SysStateIdleXY_KeepLastPosition	Keep the XY-Axis value

Remarks

The XYOffset is defined by Scan.CenterPosX/Y properties. If the SysStateIdleXY_ImageCenter mode is active any change of the CenterPosition moves the tip also during SysState_Idle.

See also

Properties Scan.CenterPosX, Scan.CenterPosY

7.17.2 Methods

7.17.2.1 System::MotorMove

Performs a motor move

Syntax

system.M**otorMove(***nMotor, direction, speed***)**

Argument

Parameter	Type	Description
nMotor	long	Motor ID
nDirection	long	Direction
nSpeed	long	Level

Remarks

This function requires corresponding motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach         = 0,
MotorA                = 1,
MotorB                = 2,
MotorC                = 3,
MotorFocus            = 4,
```

```
MotorPTEX              = 5,
MotorPTEY              = 6,
MotorBeamDeflectionX   = 7,
MotorBeamDeflectionY   = 8,
MotorPhotodiodeLateral = 9,
MotorPhotodiodeNormal  = 10,
MotorLensGimbal        = 11
```

Available nDirections are:

```
Positive = 0,
Negative = 1
```

Available nSpeed levels are:

```
VerySlow = 0,
Slow     = 1,
Normal   = 2,
Fast     = 3,
VeryFast = 4
```

See also

Method MotorStop

7.17.2.2 System::MotorStep

Performs a motor step

Syntax

system.**MotorStep(***nMotor*, *stepSize***)**

Argument

Parameter	Type	Description
nMotor	long	Motor ID
stepSize	double	Step size for motor to perform

Remarks

This function requires corresponding approach motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach          = 0,
MotorA                 = 1,
MotorB                 = 2,
MotorC                 = 3,
```

```
MotorFocus              = 4,
MotorPTEX               = 5,
MotorPTEY               = 6,
MotorBeamDeflectionX    = 7,
MotorBeamDeflectionY    = 8,
MotorPhotodiodeLateral  = 9,
MotorPhotodiodeNormal   = 10,
MotorLensGimbal         = 11
```

See also

Method MotorStop

7.17.2.3 System::MotorStop

Stops motors movement

Syntax

system.**MotorStop()**

Remarks

This function requires corresponding motorization to work correctly.

See also

Method MotorStep, MotorMove

7.17.2.4 System::ForceMotorPosUpdate

Requests an update of the motor positions

Syntax

system.**ForceMotorPosUpdate()**

Remarks

This function requires corresponding motorization to work correctly.

See also

Method GetMotorPosition

7.17.2.5 System::MotorSetPosZero

Sets current position of given motor to 0.0

Syntax

system.**MotorSetPosZero(***nMotor***)**

Argument

Parameter	Type	Description
nMotor	long	Motor ID

Remarks

This function requires corresponding approach motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach          = 0,
MotorA                 = 1,
MotorB                 = 2,
MotorC                 = 3,
MotorFocus             = 4,
MotorPTEX              = 5,
MotorPTEY              = 6,
MotorBeamDeflectionX   = 7,
MotorBeamDeflectionY   = 8,
MotorPhotodiodeLateral = 9,
MotorPhotodiodeNormal  = 10,
MotorLensGimbal        = 11
```

See also

Method GetMotorPosition

7.17.2.6 System::LevelScanhead

Levels the scanhead

Syntax

system.**LevelScanhead()**

Remarks

This function requires corresponding motorization to work correctly.

7.17.2.7 System::MotorReference

References motors

Syntax

system.**MotorReference()**

Argument

Parameter	Type	Description
nMotor	long	Motor ID

Remarks

This function requires corresponding motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach          = 0,
MotorA                 = 1,
MotorB                 = 2,
MotorC                 = 3,
MotorFocus             = 4,
MotorPTEX              = 5,
MotorPTEY              = 6,
MotorBeamDeflectionX   = 7,
MotorBeamDeflectionY   = 8,
MotorPhotodiodeLateral = 9,
MotorPhotodiodeNormal  = 10,
MotorLensGimbal        = 11
```

See also

Method MotorReferenceAndMoveBack, IsMotorReferenced

7.17.2.8 Systen::MotorReferenceAndMoveBack

References motors and goes back to the previous position

Syntax

system.**MotorReferenceAndMoveBack()**

Argument

Parameter	Type	Description

nMotor	long	Motor ID

Remarks

This function requires corresponding approach motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach          = 0,
MotorA                 = 1,
MotorB                 = 2,
MotorC                 = 3,
MotorFocus             = 4,
MotorPTEX              = 5,
MotorPTEY              = 6,
MotorBeamDeflectionX   = 7,
MotorBeamDeflectionY   = 8,
MotorPhotodiodeLateral = 9,
MotorPhotodiodeNormal  = 10,
MotorLensGimbal        = 11
```

See also

Method MotorReference, IsMotorReferenced

7.17.2.9 System::IsMotorReferenced

Checks whether motor is referenced

Syntax

flag = *system*.**IsMotorReferenced(nMotor)**

Argument

Parameter	Type	Description
nMotor	long	Motor ID

Result

Result	Type	Description
flag	Boolean	Returns `True` if motors are referenced

Remarks

This function requires corresponding motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach           = 0,
MotorA                  = 1,
MotorB                  = 2,
MotorC                  = 3,
MotorFocus              = 4,
MotorPTEX               = 5,
MotorPTEY               = 6,
MotorBeamDeflectionX    = 7,
MotorBeamDeflectionY    = 8,
MotorPhotodiodeLateral  = 9,
MotorPhotodiodeNormal   = 10,
MotorLensGimbal         = 11
```

See also

Method MotorReference, MotorReferenceAndMoveBack

7.17.2.10 System::GetMotorPosition

Returns position of given motor

Syntax

position = *system*.**GetMotorPosition(***nMotor***)**

Argument

Parameter	Type	Description
nMotor	long	Motor ID

Result

Result	Type	Description
position	double	Position of a given motor

Remarks

This function requires corresponding approach motorization to work correctly.

Available nMotor ID's are:

```
MotorApproach           = 0,
MotorA                  = 1,
MotorB                  = 2,
MotorC                  = 3,
MotorFocus              = 4,
```

```
    MotorPTEX            = 5,
    MotorPTEY            = 6,
    MotorBeamDeflectionX  = 7,
    MotorBeamDeflectionY  = 8,
    MotorPhotodiodeLateral = 9,
    MotorPhotodiodeNormal  = 10,
    MotorLensGimbal       = 11
```

See also

Method ForceMotorPosUpdate

7.18 Video

The Video class handles the microscope's video camera.

The Video Cameras in the scan head can be controlled by this class. Two cameras are available. A TopView camera to look vertical to the sample and the cantilever and a SideView camera to look about horizontal to the cantilever. **VideoSource** select one of them to be displayed in the "Position Window". For each camera the **Illumination**, the **Brightness** and the **Contrast** of the video display can be adjusted.

A snap shot of the current video image if a compact video camera device is used creates **SaveFrame**. If a flex or a highres video camera device is used use **SaveFrameMPX1** (side view) or **SaveFrameMPX2** (top view).

A object pointer to this class is provided by the Application.Video object property.

Table of properties for Video class:

Property name	**Purpose**
VideoSource	Select either TopView or SideView camera
Illumination	Set the power of sample illumination
Brightness	Set the brightness of video image
Contrast	Set the contrast of video image

Table of methods for Video class:

Method name	**Purpose**
CopyFrame	Copy the video frame to the clipboard
CopyFrameMPX1	Copy the video frame (side view) to the clipboard
CopyFrameMPX2	Copy the video frame (top view) to the clipboard
SaveFrame	Save the video frame as JPEG Image file

SaveFrameMPX1	Save the video frame (side view) as PNG, JPG, BMP image file.
SaveFrameMPX2	Save the video frame (top view) as PNG, JPG, BMP image file.
Start	Start video system, hardware detection, open video panel
Shutdown	Stop video system, release hardware, closes video panel
IsStarted	Check if video system is running

7.18.1 Properties

7.18.1.1 Video::Brightness

Returns or set the video image brightness.

Syntax

video.**Brightness** [= value]

Setting

Argument	Type	Description
value	double	Defines the video image brightness [%]. Values of 0 to 100% are valid.

Remarks

This property defines the brightness of the video image.

Attention: For each video camera the properties Illumination, Brightness and Contrast saves their independent values. Therefore select first the right video source and then set one of the properties.

Example

```
' show side view camera
objOpMode.VideoSource = 0
objOpMode.Brightness  = 100 '%
```

See also

Property VideoSource, Illumination, Contrast.

7.18.1.2 Video::Contrast

Returns or set the video image contrast.

Syntax

video.**Contrast** [= value]

Setting

Argument	Type	Description
value	double	Defines the video image contrast [%]. Values of 0 to 100% are valid.

Remarks

This property defines the contrast of the video image.

Attention: For each video camera the properties Illumination, Brightness and Contrast saves their independent values. Therefore select first the right video source and then set one of the properties.

Example

```
' show top view camera
objOpMode.VideoSource = 1
objOpMode.Contrast    = 80 '%
```

See also

Property VideoSource, Illumination, Brightness.

7.18.1.3 Video::Illumination

Returns or set the sample illumination.

Syntax

video.**Illumination** [= value]

Setting

Argument	Type	Description
value	double	Defines the illumination of the sample in [%]. Values of 0 to 100% are valid.

Remarks

This property defines the sample illumination with the build in light sources of the AFM scan head.

Attention: For each video camera the properties Illumination, Brightness and Contrast saves their independent values. Therefore select first the right video source and then set one of the properties.

Example

```
' show side view camera
objOpMode.VideoSource = 0
objOpMode.Illumination = 60 '%
```

See also

Property VideoSource, Brightness, Contrast.

7.18.1.4 Video::VideoSource

Returns or set the active video camera.

Syntax

video.**VideoSource** [= camera]

Setting

Argument	Type	Description
camera	long	Selects the active video camera. See valid camera index in the table below.

Remarks

The AFM scan head is equipped with two video cameras. This property activates one of them and display its video in the "Position Window".

Attention: For each video camera the properties Illumination, Brightness and Contrast saves their independent values. Therefore select first the right video source and then set one of the properties.

Table of operating mode values and description:

State No.	Name	Description

0	Video_SideView	Activates the horizontal video camera
1	Video_TopView	Activates the vertical video camera

Example

```
' show side view camera
objOpMode.VideoSource = 0
```

See also

Property Illumination, Brightness, Contrast.

7.18.2 Methods

7.18.2.1 Video::Start

Start the video system.
This method starts the hardware detection. opens the VideoPanels (if a camera is connected).

Syntax

video.**Start**

Arguments

none

Result

none

Remarks

If the video system is already started, this has no impact.

Example

```
' start the video system
objVideo.Start()
```

Version info

Software v3.8.2.0 or later

See also

Methode Video::Shutdown
Methode Video::IsStarted

7.18.2.2 Video::Shutdown

Shutdown the video system.
This method shuts down the video hardware detection, releases all hardware resources and closes the VideoPanel;

Syntax

video.**Shutdown**

Arguments

none

Result

none

Remarks

If the video system is already shut off, this method has no impact.

Example

```
' shutdown the video system
objVideo.Shutdown()
```

Version info

Software v3.8.2.0 or later

See also

Methode Video::Start
Methode Video::IsStarted

7.18.2.3 Video::IsStarted

Test if the video system is started

Syntax

video.**IsStarted**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if the video system is started. Returns `False` if it is shut down.

Remarks

none

Example

```
' Test if the video system is started
If objVideo.IsStarted() == False Then
  MsgBox "Video system offline!"
End If
```

Version info

Software v3.8.2.0 or later

See also

Methode Video::Shutdown
Methode Video::IsStarted

7.18.2.4 Video::CopyFrame

Copy the video frame to the clipboard.

Syntax

ok = *video*.**CopyFrame**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if the frame could be copied otherwise `False`.

Remarks

none

Example

```
' save snap shot of top view camera
objVideo.VideoSource = 1
If objVideo.CopyFrame == False Then
  MsgBox "Could not copy video frame!"
End If
```

See also

Property VideoSource

7.18.2.5 Video::CopyFrameMPX1

Copy the video frame to the clipboard.

Syntax

ok = *video*.**CopyFrameMPX1**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if the image could be copied otherwise `False`.

Remarks

none

Example

```
' save snap shot of top view camera
If objVideo.CopyFrameMPX1() == False Then
  MsgBox "Could not save video image!"
End If
```

See also

7.18.2.6 Video::CopyFrameMPX2

Copy the video frame to the clipboard.

Syntax

ok = *video*.**CopyFrameMPX2**

Arguments

none

Result

Result	Type	Description
ok	Boolean	Returns `True` if the image could be copied otherwise `False`.

Remarks

none

Example

```
' save snap shot of top view camera
If objVideo.CopyFrameMPX2() == False Then
  MsgBox "Could not save video image!"
End If
```

See also

7.18.2.7 Video::SaveFrame

Save the video frame into a file.

Syntax

ok = *video*.**SaveFrame(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the video frame. File extension should be .JPG

Result

Result	Type	Description
ok	Boolean	Returns `True` if the frame could be saved otherwise `False`.

Remarks

This method saves a snap shot of the current video display to a file. The file is a JPEG compressed video frame.

Example

```
' save snap shot of top view camera
objVideo.VideoSource = 1
If objVideo.SaveFrame("topimage.jpg") == False Then
  MsgBox "Could not save video image!"
End If
```

See also

Property VideoSource

7.18.2.8 Video::SaveFrameMPX1

Save the video image into a file.

Syntax

ok = *video*.**SaveFrameMPX1(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the video image. File extension should be .JPG

Result

Result	Type	Description
ok	Boolean	Returns `True` if the image could be saved otherwise `False`.

Remarks

This method saves a snap shot of th current video display to a file. The file is a JPEG compressed video image.

Example

```
' save snap shot of side view camera
If objVideo.SaveFrameMPX1("sideimage.jpg") == False Then
  MsgBox "Could not save video image!"
End If
```

See also

7.18.2.9 Video::SaveFrameMPX2

Save the video image into a file.

Syntax

ok = *video*.**SaveFrameMPX2(**filename**)**

Arguments

Argument	Type	Description
filename	String	Path and filename of the video image. File extension should be .JPG

Result

Result	Type	Description
ok	Boolean	Returns `True` if the image could be saved otherwise `False`.

Remarks

This method saves a snap shot of th current video display to a file. The file is a JPEG compressed video image.

Example

```
' save snap shot of top view camera
```

```
If objVideo.SaveFrameMPX2("topimage.jpg") == False Then
  MsgBox "Could not save video image!"
End If
```

See also

7.19 Thermal Tune

The Thermal Tune class handles the microscope thermal tune procedure.

A object pointer to this class is provided by the Scanhead.ThermalTuning object property.

Table of properties for Thermal Tune class:

Property name	Purpose
FreqBandUpperBound	Get or set upper bound of frequency band to be analyzed [Hz]
FreqResolution	Get or set frequency resolution of FFT [Hz]
BlockCount	Get or set how many blocks are sampled (0 = continuous)
AverageType	Get or set the averaging function to use
CantileverTemperature	Get or set temperature around cantilever
FreqLowerBound	Get or set frequency lower bound (used for fitting)
FreqUpperBound	Get or set frequency upper bound (used for fitting)

Table of methods for Thermal Tune class:

Method name	Purpose
Start	Start Thermal Tune data capture and calculation procedure
Stop	Stop Thermal Tune data capture and calculation procedure
AutoSetupFrequencies	Calculates suitable lower and upper bound values for the peak search algorithm based on cantilever characteristics
GetCurrentBlockCount	Returns current block count of data acquisition
GetFrequencyList	Returns a list of frequencies taking frequency band and resolution into account
GetBlock	Returns a vector of captured data
GetCurrentAverage	Returns a buffer with the average of all measured blocks
NsfCustomFit	Fits a curve to match pwrSpectrum based on Nanosurf's

	own method
NsfCustomFitOnCurrentAverageAndBounds	Fits a curve to match the current average spectrum and bounds based on Nanosurf's own method
NsfCustomFitOnCurrentAverage	Fits a curve to match the current average spectrum based on Nanosurf's own method
NsfCustomFitCurve	Calculates y-values for a list of x-values based on Nanosurf's own curve fit algorithm
SimpleHarmonicOscFit	Fits a curve to match pwrSpectrum based on simple harmonic oscillator method
SimpleHarmonicOscFitOnCurrentAverageAndBounds	Fits a curve to match the current average spectrum and bounds based on simple harmonic oscillator method
SimpleHarmonicOscFitOnCurrentAverage	Fits a curve to match the current average spectrum based on simple harmonic oscillator method
SimpleHarmonicOscFitCurve	Calculates y-values for a list of x-values based on simple harmonic oscillator method
CalculateSpringConstant_Sader	Calculates the spring constant from the current noise measurement average with the Sader methode
CalculateSpringConstant_Equipartition	Calculates the spring constant from the current noise measurement average with the equipartition method

7.19.1 Properties

7.19.1.1 ThermalTuning::FreqBandUpperBound

Get or set upper bound of frequency band to be analyzed in Hz.

Syntax

ThermalTuning.**FreqBandUpperBound** [= value]

Setting

Argument	Type	Description
value	double	Upper bound taken into account for FFT

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.2 ThermalTuning::FreqResolution

Get or set frequency resolution of FFT in Hz.

Syntax

ThermalTuning.**FreqResolution** [= value]

Setting

Argument	Type	Description
value	double	Requested frequency resolution of FFT

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.3 ThermalTuning::BlockCount

Get or set how many blocks are sampled (0 = continuous).

Syntax

ThermalTuning.**BlockCount** [= value]

Setting

Argument	Type	Description
value	long	Defines how many blocks are sampled (0 = continuous)

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.4 ThermalTuning::AverageType

Get or set how many blocks are sampled (0 = continuous).

Syntax

ThermalTuning.**AverageType** [= value]

Setting

Argument	Type	Description
value	long	Defines the averaging function to use

Remarks

Value	Name	Description
0	ExponentialDecay	Exponential decay averaging
1	ProportionalWeight	Arithmetic mean calculation

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.5 ThermalTuning::CantileverTemperature

Get or set temperature of cantilever environment.

Syntax

ThermalTuning.**CantileverTemperature** [= value]

Setting

Argument	Type	Description
value	double	Environment temperature in degree Celsius

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.6 ThermalTuning::FreqLowerBound

Get or set frequency lower bound (used for fitting)

Syntax

ThermalTuning.**FreqLowerBound** [= value]

Setting

Argument	Type	Description
value	double	Frequency lower bound used for fitting

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.1.7 ThermalTuning::FreqUpperBound

Get or set frequency upper bound (used for fitting)

Syntax

ThermalTuning.**FreqUpperBound** [= value]

Setting

Argument	Type	Description
value	double	Frequency upper bound used for fitting

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.2 Methods

7.19.2.1 ThermalTuning::Start

Starts continuous data capture of thermal tune data.

Syntax

ThermalTuning.**Start**

Remarks

Thermal refers to the thermally activated spontaneous motion of the cantilever that can be seen in such measurements, tune refers to the procedure of determining the resonance characteristics of the cantilever from this data.

Example

```
' create objects
Dim objApp  : Set objApp  = SPM.Application
Dim objScanhead : Set objScanhead = objApp.Scanhead
Dim objThermalTune : Set objThermalTune = objScanhead.ThermalTuning

' prepare
objThermalTune.FreqBandUpperBound 188000
dim currentAverage
```

```
' start scan
objThermalTune.Start

' do something

' finish immediately
objThermalTune.Stop
```

See also

Method ThermalTuning.Stop

7.19.2.2 ThermalTuning::Stop

Stop continuous data capture of thermal tune data.

Syntax

ThermalTuning.**Stop**

Remarks

Thermal refers to the thermally activated spontaneous motion of the cantilever
that can be seen in such measurements, tune refers to the procedure of determining the
resonance characteristics of the cantilever from this data.

Example

```
' prepare
objThermalTune.FreqBandUpperBound 188000
dim currentAverage

' start scan
objThermalTune.Start

' get data
currentAverage = objThermalTune.GetCurrentAverage()
' do something

' finish immediately
objThermalTune.Stop
```

See also

Method ThermalTuning.Start

7.19.2.3 ThermalTuning::AutoSetupFrequencies

Automatically calculates suitable lower and upper bound values for the peak search algorithm.
Based on a margin of the resonance frequency supplied by the manufacturer and registered in the cantilever list.

Syntax

ThermalTuning.**AutoSetupFrequencies(***bOnlyCalculateIfBadValues***)**

Argument

Parameter	Type	Description
bOnlyCalculateIfBadValues	bool	only if the current resonance frequency is out of the lower/upper bound the calculation should be run

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Properties ThermalTuning.FreqBandUpperBound

7.19.2.4 ThermalTuning::GetCurrentBlockCount

Returns the current block count.

Syntax

value = *objThermalTune*.GetCurrentBlockCount**()**

Result

Result	Type	Description
value	long	Current index of block

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Property ThermalTuning.BlockCount

Method ThermalTuning.GetBlock

7.19.2.5 ThermalTuning::GetFrequencyList

Returns a buffer with the frequencies associated with the FFT bins

Syntax

value = *objThermalTune*.GetFrequencyList(*bool*)

Argument

Parameter	Type	Description
bOnlyCalculateIfBad Values	bool	Calculate frequency list in place instead of returning internal list which may not have been calculated yet (is calculated when sampling is started).

Result

Result	Type	Description
variant_array	double	Buffer with frequencies

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.2.6 ThermalTuning::GetBlock

Returns a buffer with the frequencies associated with the FFT bins

Syntax

value = *objThermalTune*.GetFrequencyList(*bool*)

Argument

Parameter	Type	Description
bOnlyCalculateIfBad Values	bool	Calculate frequency list in place instead of returning internal list which may not have been calculated yet (is calculated when sampling is started).

Result

Result	Type	Description
variant_array	double	Buffer with frequencies

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.2.7 ThermalTuning::GetCurrentAverage

Returns a buffer with the average of all measured blocks.

Syntax

value = *objThermalTune*.GetCurrentAverage()

Result

Result	Type	Description
variant_array	double	Current average over all measured blocks

Remarks

None

Example

See Scanhead.ThermalTuning

See also

None

7.19.2.8 ThermalTuning::NsfCustomFit

Fits a curve to the power spectral density based on Nanosurf's own method.

Syntax

value = *objThermalTune*.NsfCustomFit(*frequencyList, pwrSpectrum, nLowerBound, nUpperBound*)

Argument

Parameter	Type	Description
frequencyList	Variant	Frequency list for the FFT power spectrum to be fitted [Hz]
pwrSpectrum	Variant	Power spectrum values (shall be same size as frequencyList)
nLowerBound	double	Lower bound of frequeny range to be used to find the resonance peak [Hz]
nUpperBound	double	Upper bound of frequeny range to be used to find the resonance peak [Hz]

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with custom fit function)

```
enum NSFFitParams
{
  NSF_Damping = 0,
  NSF_Sigma = 1,
  NSF_ResFreq = 2,
  NSF_QualityFactor = 3,
  NSF_ResPkAmplitudeAboveNoise = 4,
  NSF_NumOfParams = 5
};
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also
Method ThermalTuning.NSFCustomFitOnCurrentAverageAndBounds
Method ThermalTuning.NSFCustomFitOnCurrentAverage

7.19.2.9 ThermalTuning::NsfCustomFitOnCurrentAverageAndBounds

Fits a curve to the power spectral density based on Nanosurf's own method.

Syntax

value = *objThermalTune*.NsfCustomFitOnCurrentAverageAndBounds()

Argument

Parameter	Type	Description
nLowerBound	double	Lower bound of frequeny range to be used to find the resonance peak [Hz]
nUpperBound	double	Upper bound of frequeny range to be used to find the resonance peak [Hz]

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with custom fit function)

```
enum NSFFitParams
{
  NSF_Damping = 0,
  NSF_Sigma = 1,
  NSF_ResFreq = 2,
  NSF_QualityFactor = 3,
  NSF_ResPkAmplitudeAboveNoise = 4,
  NSF_NumOfParams = 5
};
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.NSFCustomFitOnCurrentAverage
Method ThermalTuning.NSFCustomFit

7.19.2.10 ThermalTuning:NsfCustomFitOnCurrentAverage

Fits a curve to the power spectral density based on Nanosurf's own method.

Syntax

value = *objThermalTune*.NsfCustomFitOnCurrentAverage(*nLowerBound, nUpperBound*)

Argument

Parameter	Type	Description
nLowerBound	double	Lower bound of frequeny range to be used to find the resonance peak [Hz]
nUpperBound	double	Upper bound of frequeny range to be used to find the resonance peak [Hz]

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with custom fit function)

```
enum NSFFitParams
{
  NSF_Damping = 0,
  NSF_Sigma = 1,
  NSF_ResFreq = 2,
  NSF_QualityFactor = 3,
  NSF_ResPkAmplitudeAboveNoise = 4,
  NSF_NumOfParams = 5
};
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.NSFCustomFitOnCurrentAverageAndBounds
Method ThermalTuning.NSFCustomFit

7.19.2.11 ThermalTuning:NsfCustomFitCurve

Calculates y-values for a list of x-values based on Nanosurf's own curve fit algorithm.

Syntax

value = *objThermalTune*.NsfCustomFitCurve(*frequencyList, fitParams*)

Argument

Parameter	Type	Description
frequencyList	Variant	List of frequencies (X-axis positions) where Y-axis values shall be computed]
fitParams	Variant	fitParams Set of curve fitting parameters (compatible with custom fit function)

```
enum NSFFitParams
{
  NSF_Damping = 0,
  NSF_Sigma = 1,
  NSF_ResFreq = 2,
  NSF_QualityFactor = 3,
  NSF_ResPkAmplitudeAboveNoise = 4,
  NSF_NumOfParams = 5
};
```

Result

Result	Type	Description
variant_array	Variant	Y-axis values on fitted curve corresponding to positions on X-axis

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.NSFCustomFitOnCurrentAverageAndBounds
Method ThermalTuning.NSFCustomFitOnCurrentAverage

7.19.2.12 ThermalTuning:SimpleHarmonicOscFit

Fits a curve to the power spectral density based on simple harmonic oscillator model.

Syntax

value = *objThermalTune*.SimpleHarmonicOscFit(*frequencyList, pwrSpectrum, nLowerBound, nUpperBound*)

Argument

Parameter	Type	Description
frequencyList	Variant	Frequency list for the FFT power spectrum to be fitted [Hz]
pwrSpectrum	Variant	Power spectrum values (shall be same size as frequencyList)
nLowerBound	double	Lower bound of frequeny range to be used to find the resonance peak [Hz]
nUpperBound	double	Upper bound of frequeny range to be used to find the resonance peak [Hz]

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with simple harmonic fit function)

```
enum SHOFitParams
  {
    SHO_WhiteNoise = 0,
    SHO_PinkNoise = 1,
    SHO_ResFreq = 2,
    SHO_QualityFactor = 3,
    SHO_ResPkAmplitudeAboveNoise = 4,
    SHO_NumOfParams = 5
  };
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverageAndBounds
Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverage

7.19.2.13 ThermalTuning:SimpleHarmonicOscFitOnCurrentAverageAndBounds

Fits a curve to the power spectral density based on simple harmonic oscillator model.

Syntax

value = *objThermalTune*.SimpleHarmonicOscFitOnCurrentAverageAndBounds()

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with simple harmonic fit function)

```
enum SHOFitParams
  {
    SHO_WhiteNoise = 0,
    SHO_PinkNoise = 1,
    SHO_ResFreq = 2,
    SHO_QualityFactor = 3,
    SHO_ResPkAmplitudeAboveNoise = 4,
    SHO_NumOfParams = 5
  };
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.SimpleHarmonicOscFit
Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverage

7.19.2.14 ThermalTuning:SimpleHarmonicOscFitOnCurrentAverage

Fits a curve to the power spectral density based on simple harmonic oscillator model.

Syntax

value = *objThermalTune*.SimpleHarmonicOscFitOnCurrentAverage(*nLowerBound, nUpperBound*)

Argument

Parameter	Type	Description
nLowerBound	double	Lower bound of frequeny range to be used to find the resonance peak [Hz]
nUpperBound	double	Upper bound of frequeny range to be used to find the resonance peak [Hz]

Result

Result	Type	Description
variant_array	Variant	Set of curve fitting parameters (compatible with simple harmonic fit function)

```
enum SHOFitParams
  {
    SHO_WhiteNoise = 0,
    SHO_PinkNoise = 1,
    SHO_ResFreq = 2,
    SHO_QualityFactor = 3,
    SHO_ResPkAmplitudeAboveNoise = 4,
    SHO_NumOfParams = 5
  };
```

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.SimpleHarmonicOscFit
Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverageAndBounds

7.19.2.15 ThermalTuning:SimpleHarmonicOscFitCurve

Calculates y-values for a list of x-values based on simple harmonic oscillator curve fit algorithm.

Syntax

value = *objThermalTune*.SimpleHarmonicOscFitCurve(*frequencyList, fitParams*)

Argument

Parameter	Type	Description
frequencyList	Variant	List of frequencies (X-axis positions) where Y-axis values shall be computed]
fitParams	Variant	fitParams Set of curve fitting parameters (compatible with custom fit function)

```
enum SHOFitParams
{
  SHO_WhiteNoise = 0,
  SHO_PinkNoise = 1,
  SHO_ResFreq = 2,
  SHO_QualityFactor = 3,
  SHO_ResPkAmplitudeAboveNoise = 4,
  SHO_NumOfParams = 5
};
```

Result

Result	Type	Description
variant_array	Variant	Y-axis values on fitted curve corresponding to positions on X-axis

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverage
Method ThermalTuning.SimpleHarmonicOscFitOnCurrentAverageAndBounds

7.19.2.16 ThermalTuning:CalculateSpringConstant_Sader

Calculates the spring constant with the Sader method.

Syntax

value = *objThermalTune*.CalculateSpringConstant_Sader(*nCantileverLength, nCantileverWidth, nResFreq, nQualityFactor, nViscosity, nDensity*)

Argument

Parameter	Type	Description
nCantileverLength	double	Cantilever length [m]
nCantileverWidth	double	Cantilever width [m]
nResFreq	double	Cantilever resonance frequency [Hz]
nQualityFactor	double	Cantilever resonance peak quality factor
nViscosity	double	Viscosity of environment [kg/m/s]
nDensity	double	Densitiy of envivornment [kg/m3]

Result

Result	Type	Description
value	double	Spring constant in N/m

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.CalculateSpringConstant_Equipartition

7.19.2.17 ThermalTuning:CalculateSpringConstant_Equipartition

Calculates the spring constant with the equipartition theorem method.

Syntax

value = *objThermalTune*.CalculateSpringConstant_Equipartition(*nAbsTempKelvin, nTipHeight, nCantileverLength, nCantileverAngle, nA, nResFreq, nQualityFactor)*

Argument

Parameter	Type	Description
nAbsTempKelvin	double	Absolute temperature [Kelvin]
nTipHeight	double	Tip height [m]
nCantileverLength	double	Cantilever length [m]
nCantileverAngle	double	Cantilever angle [rad]
nA	double	Amplitude of resonance peak [m/sqrt(Hz)]
nResFreq	double	Cantilever resonance frequency [Hz]
nQualityFactor	double	Cantilever resonance peak quality factor

Result

Result	Type	Description
value	double	Spring constant in N/m

Remarks

None

Example

See Scanhead.ThermalTuning

See also

Method ThermalTuning.CalculateSpringConstant_Sader

7.20 ZController

The ZController class handles the microscope's Z feedback loop controller properties.

The Z-Controller is controlling the z-axis of the scan head and track the surface by keeping the tip sample distance constant. This is done by sensing a input signal and compare it to the value in **SetPoint**. Deviations of the input signal to this set point value is resulting in a z motion. The translation of the error to the z-motion is adjusted by different controller gains. The **PGain**, the **IGain** and the **DGain**. Also different controller algorithm can be chosen to adapt the controller best to the sample by **Algorithm**. The signal used as the input signal to the feedback controller is defined by the selected operating mode of the sensor. Refer to class OperatingMode.

A tip voltage can be applied to the tip to improve the sensors signal by **TipVoltage**.

A object pointer to this class is provided by the Application.ZController object property.

Table of properties of ZController class:

Property name	Purpose
SetPoint	Define the reference value for the input signal
PGain	Proportional amplification of input error
IGain	Amplification of the sum of the input error
DGain	Amplification of the change of the input error
LoopMode	operating mode of the z controller feedback loop
ErrorInputGain	Amplification of the error signal prior the ADC
Algorithm	Defines the algorithm used in the z feedback controller
TipVoltage	Defines the voltage applied to the sensors tip
SetPointForceUnitMode	Defines the unit of the Setpoint for static force operating modes
OutputSel	Defines the output channel of the z-feedback controller
PGain2IGain2DGain2	Defines the PID Gains used for the secondary feedback output

Table of methods of class ZController:

Method name	Purpose
Retract	Retract the tip.
IsRetracting	Returns f the retracting process is active
GetInputValue	Returns the current feedback input value
GetOutputValue	Returns the current feedback output value

7.20.1 Properties

7.20.1.1 ZController::Algorithm

Returns or set used z-feedback loop algorithm.

Syntax

zctrl.**Algorithm** [= algo]

Setting

Argument	Type	Description
algo	long	Defines the z-feedback loop algorithm. See available modes in the table below.

Remarks

Not available with C3000!

For the z-feedback control loop various algorithm can be used. This property selects one.

The algorithm is defining on how the z-feedback is reacting on a input signal error. Different algorithm can be selected to change the behaviour and can adapted to different surfaces properties.

Table of algorithm values and description:

State No.	Name	Description
0	CtrlAlgo_StandardPID	Classic PID-Controller
1	CtrlAlgo_PIAndFilter	PI-Controller with moving averaging filter for input signals

See also

None

7.20.1.2 ZController::DGain

Returns or set the differential gain of the z-feedback controller.

Syntax

zctrl.**DGain** [= gain]

Setting

Argument	Type	Description
gain	double	Defines the amplification of the change speed of the difference between input signal and set point value. Valid values are 0 .. 32767. For C3000 controller [0 ... 2^24]

Remarks

The D-Gain is defining the amplification of difference between the last and the current difference between input signal and the set point value. Differential gain has to use very carefully. A higher amplification generates a faster response but a gain value too high can lead to oscillation of the z feedback loop and the D-Gain amplifies noise from the input signal too.

A value of zero switch of the differential gain completely.

Example

```
objZCtrl.DGain = 5
```

See also

Property PGain, IGain, SetPoint

7.20.1.3 ZController::ErrorInputGain

Returns or set the amplification of the error input signal.

Syntax

zctrl.**ErrorInputGain** [= gain]

Setting

Argument	Type	Description
gain	long	Defines the amplification of the difference between input signal and set point value prior the ADC as the exponent of 2. Valid values are 0 .. 4.

Remarks

Not available with C3000!

The error input amplifier is a analog circuit which can enhance the sensitivity of the z-feedback controller signal input. Event so that the input sensitivity is high for very smooth surfaces and special samples a increasing of the input sensitivity is desired. With this property the amplification can be set up to 16 in steps of power of two. The property value is the exponent for the number two.

Actual error gain is calculated:

amplification = 2 ^ ***ErrorInputGain***

Note that by increasing the amplification the maximal signal range is decreasing proportionally!

Example

```
' set the preamplifier to 8 = 2 ^4
objZCtrl.ErrorInputGain = 4
```

See also

Property SetPoint

7.20.1.4 ZController::SetPointForceUnitMode

Defines used unit for the static force setpoint

Syntax

zctrl.**SetPointForceUnitMode** [= index]

Argument

Argument	Type	Description
index	long	Defines the unit of the deflection signal.

Remarks

For Static Force Mode AFM different signal units for the setpoint could be of interest. How the SetPoint is interpreted is defined by this property.

The following mode indexes are defined:

```
DefUnitMode_V     = 0,
DefUnitMode_m     = 1,
DefUnitMode_N     = 2,
```

See also

objScanHead.DeflectionUnitMode

7.20.1.5 ZController::IGain

Returns or set the integral gain of the z-feedback controller.

Syntax

zctrl.**IGain** [= gain]

Setting

Argument	Type	Description
gain	double	Defines the amplification of the accumulating sum of the difference between input signal and set point value. Valid values are 0 .. 32767. For C3000 controller [0 ... 2^24]

Remarks

The I-Gain is defining the amplification of sum of the difference between input signal and the set point value. A higher amplification generates a faster response to a input signal error and therefore the topography is reproduced better by the z-scanner. But a gain value too high can lead to oscillation of the z feedback loop and amplifies also noise from the input signal.

A value of zero switch of the integral gain completely.

Example

```
objZCtrl.IGain = 2000
```

See also

Property PGain, DGain, SetPoint

7.20.1.6 ZController::LoopMode

Returns or set the z-feedback loop mode.

Syntax

zctrl.**LoopMode** [= mode]

Setting

Argument	Type	Description
mode	long	Defines the z-feedback loop mode. See modes in the table below.

Remarks

Not available with C3000!

The z-feedback control loop can be in various states. This property defines them.

Standard operating for imaging needs Loopmode_Run for operation.
X/Y-Slope compensation and ZPlane offsets are always active and can be used to move the tip without feedback control.

For states other than Loopmode_Run the risk of tip damaging is high.

Table of loop mode values and description:

State No.	Name	Description
0	Loopmode_Run	Standard operating of feedback loop
1	Loopmode_Freeze	Feedback controller is frozen at the last position. No controlling of distance is performed.
2	Loopmode_StopAndClear	Feedback controller is stoped and integrator set to zero. No distance controlling is performed.

See also

Class Scan

7.20.1.8 ZController::PGain

Returns or set the proportional gain of the z-feedback controller.

Syntax

zctrl.**PGain** [= gain]

Setting

Argument	Type	Description
gain	double	Defines the amplification of the difference between input signal and set point value. Valid values are 0 .. 32767. For C3000 controller [0 ... 2^24]

Remarks

The P-Gain is defining the amplification of the input signal error compared to the set point value. A higher amplification generates a faster response to a input signal error and therefore the topography is reproduced better by the z-scanner. But a gain value too high can lead to oscillation of the z feedback loop and amplifies also noise from the input signal.

A value of zero switch of the proportional gain completely.

Example

```
objZCtrl.PGain = 10000
```

See also

Property IGain, DGain, SetPoint

7.20.1.10 ZController::SetPoint

Returns or set the reference value of the z controller.

Syntax

zctrl.**SetPoint** [= value]

Setting

Argument	Type	Description
value	double	Defines the reference value for the sensor signal from the scan head.

Remarks

The set point is the reference value for the z-controller's input signal. The z-controller tries to keep the sensor input signal as close to this reference value as possible by moving the sensor tip in along the z-axis of the scanner.

The unit of these property depends on the operating mode selected by property OperatienMode.OperatingMode.

Op. mode	Input Signal	Unit
STM	Tunneling Current	Ampere
Static AFM	Deflection	Newton
Dynamic AFM	Amplitude	Percentage of resonance peak [0 .. 100%]
Phase Contrast	Amplitude	Percentage of resonance peak [0 .. 100%]
Force Modulation	Deflection	Newton
Spreading Resistance	Deflection	Newton

If the operating mode is changed the set point changes to the last value defined for this mode too.

Example

```
' dynamic force AFM mode: 50% of resonance peak height
objZCtrl.SetPoint = 50 '[%]

' spreading resistance mode: 10nN force
objZCtrl.SetPoint = 10e-9 '[N]
```

See also

Class OperatingMode

7.20.1.11 ZController::TipVoltage

Returns or set the sensors tip potential.

Syntax

zctrl.**TipVoltage** [= potential]

Setting

Argument	Type	Description
potential	double	Defines the potential applied to the tip in voltage. Valid range from -10V to +10V.

Remarks

The potential on the tip can be defined with this property. This could be usefully to compensate electrostatic charges.

Example

```
' set the tip voltage to -3.5V
objZCtrl.TipVoltage = -3.5
```

See also

None.

7.20.2 Methods

7.20.2.4 ZController::Retract

7.20.2.4 ZController::Retract

7.20.2.4 ZController::Retract

7.20.2.4 ZController::Retract

Retract the sensor Tip.

Syntax

zctrl.**Retract(***pos***)**

Argument

Paramete r	Type	Description
pos	short	retract position. -32768 = pull back, +32767 release

8 Version history

List of changes in this document and the object reference

Software v3.10.1

Adaptations in Scanhead, Approach and System

Software v3.10.0

General improvements of interface descriptions for: Stage, Scan (Prescan), Spec, Video,

Corrections in ToC

Software v3.5.0

New Classes Stage and BatchManager

Software v3.4.0

Class Spec update for new Spectroscopy