TEM help manual -- Scripting

Table of Contents

1 TEM Scripting Software License Agreement

TEMScripting Software License *Revised October 2012*

"Software" means the TEMScripting software program, including any updates and parts thereof, whether expressed in object code, source code or otherwise. Software is copyrighted, and FEI retains exclusive right, title and interest in and to the Software and all copies or portions thereof, including all intellectual property rights. Subject to the payment of all fees due hereunder, FEI hereby grants Buyer a nonexclusive, nontransferable license in perpetuity to use the Software on the system on which it is originally installed and generate or develop scripts or programs for internal purposes only (and not for distribution or resell, in either case on a commercial or non-commercial basis), subject to the provisions of this license. As a condition to the license of the Software, Buyer hereby grants to FEI a royalty-free, transferable, perpetual license to such scripts or programs. Buyer understands that certain scripts or programs may adversely impact instrument performance and Buyer takes full responsibility for any script or program it creates using the Software. Seller does not warrant that (i) the Software will meet Buyer's requirements, (ii) the Software will operate in combination with other hardware, software, systems or data not provided by Seller (except as expressly specified in the documentation provided with the Product), (iii) the operation of the Software, scripts, or programs will be uninterrupted or error-free, or (iv) all Software errors will be corrected. THE WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO THE PRODUCT. NO WARRANTIES SHALL ARISE UNDER THIS AGREEMENT FROM COURSE OF DEALING, COURSE OF PERFORMANCE OR USAGE OF TRADE. SELLER EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR ANY PARTICULAR PURPOSE.

The Software and documentation and any copies, translations, compilations, partial copies, modifications, improvements, enhancements and updates are proprietary to FEI or its licensors, and contain copyrighted material, trade secrets and other proprietary material. In order to protect such intellectual property rights and preserve the confidentiality of the Software, Buyer may not decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form, except to the extent expressly permitted by mandatory provisions of applicable law (including national laws implementing Directive 91/250/EEC on the legal protection of computer programs) in order to gain certain information specified therein, provided that Buyer shall not exercise its rights under such laws, unless and until Buyer has first requested the required information from FEI in writing, and FEI, at its sole discretion, has not complied with Buyer's request within a commercially reasonable period of time. Buyer may not modify, network, rent, lease, loan, distribute or create derivative works (other than the scripts or programs) based upon the Software, in whole or in part. Buyer shall not remove any proprietary notices from any part of the Software or documentation. Licensors of third party software that may be included in the Software have all the rights and benefits of FEI under this license, and, to the extent permitted by applicable law, shall have no liability for any damages, whether direct, indirect, incidental or consequential, arising from the use of such third party software. Except as explicitly set forth in the FEI quotation relating to this Software, Buyer shall not make Software or any script or programs available in any form to any third party without the prior written consent of FEI, which consent may be contingent upon the payment of a transfer fee for the Software.

IN NO EVENT SHALL FEI OR ITS SUPPLIERS BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OR LOSSES ALTHOUGH FEI MAY BE INFORMED OF THE POSSIBILITY OF SUCH DAMAGES IN ADVANCE. EXCEPT AS EXPRESSLY PROVIDED HEREIN, FEI DISCLAIMS ALL OTHER LIABILITY TO BUYER OR ANY OTHER PERSON IN CONNECTION WITH THIS LICENSE OR THE DELIVERY OR NON-DELIVERY, SALE, MAINTENANCE, USE OR PERFORMANCE OF PRODUCT, INCLUDING SPECIFICALLY, BUT WITHOUT LIMITATION, LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY IN TORT. NOTWITHSTANDING ANY OTHER PROVISION OF THIS AGREEMENT, IN NO EVENT SHALL FEI'S OR FEI'S SUPPLIERS' LIABILITY UNDER THIS AGREEMENT EXCEED THE PURCHASE PRICE PAID FOR THE PRODUCT BY BUYER. BUYER ACKNOWLEDGES THAT THE PRICING OF THE PRODUCT AND THE OTHER TERMS AND CONDITIONS OF THIS AGREEMENT REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT FEI WOULD NOT ENTER INTO THIS AGREEMENT WITHOUT THESE LIMITATIONS OF ITS LIABILITY.

2 TEM Scripting

IMPORTANT NOTE: 64-bit scripting is not supported, all scripting is 32-bit.
This implies that you have to modify your scripts/programs so they work in 32-bit when connecting to scripting. For VB, run under the 32-bits interpreter.
Note: If you receive back from the scripting adapter that "… the dark field element was not found", you are trying to run as 64-bits.

The TEM scripting adapter enables communication between a script and the TEM microscope (Tecnai or Titan).
To make use of TEM scripting, it is not necessary to have a deeper understanding of the internal structure of the TEM software. The adapter's functionality should be understandable to somebody who is able to use a TEM microscope.
Most scripting or programming languages (e.g. VBScript, JScript, Visual Basic, C++, Delphi, Python, C#) are supported. The examples you find in the various chapters of this help file use JScript syntax, but the help also includes chapters that address the peculiarities of other languages. You can also inspect the example programs that are delivered with TEM scripting (the default installation directory is tecnai\scripting or titan\scripting).

2.1 Introduction

The TEM Scripting adapter supports its clients (in other words: your scripts) with a number of software objects that correspond to the various parts of the microscope. These objects are the main components of the adapter. The adapter further provides some other features such as User Button events and TEM constants for convenience.

2.1.1 Tem objects

2.1.1.1 The 'Instrument' object

This is the main object. It manages access to all parts of the microscope, for example the gun, the camera and so forth, that are modelled as 'secondary microscope objects'.

Please note: Because of the introduction of the Titan Microscope, the TEM scripting object (previously called "Tecnai") has become more generic: it is now called TEMScripting. Because of this change you may have to recompile any existing code. Additionally, 'virtual/pseudo names' for some vacuum gauges on Tecnai are no longer supported; see section The Gauge (utility object) for more information.

But in the main, all functions previously present have not changed.

2.1.1.2 The secondary microscope objects

These objects represent the parts of the microscope. They have properties and methods as is common for objects in most scripting languages. The properties directly correspond to microscope parameters (such as magnification, beam shift etc.). Together they define the state of the microscope that can be controlled through scripting. The methods generally invoke some actions (taking an exposure, normalizing the lenses etc).

2.1.1.3 The utility objects

There are some TEM specific utility objects that are used to handle multidimensional properties (such as the stage position, which consists of 5 coordinates).

2.1.2 TEM Scripting Features:

2.1.2.1 Synchronous functions

In order to support a simple, sequential way of scripting, all the adapters functions work synchronous, i.e. they will only return after they have finished executing. (The setting of properties is equivalent to a function call.)

2.1.2.2 Events

The adapter supports the usage of the six User Buttons L1..L3, R1..R3 (on the TEM control pads) and can, if desired, generate events when they are pressed. These events can be handled by the script. Thus a user can interact with the script via the control pads of the microscope.

2.1.2.3 TEM constants

For ease of programming, a number of TEM-specific constants have been defined. These constants are accessed as elements of enumerations. For example, there exists an enumeration named ScreenPosition that enumerates the possible positions of the fluorescent screen. Using the values spUP, spDown, and spUnknown, that are elements of this enumeration, instead of direct values like 1 ,2 , or 3 makes programming easier and reduces the chance of programming errors. Some programming environments (such as the one for VB, for example) possess an IntelliSense mechanism that supports the use of enumerated constants via dropdown menus.

2.1.2.4 TEM error codes

For the same reason the adapter includes some TEM-specific error codes, that represent physical error conditions.

2.2 Testing your Scripting Programs

Testing your TEM Scripting programs can obviously be done on the Microscope itself. However it is wise to do a reasonable amount of testing on a test PC running Tecnai/Titan in simulation mode before experimenting on the hardware.

Where possible make sure that the simulation software is being installed on a system with ‘no other software’. FEI recommends not installing the simulation software on a PC that is heavily used for other purposes. At the very least make sure that a full backup image is taken of the PC beforehand in case of problems. *Explanation - the simulation software is a subset of what is actually installed on the Microscope PC which is a ‘relatively controlled environment’. In other words, it is not practical to test that the simulation software runs in combination with all other software packages that might be running on the test PC.*

Installing the TEM Scripting Adapter currently means that the regular TEM software is also installed (many of the regular software files are used during the registration of the scripting adapter and without proper registration the software will not run). It can be installed in two ways:

1. Use the TEM software DVD to install the software. In this case the software itself is not switched to simulation and cannot be run (it expects the hardware to be present). So it will not be able to do any testing on the test PC with this setup.

2. Request a Simulation DVD from FEI.

2.3 A first example

It is easy to start with TEM scripting. Suppose you want to know the actual magnification. You need to do three things:

1. create an ‘Instrument’ object
2. get access to the projection system
3. check the property ‘Magnification’

The following code does this (in JScript syntax):

```
var MyTem    = new ActiveXObject("TEMScripting.Instrument")
var Proj     = MyTem.Projection
var m        = Proj.Magnification
```

Please note: Because of the introduction of the Titan microscope, the TEM scripting object (previously called "Tecnai") has become more generic : it is now called TEMScripting.

The variable m now contains the magnification. That's it!
In the same way you can set properties to new values. For example

```
Proj.MagnificationIndex = Proj.MagnificationIndex + 1
```

is essentially the same as increasing the magnification by turning the button on the TEM control pad by one click clockwise.

2.4 A second example (using utility objects)

Some properties do not consist of a ‘simple type’. For example, the image shift is a two-dimensional property, having an x- and an y-component. The ‘utility’ objects are used to handle these, because generally you would like to set and read all components of those properties simultaneously.

The following example shows how to use them (JScript syntax):
Start again by creating access to the projection system

```
var MyTem  = new ActiveXObject("TEMScripting.Instrument")
var Proj   = MyTem.Projection
```

Suppose we want to shift the image in x-direction to +500nm, leaving the y-position at 0:

```
var MyImageShift = new MyTem.Vector(0.5E-6,0)
Proj.ImageShift  = MyImageShift
```

The first statement creates a ‘Vector’ object that is already initialized with the desired the image shift. (Since the ‘Vector’ object is TEM-specific, its creation is also handled by the main ‘Instrument’ object.)
The second line uses the ‘Vector’ to set the image shift. The described way of creating a utility object currently only seems to work in JScript.
Alternatively, if you read the image shift you will of course also get a ‘Vector’ returned:

```
var MyImageShift = Proj.ImageShift
MyImageShift.X   = MyImageShift.X + 0.5E-6.0
Proj.ImageShift  = MyImageShift
```

This will result in a relative image shift in x-direction of 500nm.

Note that setting the image shift x (changing the value of MyImageShift.X alone) to a different value does not do anything on the microscope itself. You have to set the modified vector as a whole back (in other words, in the previous example it is the third line of code that does it).

2.5 TEM modes

There is one important feature of the TEM microscopes inner working that needs to be explained before the TEM objects and properties are described in detail.

Certain microscope parameters such as the intensity setting, the focus value, the spotsize, etc., are dependent on the optical "mode" of the microscope. A "mode" is characterized by a specific setting of the lenses that interact to form the image and the beam. As such, the optical mode is determined by the following parameters that you probably know from working with the microscope:

- the probe setting (microprobe/nanoprobe)
- the projector mode (diffraction/imaging)
- the range of magnification or diffraction (LM, Mi, SA, Mh, LAD, D)
- the lens program (Regular or EFTEM)
- the high tension range (there are three ranges named high, medium and low for microscopes with maximum high tension greater or equal than 200kV)

Some parameters of the optical system change on a transition between modes (i.e., internally, they exist ‘per mode’). For example, when you switch from low LM to Mi magnification, the focus setting will change. It will return to its old ‘LM-value’, if you lower the magnification again. Such parameters are called ‘mode dependent’.

The same will happen with the corresponding properties of the TEM scripting adapter's microscope objects. If you change a mode dependent property, this will only affect the current mode and thus will not affect its values in the other modes. Only the values of the active mode can be accessed and, consequently, switching between modes changes the exposed values of properties.
The adapter objects that possess such mode dependent properties are of course the ones that have something to do with lens settings, i.e. the ‘Illumination’ and the ‘Projection’ objects.

2.6 Remote Scripting

Using TEM Scripting, it is possible to write applications that communicate with a microscope server that runs on another computer. Thus it is possible to control the microscope remotely. For this purpose, you have to have a version of TEM Scripting installed locally. This means that the files stdscript.dll (and for VBScript users also scriptevents.dll) plus all other TEM software files that these files depend on have to be installed on your local machine and registered.

Important note: The acquisiton does not work with remote scripting (the images are memory mapped and that cannot be accessed by the remote PC).

In order to be able to install the TEM Scripting adapter you have to install the regular TEM software on the same PC (many of the regular software files are used during the registration of the scripting adapter and without proper registration the software will not run). Section Testing your Scripting Programs above outlines the options for doing this.

2.6.1 Specifying the microscope server

To access the remote microscope computer, its name has to be known to TEM Scripting (and also possibly to other remote TEM tools that are available at the time of publishing). The name of the remote

server is stored in the system's database, the so-called registry. You can do it yourself, by editing the system registry. (Be careful not to delete or change any other registry entries by accident!):

1. Login as administrator.
2. Open a registry editor: Click the Windows 'Start' button, choose 'run' and type 'regedit' or 'regedt32', click 'ok'.
3. A registry editor should open. Under the key HKEY_LOCAL_MACHINE\SOFTWARE\FEI, add a new sub-key named 'Scripting'.
4. Add a new string value named "RemoteHost" to this key and assign the name of the server to this value (for example "\\ACHT405").
5. Close the registry editor.
6. Logoff and login under your own user account.

From now on, every application that uses TEM Scripting will connect to the remote microscope server. If you have a microscope server installed on your local machine as well, and you want to make contact to the latter again, then you have to set the name of the server in the registry value "RemoteHost" in the registry key HKEY_LOCAL_MACHINE\SOFTWARE\FEI\Scripting to an empty string or delete the whole key again.

2.6.2 Receiving events from a remote microscope server

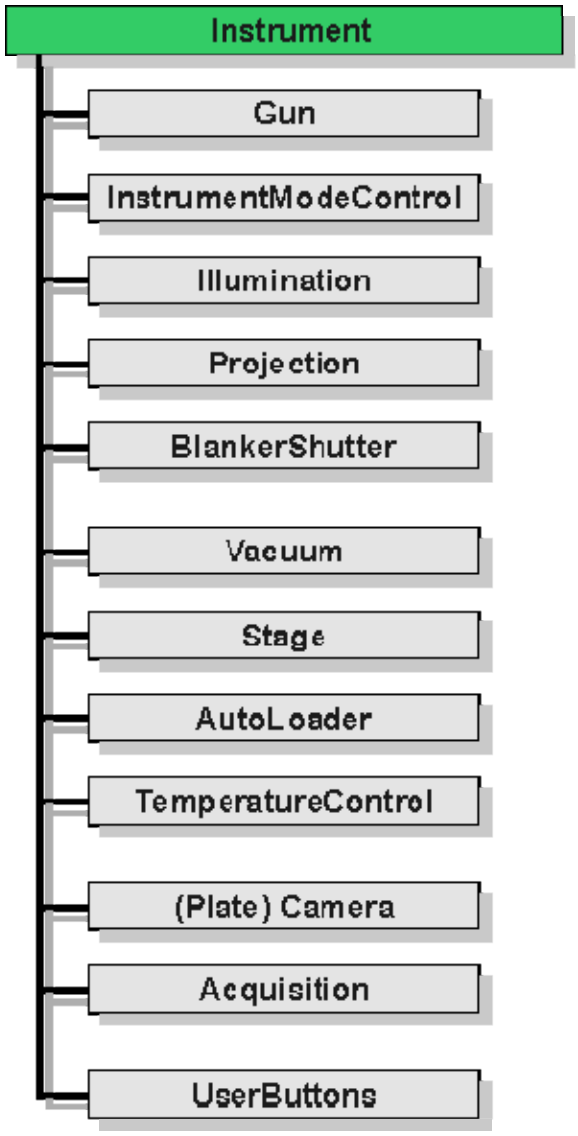
The only events that your application can receive from the microscope server are the events that are fired when the user buttons on the TEM hand panels are used. If you want to receive those events also from a remote microscope, then you may have to add some additional code to your application. The reason for this is security. The communication between application and microscope server uses COM (the Microsoft common object model). COM does a lot of things underwater for you, including initializing security. Now we have the situation that a system service (the microscope server probably runs as such) on a remote computer wants to call a function in your application. COM does not allow that by default. To allow it, you have to call a function named 'CoInitializeSecurity'. This function call has to be made, before COM does this itself underwater, because this function can only be called once per application. Since the details of how and when to call 'CoInitializeSecurity' has to be called are language dependent, we refer here to the chapter 'Scripting in various languages'.

Applications that rely on a browser to run (for example scripts written in JScript or VBScript, or ActiveX components written in any language), cannot do that. They will probably work anyway (because we are able to handle security issues from the microscope server side in this case). Workaround: if 'temserver' service and remote application run under the same user account, then there is no problem. You would have to create a special account for remoting.

3 The TEM Object Model

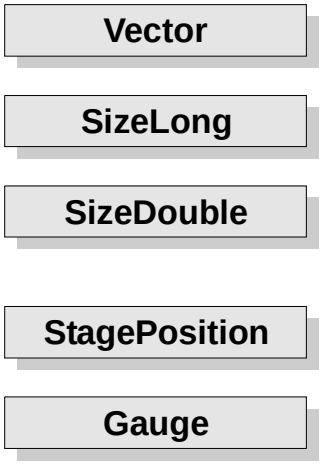
3.1 The microscope objects

The following tree shows the main adapter object (‘Instrument’, the only microscope object to be created by your script) and all secondary objects that can be accessed through it. These secondary objects are modelled as read-only properties of the ‘Instrument’ object. They all relate to a specific part of the microscope and are directly connected to the running microscope. Thus, changes in microscope parameters (for example via the TEM user interface) will affect their properties and vice versa. One of the microscope objects (the ‘UserButtons’) in the object-tree appears in the plural form in the object-tree. In this case the returned object is a collection of 6 ‘UserButton’objects.



3.2 Utility objects

Besides the objects listed above, there are some utility objects that can contain more complex data, for example the stage position, the image shift etc. They are used to handle ‘compound’ or multi-dimensional properties. The utility objects ensure that pieces of information that belong together are read and set simultaneously. These objects are therefore local copies (they exist in your script only). In contrast to the microscope objects they have no connection to the TEM server.



4 The Instrument object

This is the main object of the scripting adapter. It is the only object that can be created directly by your script. Every other microscope object must be assessed as a property of the 'Instrument' object:

```
var MyTem = new ActiveXObject("TEMScripting.Instrument")
```

creates an instance of this object (in JScript syntax).

Property	Description
Acquisition	[Object] Interface to the acquisition of CCD and STEM images.
AutoLoader	[Object] Interface to the Autoloader.
AutoNormalizeEnabled	[Boolean], read/write Enables/disables the automatic normalization procedures performed by the TEM microscope. Normally they are active, but for scripting it can be convenient to disable them temporarily.
BlankerShutter	[Object] Interface to the BlankerShutter ShutterOverrideOn property.
Camera	[Object] Interface to the camera module. Also gives access to control of the fluorescent screens.
Configuration	[Object] Interface to the configuration object that allows querying whether the microscope is a Tecnai or Titan.
Gun	[Object] Interface to the gun deflection and high tension functionality.
Illumination	[Object] Interface to the 'Illumination' system that comprises the condenser lenses, the mini condenser lens, the beam deflection coils and the condenser stigmator. Responsible for beam (or 'probe') properties.
InstrumentModeControl	[Object] Interface to the instrument control functions that allow checking whether the microscope has STEM and, if so, switch between TEM and STEM modes.
Projection	[Object] Interface to the part of the instrument that forms the image, comprising the objective, diffraction and projection lenses, the corresponding stigmators and the image deflectors.
Stage	[Object] Interface to the stage / goniometer functions.
TemperatureControl	[Object] Interface to the temperature controlller.
UserButtons	[Collection of UserButton objects] UserButtons interface to the user buttons L1..L3, R1..R3 of the TEM hand panels.
Vacuum	[Object] Interface to the vacuum system.

Methods	Description
NormalizeAll()	[Void] Normalizes all lenses. To normalize portions of the microscope, refer to the subsystems.

5 The microscope objects

These secondary microscope objects relate to specific parts of the microscope. Through these objects, most of the functionality of the microscope is accessible. They can (and have to) be retrieved as properties of the main ‘instrument’ object. For example

```
var Proj = MyTem.Projection
```

gives you an instance of the ‘Projection’ object (supposing your ‘Instrument’ is named ‘MyTem’).

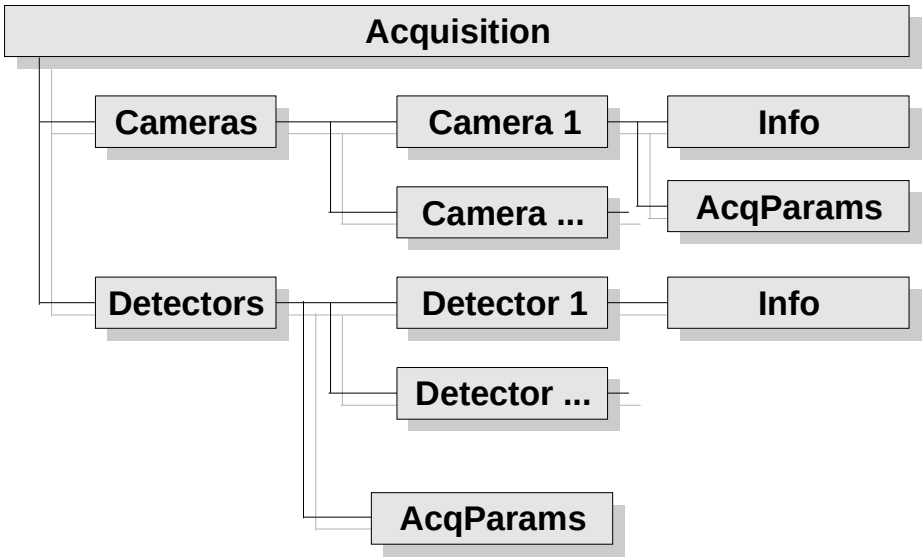
Object	Description
Acquisition	Interfaces to the CCD and STEM acquisition
AutoLoader	Interfaces to the Autoloader
BlankerShutter	Interface for the shutter override function
Camera	Interfaces to the camera and the fluorescent screens
Configuration	Interface to the configuration object that allows querying whether the microscope is a Tecnai or Titan.
Gun	Interfaces to the functionality of the gun and the high tension.
Illumination	Interfaces to the illumination system that forms and manipulates the beam
InstrumentControlMode	Interfaces to the TEM/STEM switch
Projection	Interfaces to the imaging system
Stage	Interfaces to stage and specimen holder
TemperatureControl	Interfaces to the temperature controller
UserButton(s)	Interface(s) to the user button(s), can receive events
Vacuum	Interfaces to the vacuum system

6 The Acquisition object

The Acquisition object gives access to CCD and STEM image acquisition.

Notes:

- **In order for acquisition to be available TIA (TEM Imaging and Acquisition) must be running (even if you are using DigitalMicrograph as the CCD server).**
- **If it is necessary to update the acquisition object (e.g. when the STEM detector selection on the TEM UI has been changed), you have to release and recreate the main microscope object. If you do not do so, you keep accessing the same acquisition object which will not work properly anymore.**



The Acquisition object exposes the CCDCameras and STEMDetectors together with a number of general methods.

The main entry point to the acquisition functionality is the Acquisition interface. From this interface, it is possible to query all available (i.e. installed on the system) acquisition devices: CCD cameras and STEM detectors. For each of these devices it is possible to retrieve an information object, which tells something about (hardware) parameters of the device, for instance, the dimensions of the CCD chip (in pixels).
For each of the CCD cameras, an acquisition parameters object can be retrieved to change the default acquisition settings (i.e. exposure time). In contrast, the STEM acquisition parameters are not device-specific but instead apply to all detectors.

In order to acquire an image from an acquisition device (CCD or STEM detector), the following steps need to be taken:

- TIA (TEM Imaging and Acquisition) application must be running.
- Get the Acquisition object from the main instrument interface of the Standard Scripting component.
- Query available CCD cameras and STEM detectors, looking for the device from which you would like to acquire an image. Note: in the current version of Standard Scripting only devices which are selected in the Microscope User Interface will be available in the query. In future versions of the Standard Scripting we envision the possibility to query and select any acquisition device available in the system, without the need for human interaction with the Microscope User Interface. The software interfaces of the Acquisition objects are already prepared for such extensions.
- Add the queried acquisition device to the list of devices in the Acquisition object. Internally, the Acquisition object maintains a list of acquisition devices on which is has to perform image acquisition when AcquireImages() method is called. You can manipulate this list of devices through Acquisition object interface.
- Acquire images by calling AcquireImages() method on the Acquisition object. The method will acquire the images from all the devices currently found in its internal list of acquisition devices (acquisition is performed sequentially). When acquisition is finished, the method returns an array of acquired images.
- Each of the acquired images has a name property which returns the device name the image was acquired from. The actual image data can be retrieved as a safe-array from the Image object.

6.1 Usage example in a pseudo-programming language

```
// get Acquisition object
  Instrument instrObj = new TEMScripting.Instrument;
  Acquisition acqObj = instrObj.Acqusition();

// Query available acquisition devices
  CCDCameras ccdCollection = acqObj.CCDCameras;
  For (index = 0; index < ccdCollection.Count(); index++)
  {
  CCDCamera ccd = ccdCollection.At(index);
  Print (“found CCD camera: %s“, ccd.Name());
  }

  STEMDetectors stemCollection = acqObj.STEMDetectors;
  For (index = 0; index < stemCollection.Count(); index++)
  {
  STEMDetector stem = stemCollection.At(index);
  Print (“found STEM Detector: %s”, stem.Name());
  }

// Acquire an image from “WAC CCD” camera
  acqObj.AddAcqDeviceByName(“WAC CCD”);
  AcqImages imageCollection = acqObj.AcquireImages();

// since we added only one acquisition device, there will be only
// one image in the image collection
  AcqImage img = imageCollection.At(0);

// get access to pixels
  Array imgData = img.AsSafeArray();

// do something with image data&ldots;
```

Property	Description
Cameras	[Object] Interface to the CCD cameras.
Detectors	[Object] Interface to the STEM detectors.

Method	Description
AddAcqDevice ([in] pDevice)	[Void] Adds an acquisition device to the list of active acquisition devices.
AddAcqDeviceByname ([in] deviceName String)	[Void] Adds the acquisition device with the name specified to the list of active acquisition devices.
RemoveAcqDevice([in] pDevice)	[Void] Removes the acquisition device from the list of active acquisition devices.
RemoveAcqDeviceByName ([in] deviceName String)	[Void] Removes the acquisition device with the name specified from the list of active acquisition devices.
RemoveAllAcqDevices	[Void] Clears the list of active acquisition devices.
AcquireImages	[AcqImages] Acquires the image or images using the currently set list of acquisition devices and returns an interface to the image collection.

6.2 CCDCameras

CCDCameras contains all available CCD cameras on the microscope.

Note: In order for a camera to be "available" it must be selected in the microscope user interface. Currently it is not possible to do this selection through scripting.

6.2.1 CCDCamera

Property	Description
Info	[Object] Interface to the information on the CCD camera.
AcqParams	[Object] Interface to the acquisition parameters of the CCD camera.

6.2.1.1 CCDCameraInfo

Property	Description
Name	[String], read only A string with the name of the CCD as it appears in the microscope's user interface.
Height	[Long], read only The height of the CCD camera in pixels.
Width	[Long], read only The width of the CCD camera in pixels.
PixelSize	[Vector], read only The physical size in X and Y direction of the CCD pixels (that is, the size as they are on the CCD chip, typically a value in the range 15 to 30 micrometers).
Binnings	[SafeArray], read only An array with the binning values supported by the CCD.
ShutterModes	[SafeArray], read only An array with the shutter modes supported by the CCD.
ShutterMode	[AcqShutterMode], read/write The currently selected shutter mode of the CCD.
BinningsAsVariant	[VARIANT], read only An array with binning values supported by the CCD, should be used by JScript and VBScript clients instead of the Binnings property.
ShutterModesAsVariant	[VARIANT], read only An array with the shutter modes supported by the CCD, should be used by JScript and VBScript clients instead of the Binnings property.

Notes:

- Generally the binning values and exposure time are related (quadratically). If you increase the binning from 1 to 4, you normally have to decrease the exposure time by 4^2 in order to prevent CCD saturation. This relation does not exist for the Eagle CCD camera, which uses an optimised scheme where the ratios are as given in the table below.
- The ShutterMode of the CCD camera refers to the shutter(s) being used. The availability of different shutters is dependent on the type of CCD camera (e.g. SIS cameras have none, Gatan cameras typically have pre- and post-specimen, the Eagle has pre- and post-specimen and can also use both shutters simultaneously). The shutter mode is a **global microscope setting**, so if this is changed in a script, you will see the change back in the TEM User Interface (CCD/TV General flap-out). Make sure to retrieve the setting befor running a script and setting it back when closing down, so the user is not confronted with unintended changes to the microscope settings.

Eagle CCD type	Binning values	Exposure time
2k	1 : 2 : 4	1 : 0.5 : 0.125
4k	1 : 2 : 4 : 8	1 : 1 : 0.5 : 0.125

6.2.1.2 CCDAcqParams

Property	Description
ImageSize	[AcqImageSize], read/write The size of the image to be collected.
ExposureTime	[Double], read/write The exposure time in seconds.
Binning	[Long], read/write The binning value to be used for the image acquisition. Make sure the value is one of the supported binning values.
ImageCorrection	[AcqImageCorrection], read/write The type of correction to be applied. Bias/gain correction can only be applied if this has been done in the CCD server prior to scripting.
ExposureMode	[AcqExposureMode], read/write The currently selected exposure mode of the CCD.
MinPreExposureTime	[Double], read only The minimum available pre-exposure time in seconds.
MaxPreExposureTime	[Double], read only The maximum available pre-exposure time in seconds.
MinPreExposurePauseTime	[Double], read only The minimum available pre-exposure pause time in seconds.
MaxPreExposurePauseTime	[Double], read only The maximum available pre-exposure time in seconds.
PreExposureTime	[Double], read/write The pre-exposure time in seconds.
PreExposurePauseTime	[Double], read/write The pre-exposure pause time in seconds.

Note: Pre-exposures can only be be done when the shutter mode is set to "Both". This is only available for Eagle CCD cameras. When pre-exposure is used, the specimen is illuminated for the pre-exposure time defined. The pre-exposure is done by opening the pre-specimen shutter while the post-specimen shutter remains closed to prevent electrons from falling on the CCD. A pre-exposure may help to stabilize specimens (e.g. when charging). When a pre-exposure pause is used, there is a delay (of the defined pause time) inserted between the pre-exposure and the actual CCD exposure. During this delay both shutters are closed.
The exposure mode is **NOT** a global microscope setting, so if this is changed in a script, you will **NOT** see the change back in the TEM User Interface (CCD/TV General flap-out).

6.3 STEMDetectors

STEMDetectors contains all available STEM detectors on the microscope.

6.3.1 STEMDetectors

Property	Description
AcqParams	[Object] Interface to the acquisition parameters of the STEM detectors. Acquisition parameters for STEM are generic and not bound to a particular detector.

Note: In order for a detector to be "available" it must be selected in the microscope user interface. Currently it is not possible to do this selection through scripting.

6.3.2 STEMDetector

Property	Description
Info	[Object] Interface to the information on the STEM detector.

6.3.2.1 STEMDetectorInfo

Note: The maximum size of the unbinned STEM image is 4096² pixels (only if the 4K STEM option has been purchased, if not then the maximum is 2K²).

Property	Description
Name	[String], read only A string with the name of the STEM detector as it appears in the microscope's user interface.
Brightness	[Double], read/write The brightness setting of the STEM detector.
Contrast	[Double], read/write The contrast setting of the STEM detector.
Binnings	[SafeArray], read only An array with the binning values supported by the STEM detector. Technically speaking these are "pixel skipping" values, since in STEM we do not combine pixels as a CCD does, but other than that these values work the same way as in CCD acquisition (e.g. half frame with binning 4 gives a 256² image on a 2k CCD as well as on the STEM).
BinningsAsVariant	[VARIANT], read only An array with binning values supported by the STEM detector, should be used by JScript and VBScript clients instead of the Binnings property.

6.3.2.2 STEMAcqParams

Property	Description
ImageSize	[AcqImageSize] The size of the image to be collected.
DwellTime	[Double] The pixel dwell time in seconds. The frame time equals the dwell time times the number of pixels plus some overhead (typically 20%, used for the line flyback).
Binning	[Long] The binning value to be used for the image acquisition. Make sure the value is one of the supported binning values.

6.4 AcqImages

The AcqImages contains all images acquired (through the AcquireImages; it does not contain a backlist of all previous recordings).

6.4.1 AcqImage

Property	Description
Name	[String], read only The name of the detector (CCD camera or STEM detector) used to acquire the image.
Width	[Long], read only The width (in pixels) of the image.
Height	[Long], read only The height (in pixels) of the image.
Depth	[Long], read only The maximum number of bits in the image. The image as retrieved always has 16 bits, but the original - CCD - image may have had less depth, dependent on the CCD camera used.
AsSafeArray	[SafeArray] read only A SafeArray with the pixel values (32-bit signed integer) of the image.
AsVariant	[VARIANT], read only An array with pixel values (32-bit signed integer) of the image, should be used by JScript and VBScript clients instead of the Binnings property.
AsFile	[String, AcqImageFileFormat, Boolean], read only Image is saved to the specified file in the specified format (see AcqImageFileFormat type below). Boolean allows the image to be normalized, see note below.

AsFile method : The third parameter of the AsFile method is a boolean indicating whether the image should be normalized before saving to file. E.g. if the image has min and max pixel values of 213 and 567 respectively then after normalizing these values are re-scaled to 0 and 65535 when saving to 16-bit png format. The primary purpose of this is to allow easy display of the image in client code without any manipulation (e.g. useful for JScript clients). ***The parameter default is false, which is what is needed if the client wishes the image as it was originally acquired.***

6.5 Acquisition Constants

6.5.1 Enum AcqImageSize

The image sizes supported by the CCD and STEM acquisition are a subset of what is usually available on the microscope. They have been chosen so they are supported by all CCD cameras.

Value	Description
AcqImageSize_Full	Image size covers the whole CCD or STEM range
AcqImageSize_Half	Image size covers half of the CCD or STEM range, centered
AcqImageSize_Quarter	Image size covers one quarter of the CCD or STEM range, centered

6.5.2 Enum AcqImageCorrection

Property	Description
AcqImageCorrection_Unprocessed	CCD images are uncorrected
AcqImageCorrection_Default	CCD images are bias- and gain-corrected

6.5.3 Enum AcqShutterMode

Property	Description
AcqShutterMode_PreSpecimen	The pre-specimen shutter (blanking before the specimen) is used)
AcqShutterMode_PostSpecimen	The post-specimen shutter (blanking after the specimen) is used
AcqShutterMode_Both	Both pre- and post-specimen shutters are used together

6.5.4 Enum AcqExposureMode

Property	Description
AcqExposureMode_None	Default setting
AcqExposureMode_Simultaneous	The pre- and post-specimen shutter are used together
AcqExposureMode_PreExposure	Same as previous but before the actual CCD exposure the specimen is illuminated for the duration of the pre-exposure time set
AcqExposureMode_PreExposurePause	Same as previous but after the pre-exposure and before the actual CCD exposure is a pause fro the duration of the pre-exposure pause time set.

6.5.5 Enum AcqImageFileFormat

Property	Description
AcqImageFileFormat_TIFF	Image is saved in 16-bit TIFF format.
AcqImageFileFormat_JPG	Image is saved in 8-bit JPG format.
AcqImageFileFormat_PNG	Image is saved in 16-bit PNG format.
AcqImageFileFormat_RAW	Image is saved in FEI RAW format.
AcqImageFileFormat_SER	Image is saved in TIA SER format.
AcqImageFileFormat_MRC	Image is saved in MRC format.

6.5.6 AsFile method

This allows clients to request that an acquired image be saved in a specified format and in a specified location. TIFF (16-bit), JPG (8-bit), and PNG (16-bit) are currently supported. Of course this method can be used by any type of client, Scripting or otherwise.

The following JavaScript code snippet shows how the property might be used. For illustration it shows saving a previously acquired image in the three supported formats.

```
// file formats for saving images
var AcqImageFileFormat_TIFF = 0;
var AcqImageFileFormat_JPG = 1;
var AcqImageFileFormat_PNG = 2;

g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.tif", AcqImageFileFormat_TIFF);
g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.jpg", AcqImageFileFormat_JPG);
g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.png", AcqImageFileFormat_PNG);
```

Additionally, the following formats can in principle be used. Subject to proper testing and depending on feedback from customers these formats may be ‘officially added’ at a later date.

```
// these formats still under discussion/development, magic cookie of 123 needed
// to experiment with them
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.raw", 123);
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.ser", 123);
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.mrc", 123);
```

The raw format is an internal format used by FEI; its most likely use is when trying to diagnose problems with images since it allows inspection of the image data ‘exactly’ as it is returned from the server code. The ser format is the officially published TIA format. And the mrc format is the format used by several microscopy related packages; it does not currently include the FEI extended header since only a single image is returned from the scripting component. Any feedback on the merits or otherwise of these formats is welcome (please mail to Dave.Karetnyk@fei.com).

6.6 SafeArray handling

SafeArrays (or variant arrays) are the standard way of marshaling data arrays through COM. In general you should look up the documentation provided with the programming language you are using, but below is some additional information. For using SafeArrays int Delphi, see the code of the example program provided.

C++

Very important: Make sure the declaration for the image array is _variant_t, otherwise you run the risk that C++ clears the data before you can access them. The code snippet below gives an example on how you can access the pixels.

```
if (m_pCamServer) {
  HRESULT hr = m_pCamServer->AcquireImages(&m_pAcquisition);
  if (SUCCEEDED(hr)) {
    int n = m_pAcquisition->Count;
    if (n == 0)
      ShowMessage(_T("No images here"));
    else {
      m_pAcqImage = m_pAcquisition->Item[0];
      if (m_pAcqImage) {
        int h = m_pAcqImage->get_Height();
        int w = m_pAcqImage->get_Width();


        _variant_t img = m_pAcqImage->AsSafeArray();

        CComSafeArray<short> data;
        data.Attach(img.Detach().parray);
        ATL::CWindow pWindow = NULL;
        pWindow = GetDlgItem(IDC_EDIT1);
        CString txt,txt2;
        txt = "";
        for (int i = 0; i<25; i++){
        short pixel = data[i];
        txt2.Format(_T("%s,%i"),txt,pixel);
        txt = txt2;
      }
      pWindow.SetWindowText(txt);
    }

  } else
  ShowMessage(_T("Cannot acquire image(s)"));
}
```

7 The AutoLoader object

The AutoLoader is a system that allows specimen loading/unloading through cartridges from a cassette.

Property	Description
AutoLoaderAvailable	[Boolean], read only Returns whether the AutoLoader is available on the microscope.
Numb erOfCassetteSlots	[Long], read only The number of cassette slots in a cartridge.
SlotStatus ([in] slot Long)	[CassetteSlotStatus], read only The status of the slot specified.

Method	Description
LoadCartridge ([in] fromSlot Long)	Loads the cartride in the given slot into the microscope.
UnloadCartridge()	[Void] Unloads the cartridge currently in the microscope and puts it back into its slot in the cassette.
PerformCassetteInventory()	[Void] Performs an inventory of the cassette (determines which slots are empty or occupied).

7.1 Cassette slot status constants

7.1.1 Enum CassetteSlotStatus

Value	Description
CassetteSlotStatus_Unknown	Cassette slot status has not been determined
CassetteSlotStatus_Occupied	Cassette slot contains a cartridge
CassetteSlotStatus_Empty	Cassette slot is empty
CassetteSlotStatus_Error	Cassette slot generated an error

8 The BlankerShutter object

The BlankerShutter object has only one property, ShutterOverrideOn. This property can be used in cryo-electron microscopy to burn ice off the specimen while blocking the beam from hitting the CCD camera. When the shutter override is true, the CCD camera no longer has control over the microscope shutters. The shutter below the specimen is closed. Whether the beam is on the specimen is determined by the BeamBlanked property of the Illumination object.

Property	Description
ShutterOverrideOn	[Boolean], read/write Determines the state of the shutter override function.

WARNING: Do not leave the Shutter override on when stopping the script. The microscope operator will be unable to have a beam come down and has no separate way of seeing that it is blocked by the closed microscope shutter.

Suggested procedure:

1. Blank the beam using the BeamBlanked property of the Illumination object.
2. Switch the shutter override on.
3. If necessary, wait for a short delay (one second) to allow the system to execute the shuttering.
4. Unblank the beam (the CCD no longer has control).
5. Wait for the time necessary to burn off the ice (sleep, Windows timer, ...)
6. Blank the beam.
7. Switch the shutter override off.
8. If necessary, wait for a short delay (one second) to allow the system to switch the shuttering back to normal.
9. Unblank the beam (the CCD now has control again).

9 The Camera object

Note: The plate camera has become obsolete with Win7 so most of the existing functions are no longer supported.

Property	Description
MainScreen	[ScreenPosition], read/write The position of the main screen. Setting the main screen to the upward position also moves the small screen up (out).
IsSmallScreenDown	[Boolean], read only The Position of the small screen. It cannot be moved down (in) by a software command. It moves up (out) with the main screen.
ScreenCurrent	[Double], read only The current measured on the fluorescent screen (units: Amperes).

9.1 Camera object constants

9.1.1 Enum ScreenPosition

Value	Description
spUnknown	Position of main screen is not known, for example during movement
spUp	Main screen is up (in which case the small screen will also be up)
spDown	Main screen is down

10 The Configuration object

The Configuration object allows you to query whether the microscope is a Tecnai, Titan or a Talos.

Property	Description
ProductFamily	[ProductFamily], read only The type of microscope.
CondenserLensSystem	[CondenserLensSystem], read only For Titan: The type of condenser lens system (two or three condenser lenses).

10.1 Configuration object constants

10.1.1 Enum ProductFamily

Value	Description
ProductFamily_Tecnai	The microscope is a Tecnai
ProductFamily_Titan	The microscope is a Titan
ProductFamily_Talos	The microscope is a Talos

10.1.2 Enum CondenserLensSystem

Value	Description
CondenserLensSystem_TwoCondenserLenses	The microscope is a Titan but has only two condenser lenses. This makes the illumination system similar to that the Tecnai and should be used that way (e.g. the Titan-specific functions like CondenserMode or IlluminatedArea do not work).
CondenserLensSystem_ThreeCondenserLenses	The microscope is a regular Titan

11 The Gun object

Property	Description
Tilt	[Vector], read/write The gun tilt alignment values. Range from -1.0 to +1.0 in x and y directions (logical units). The beamblanker changes the gun tilt. Therefore changing the gun tilt alignment is blocked as long as the beamblanker is active.
Shift	[Vector], read/write The gunshift alignment values. Range from -1.0 to +1.0 in x and y directions (logical units).
HTState	[HTState], read/write The state of the high tension. (The high tension can be on, off or disabled). Disabling/enabling can only be done via the button on the system on/off-panel, not via script. When switching on the high tension, this function cannot check if and when the set high tension value is actually reached.
HTValue	[Double], read/write The value of the HT setting as displayed in the TEM user interface. Units: Volts.
HTMaxValue	[Double], read only The maximum possible value of the HT on this microscope. Units: Volts.

11.1 Gun object constants

11.1.1 Enum HTState

Value	Description
htOff	The high tension is off
htOn	The high tension is on
htDisabled	The high tension is disabled, cannot be switched on by software command

12 The Illumination object

Most of the properties of the illumination object are dependent on the microscope's mode. They distinguish between three optical modes:

1. the microscope is either at low ‘magnification’ (LM or LAD),
2. at higher ‘magnification’ (Mi, SA, Mh, D) in nanoprobe or
3. at higher ‘magnification’ (Mi, SA, Mh, D) in microprobe

Note that the illumination system is independent of the projection system and thus insensitive to diffraction or imaging mode; hence the mixture of diffraction and imaging submodes. If a transition between two of these three modes occurs, then the properties will seem to have changed their values, which reflects their existence per mode.

Property	Description
mode independent	
Mode	[IlluminationMode], read/write Mode of the illumination system (either nanoprobe or microprobe). (Nearly) no effect for low magnifications (LM).
DFMode	[DarkFieldMode], read/write Holds information about whether microscope is in dark field mode and if so, which coordinates are used.
BeamBlanked	[Boolean], read/write Activates/deactivates the beamblanker.
mode dependent	
CondenserStigmator	[Vector], read/write The condenser stigmator setting. Units: logical, range: -1.0 to +1.0.
SpotsizeIndex	[Long], read/write The spot size index (usually ranging from 1 to 11).
Intensity	[Double], read/write Intensity value of the current mode (typically ranging from 0 to 1.0, but on some microscopes the minimum may be higher.)
C3ImageDistanceParallelOffset	[Double], read/write Value of the C3 image distance parallel offset for current mode. This value (three-condenser Titan only) takes the place previously of the Intensity value. The Intensity value changed the focusing of the diffraction pattern at the back-focal plane (MF-Y in Beam Settings control panel) but was rather independent of the illumination optics. As such it changed the size of the illumination but the illuminated area parameter was not influenced. To get rid of this problematic bypass, the C3 image distance offset has been created which effectively does the same focusing but now from within the illumination optics so the illuminated area remains correct.
IntensityZoomEnabled	[Boolean], read/write Activates/deactivates the intensity zoom in the current mode. This function only works, when it has been initialized by means of the microscope alignments (it needs to know at which intensity setting the spot is focused).
IntensityLimitEnabled	[Boolean], read/write Activates/deactivates the intensity limit in the current mode. This function only works, when it has been initialized by means of the microscope alignments (it needs to know at which intensity setting the

	spot is focused).
Shift	[Vector], read/write Beam shift relative to the origin stored at alignment time. Units: meters.
Tilt	[Vector], read/write Dark field beam tilt relative to the origin stored at alignment time. Only operational, if dark field mode is active. Units: radians, either in Cartesian (x,y) or polar (conical) tilt angles. The accuracy of the beam tilt physical units depends on a calibration of the tilt angles.
RotationCenter	[Vector], read/write Corresponds to the alignment beam tilt value. Units are radians, range is ± 0.2-0.3rad. Do not confuse RotationCenter with dark field (Tilt). Be aware that this is an alignment function.
StemMagnification	[Double], read/write The magnification value in STEM mode. You can change the magnification only in discrete steps (the same as on the microscope). If you specify a value that is not one of those steps, the scripting will select the nearest available step.
StemRotation	[Double], read/write The STEM rotation angle (in radians).

The following properties are specific to the Titan microscope.

Property	Description
CondenserMode	[CondenserMode] read/write Mode of the illumination system, parallel or probe.
IlluminatedArea	[Double] read/write The size of the illuminated area (in meters). Accessible only in Parallel mode.
ProbeDefocus	[Double] read/write The amount of probe defocus (in meters). Accessible only in Probe mode.
ConvergenceAngle	[Double] read/write The convergence angle (in radians). Accessible only in Probe mode.

Method	Description
Normalize ([in] IlluminationNormalization Norm)	[Void] Normalizes the condenser lenses and/or the minicondenser lens, dependent on the choice of 'Norm'.

12.1 Illumination object constants

12.1.1 Enum IlluminationMode

Value	Description
imMicroprobe	Microprobe mode
imNanoprobe	Nanoprobe mode

12.1.2 Enum CondenserMode (Titan only)

Value	Description
imParallelIllumination	Parallel illumination mode
imProbeIllumination	Probe illumination mode

12.1.3 Enum DarkFieldMode

Value	Description
dfOff	microscope is in bright field mode
dfCartesian	dark field mode, beam tilt angles are given in x and y tilt directions
dfConical	dark field mode, beam tilt angles are given as radial tilt and rotation angles

12.1.4 Enum IlluminationNormalization

Value	Description
nmSpotsize	normalize lens C1 (spotsize)
nmIntensity	normalize lens C2 (intensity) + C3
nmCondenser	normalize C1 + C2 + C3
nmMiniCondenser	normalize the minicondenser lens
nmObjectivePole	normalize minicondenser and objective
nmAll	normalize C1, C2, C3, minicondenser + objective

13 The InstrumentModeControl object

Property	Description
StemAvailable	[Boolean], read only Returns whether themicroscope has a STEM system or not.
InstrumentMode	[InstrumentMode], read/write Switches between TEM and STEM modes.

13.1 InstrumentModeControl object constants:

13.1.1 Enum InstrumentMode:

Value	Description
InstrumentMode_TEM	TEM mode
InstrumentMode_STEM	STEM mode

14 The Projection object

Some of the properties of the projection object are dependent on the microscope's optical mode. There are 5 types of mode dependencies (the two last of which are only introduced by functions created for easier handling of the modes) :

Type A differentiates between 5 TEM modes for every lens program:
- diffraction LAD (the mode is reached, when the projector is switched to diffraction from LM imaging)
- diffraction D (the mode is reached, when the projector is switched to diffraction from imaging with higher magnifications)
- imaging with low magnifications (LM)
- imaging with higher magnification and illumination system in microprobe mode
- imaging with higher magnification and illumination system in nanoprobe mode

Type B differentiates between 2 modes (irrespective, of whether microscope is switched into diffraction or imaging):
- low ‘magnification’ (LM, LAD)
- higher ‘magnification’ (Mi...Mh, D)

Type C differentiates between 3 modes:
- low ‘magnification’ (LM or LAD),
- higher ‘magnification’ (Mi, SA, Mh, D) in nanoprobe
- higher ‘magnification’ (Mi, SA, Mh, D) in microprobe

Type D differentiates between 3 modes:
- diffraction LAD
- diffraction D
- imaging

Type E differentiates between 6 modes:
- the four imaging submodes LM, Mi, SA, Mh
- the two diffraction submodes LAD, D

If a transition between modes occurs, then the properties will seem to have changed their values, which reflects their existence per mode.

Property	Description
mode independent	
Mode	[ProjectionMode], read/write Main mode of the projection system (either imaging or diffraction).
SubMode	[ProjectionSubMode], read only Submode of the projection system (either LM, Mi, ..., LAD or D). The imaging submode can change, when the magnification is changed.
SubModeString	[String], read only Submode of the projection system, given as a string. To be used as alternative to ‘SubMode’.
LensProgram	[LensProg], read/write The lens program setting (currently EFTEM or Regular). This is the third property to characterize a mode of the projection system.

Magnification	[Double], read only The reference magnification value (screen up setting). Use the 'MagnificationIndex' to change it, since the magnification can only be changed in discrete steps and the values may vary. 0, if microscope is in diffraction.
MagnificationIndex	[Long], read/write The magnification values are indexed (from minimal to maximal magnification). Increasing or decreasing the index is what the magnification button on the hand panel does. In the process the imaging 'SubMode' may change. Note: On some microscopes the magnification ranges of LM and Ml submodes may overlap. Thus at this submode border the magnification value does not necessarily rise with increasing index. 0, if microscope is in diffraction.
ImageRotation	[Double], read only The rotation of the image or diffraction pattern on the fluorescent screen with respect to the specimen. Units: radians.
DetectorShift	[ProjectionDetectorShift], read/write Sets the extra shift that projects the image/diffraction pattern onto a detector.
DetectorShiftMode	[ProjDetectorShiftMode], read/write This property determines, whether the chosen DetectorShift is changed when the fluorescent screen is moved down.
mode dependency type A	
Focus	[Double], read/write Focus setting of the currently active mode. Range: maximum between -1.0 (= underfocussed) and 1.0 (= overfocussed), but the exact limits are mode dependent and may be a lot lower.
Defocus	[Double], read/write Defocus value of the currently active mode. Changing 'Defocus' will also change 'Focus' and vice versa. 'Defocus' is in physical units (meters) and measured with respect to a origin that can be set by using 'ResetDefocus()'.
ObjectiveExcitation	[Double], read only The excitation of the objective lens in percent.
mode dependency type B	
CameraLength	[Double], read only The reference camera length (screen up setting). Use the 'CameraLengthIndex' to change it, since it can only be changed in discrete steps and the values may vary, for example with high tension. always 0, if microscope is in imaging.
CameraLengthIndex	[Long], read/write The camera length values are indexed (from minimal to maximal camera length). Setting the Index thus is equivalent to setting the camera length. The series of camera length values for LAD and for D are not identical. always 0, if microscope is in imaging.
ObjectiveStigmator	[Vector], read/write The objective stigmator setting. Range: -1.0 to +1.0 for x and y.
DiffractionStigmator	[Vector], read/write The diffraction stigmator setting. Range: -1.0 to +1.0 for x and y.

DiffractionShift	[Vector], read/write The diffraction pattern shift with respect to the origin that is defined by alignment. Units: radians.
ImageShift	[Vector], read/write The image shift with respect to the origin that is defined by alignment. Units: meters.
mode dependency type C	
ImageBeamShift	[Vector], read/write Image shift with respect to the origin that is defined by alignment. The apparent beam shift is compensated for, without affecting the Shift-property of the Illumination-object. Units: meters. Attention: Avoid intermixing ImageShift and ImageBeamShift, otherwise it would mess up the beam shift (=Illumination.Shift). If you want to use both alternately, then reset the other to zero first.
ImageBeamTilt	[Vector], read/write Beam tilt with respect to the origin that is defined by alignment (rotation center). The resulting diffraction shift is compensated for, without affecting the DiffractionShift-property of the Projection object. For proper operation requires calibration (alignment) of the Beam Tilt - Diffraction Shift (for more information, see a0050100.htm on the TEM software installation CD under privada\beamtiltdiffshift). Units: radians. Attention: Avoid intermixing Tilt (of the beam in Illumination) and ImageBeamTilt. If you want to use both alternately, then reset the other to zero first.
mode dependency type D	
ProjectionIndex	[Long], read/write This index always contains a value. It corresponds to the camera length index or the magnification index, dependent on the microscope mode.
mode dependency type E	
SubModeMinIndex	[Long], read only The minimum ProjectionIndex of the current submode. Check this if you want to change the ProjectionIndex or MagnificationIndex but do not want to leave the submode.
SubModeMaxIndex	[Long], read only The maximum ProjectionIndex of the current submode. Check this if you want to change the ProjectionIndex or MagnificationIndex but do not want to leave the submode.

Method	Description
ResetDefocus()	[Void] Resets the current ‘Defocus’ to 0 nm. This does not change the ‘Focus’ value (the focussing lens current). Use it when the image is properly focussed to adjust the ‘Defocus’ scale.
ChangeProjectionIndex ([in] Long Steps)	Changes the current Index by ‘Steps’.
Normalize ([in] ProjectionNormalization Norm)	[Void] Normalizes the objective lens or the projector lenses, dependent on the choice of ‘Norm’.

14.1 Projection object constants

14.1.1 Enum ProjectionMode

Value	Description
pmImaging	Projector in imaging mode
pmDiffraction	Projector in diffraction mode

14.1.2 Enum ProjectionSubMode

Value	Description
psmLM	Imaging mode, low magnification
psmMi	Imaging mode, lower intermediate magnification range
psmSA	Imaging mode, high magnification
psmMh	Imaging mode, highest magnification range
psmLAD	Diffraction, LAD mode (the mode entered from LM imaging)
psmD	Diffraction mode as entered from higher magnification imaging modes

14.1.3 Enum LensProg

Value	Description
lpRegular	The default lens program
lpEFTEM	Lens program used for EFTEM (energy-filtered TEM)

14.1.4 Enum ProjectionDetectorShift

Value	Description
pdsOnAxis	Does not shift the image/diffraction pattern
pdsNearAxis	Shifts the image/diffraction pattern onto a near-axis detector/camera
pdsOffAxis	Shifts the image/diffraction pattern onto an off-axis detector/camera

14.1.5 Enum ProjDetectorShiftMode

Value	Description
pdsmAutoIgnore	The 'DetectorShift' is set to zero, when the fluorescent screen moves down. When it moves up again, what happens depends on what detector TEM thinks is currently selected. Take care!.
pdsmManual	The detectorshift is applied as it is chosen in the 'DetectorShift'-property
pdsmAlignment	The detector shift is (temporarily) controlled by an active alignment procedure. Clients cannot set this value. Clients cannot set the 'DetectorShiftMode' to another value either, if this is the current value. They have to wait until the alignment is finished.

14.1.6 Enum ProjectionNormalization

Value	Description
pnmObjective	Normalize objective lens
pnmProjector	Normalize Diffraction, Intermediate, P1 and P2 lenses
pnmAll	Normalize objective, diffraction, intermediate, P1 and P2 lenses

15 The Stage object

Property	Description
Status	[StageStatus], read only The current state of the stage. Check this to determine the whether the stage is ready to perform a 'GoTo' or a 'MoveTo'.
Position	[StagePosition], read only The current position of the stage. (Changing the position can be done in various ways. Use the 'GoTo' and 'MoveTo' Methods described below.)
Holder	[StageHolderType], read only The current specimen holder type.
AxisData	([in] mask StageAxes) [StageAxisData], read only Interface to a StageAxisData object that contains the minimum and maximum values available for the particular axis.

Method	Description
GoTo ([in] StagePosition NewPosition, [in] StageAxis AxesBits)	[Void] Makes the holder move to the position given by 'NewPosition'. 'AxesBits' contains information about which axes are to be used to perform the movement. This is useful to suppress unnecessary movements of the holder due to fluctuations in position measurement. Also, you do not have to bother to give the correct parameters for axes-settings that you do not want to change. (In Delphi, this function is automatically renamed to GoTo_, because GoTo is a reserved keyword.)
GoToWithSpeed ([in] StagePosition NewPosition, [in] StageAxis AxesBits, [in] Speed Double)	[Void] Makes the holder move to the position given by 'NewPosition'. 'AxesBits' contains information about which axes are to be used to perform the movement. This is useful to suppress unnecessary movements of the holder due to fluctuations in position measurement. Also, you do not have to bother to give the correct parameters for axes-settings that you do not want to change. (In Delphi, this function is automatically renamed to GoTo_, because GoTo is a reserved keyword.) The speed is a fraction of the standard speed setting (so 1.0 means the standard speed setting). In the Delphi Exampler program the lower limit allowed is 0.1% (but this limit is not in scripting itself).
MoveTo ([in] StagePosition NewPosition, [in] StageAxis AxesBits)	[Void] Makes the holder move in a way that all possible positions can be reached without touching the objective pole. Actually, the following sequence of movements is performed (a, b denote alpha and beta tilts, x,y,z the x,y and height positions, large letters the new position): b->0, a->0, z->Z, x,y->X,Y, 0->A, 0->B. If 'AxesBits' determines that a tilt axis (A or B) is not to be included into the movement, then it will still move, but return to its old setting after X,Y,Z have finished their movement.

15.1 StageAxisData object

The StageAxisData object can be used to retrieve the minimum and maximum values allowed for a particular stage axis. This is really only useful for the dual-axis tomography holder as there the b tilt (which controls the flipflop motion) has hardware-dependent values. In all other cases the minimum and maximum values are as follows:

- X and Y -1000 to +1000 (micrometers)
- Z -375 to 375 (micrometers)
- a -80 to +80 (degrees)
- b -29.7 to +29.7 (degrees)

Property	Description
MinPos	[Double], read only The lowermost value of the stage position allowed on the axis.
MaxPos	[Double], read only The uppermost value of the stage position allowed on the axis.
UnitType	[MeasurementUnitType], read only The unit (meters or radians) of the MinPos and MaxPos values

15.2 Stage object constants

15.2.1 Enum StageStatus

Methods	Description
stReady	The stage is ready (capable to perform all position management functions)
stDisabled	The stage has been disabled either by the user or due to an error.
stNotReady	The stage is not (yet) ready to perform position management functions for reasons other than already accounted for by the other constants
stGoing	The stage is performing a ‘GoTo()’
stMoving	The stage is performing a ‘MoveTo()’
stWobbling	The stage is wobbling

15.2.2 Enum StageAxes

The ‘GoTo’ and ‘MoveTo’ methods require a parameter (of type long) that contains bitwise information about which axis is to be involved in the movement. The bit order is BAZYX , so bit 0 contains the information about whether the X-axis is involved, bit 4 contains the information about the B axis. The members of the ‘StageAxes’ enumeration can be used instead of calculating with bits. You can combine them by bitwise ‘OR’s (i.e. in JScript: MyAxBits = (axisXY | axisA | axisB) to allow the X,Y,A,B axis to move, but leave the Z constant.

Value	Description
axisX (= 1)	Use X-axis
axisY (= 2)	Use Y-axis
axisXY (= 3)	Use X- and Y-axis
axisZ (= 4)	Use Z-axis
axisA (= 8)	Use alpha tilt-axis
axisB (=16)	Use B-axis, usually the beta tilt, but on Dual-Axis Tomography holders this is the second (rotation or flip-flop) axis.

15.2.3 Enum StageHolderType

Value	Description
hoInvalid	The ‘invalid’ holder. No holder has been selected yet or the current selection has become invalid
hoSingleTilt	Single tilt holder
hoDoubleTilt	Double tilt holder
hoNone	Holder is removed
hoPolara	Non-removable Polara holder
hoDualAxis	Dual-axis tomography holder

15.2.4 Enum MeasurementUnitType

Value	Description
MeasurementUnitType_Unknown	Unknown unit type
MeasurementUnitType_Meters	Unit type is meters (linear axes)
MeasurementUnitType_Radians	Unit type is radians (tilt axes)

16 The TemperatureControl object

The Temperature controller is a system that monitors temperatures and refrigerant levels for the AutoLoader and column dewars.

Property	Description
TemperatureControlAvailable	[Boolean], read only Returns whether the Temperature controller is available on the microscope.
RefrigerantLevel([in] rl: RefrigerantLevel)	[Double], read only The level of coolant of the dewar specified.
DewarsRemainingTime	[Long], read only Returns remaining time until the next (automatic) dewar refill. Returns -1 if no refill is scheduled (e.g. All room temperature, or no dewar present).
DewarsAreBusyFilling	[Boolean], read only Returns TRUE if any of the available dewars is currently busy filling.

Method	Description
ForceRefill	[Void] Forces a refill of the refrigerant. Notes: 1. This function takes considerable time to execute. 2. If the refrigerant level in the dewar specified is above 70% then ForceRefill will not do anything and return almost immediately.

16.1 Refigerant level constants

16.1.1 Enum RefrigerantLevel

Value	Description
RefrigerantLevel_AutoLoaderDewar	The dewar of the AutoLoader
RefrigerantLevel_ColumnDewar	The dewar on the microscope column
RefrigerantLevel_HeliumDewar	The liquid-Helium dewar

17 The UserButton object

'UserButton' objects are members of the 'UserButtons' collection:

```
var MyL1 = MyTem.UserButtons("L1")
```

retrieves an object that represents -and references to- the user button L1 (supposing that 'MyTem' is an instance of an 'Instrument' object).

Property	Description
Name	[String], read only The name of the button (i.e. "L1", "L2", etc.).
Label	[String], read only The current label that is assigned to the button (not necessarily assigned by your script)
Assignment	[String], read/write The script-assigned label of the button. Setting this property directs the events generated by use of the button to the script. If the script has made an assignment, then 'Assignment' will equal 'Label', if not temporarily overwritten by somebody else, for example an alignment procedure. (In that case, no events will be received.) Default value = "". Setting this property to an empty string or a string consisting only of blanks (that would be invisible in the UI) restores the situation that was there, before the script took control over the button.

Event	Description
Pressed()	Event that is raised when the button is pressed.

18 The Vacuum object

Property	Description
Status	[VacuumStatus], read only Status of the vacuum system (see below).
ColumnValvesOpen	[WordBool], read/write The status of the column valves.
PVPRunning	[WordBool], read only Checks whether the prevacuum pump is currently running (consequences: vibrations, exposure function blocked or should not be called).
Gauges	Collection of gauge objects, giving information about the actual pressures.

Method	Description
RunBufferCycle()	[Void] Runs a pumping cycle to empty the buffer. This function may take quite some time, so be careful when using it in a single-threaded application with user interface. (The program appears to be hanging while it waits for the function to return.)

18.1 Constants:

18.1.1 Enum VacuumStatus:

Value	Description
vsUnknown	Status of vacuum system is unknown
vsOff	Vac uum system is off
vsCameraAir	Camera (only) is aired
vsBusy	Vacuum system is busy, that is: on its way to ‘Ready’, ‘CameraAir’, etc.
vsReady	Vacuum system is ready
vsElse	Vacuum is in any other state (gun air, all air etc.), and will not come back to ready without any further action of the user

19 The utility objects

'Utility' objects are used to handle compound (or multi-dimensional) properties. They can be created by your script (except for the 'gauge' that is 'read only'), but since they are specific TEM objects, this creation is also handled by the main 'Instrument' object. In JScript this may look as follows (supposed that your 'Instrument' is named 'MyTem':

```
var MyVector = new MyTem.Vector(2.0,3.0)
```

This code creates a TEM vector object MyVector that is initialized with (2.0,3.0).
Of course, reading a compound property, for example

```
var MyOtherVector = MyTem.Gun.Shift
```

will also return an instance of an utility object, in this case a vector containing the actual gun shift.

Utility object	Description
Vector	Object used to access 2-dimensional compound properties
StagePosition	Object used to read and set the position of the stage
Gauge	Object used to read information about pressures and the status of the corresponding measurement devices

19.1 The Gauge (utility object)

The gauge objects are used to retrieve information about the vacuum system measurement devices and the actual pressures measured with them. Since this a 'read only' task, you cannot create instances of this utility object yourself. 'Gauge' objects are always members of the vacuum systems 'Gauges' collection:

```
var MyGauge = MyTem.Vacuum.Gauges("P1")
```

This retrieves the latest pressure and status information for the vacuum buffer. 'P1' is the name of the measurement device associated with the buffer, as you can see from the TEM vacuum overview.

Note: On Tecnai systems so called 'virtual gauge elements', named 'PXX', were supported where XX represents a number. For instance 'P4' could be used instead of 'IGP1'. Such virtual/pseudo names are no longer supported – the gauge name as it appears on the user interface must be used.

Property	Description
Name	[String], read only Name of the gauge.
Status	[GaugeStatus], read only The status of the gauge. It is important for the interpretation of the pressure values.
Pressure	[Double], read only Last measured pressure for this gauge. Units: Pascal
PressureLevel	[GaugePressureLevel], read only Indicates, in which range the pressure lies. Actions of the vacuum system depend on this property.

Method	Description
Read	[Void] Forces a read of the pressure level. Execute before retrieving the pressure, otherwise the pressure may not be up-to-date.

19.1.1 Gauge object constants

19.1.1.1 Enum GaugeStatus:

Value	Description
gsUndefined	No information on the gauge available
gsUnderflow	Underflow, pressure is lower than can be measured with this gauge, the measurement is not interpretable
gsOverflow	Overflow, pressure is higher than can be measured with this gauge, the measurement is not interpretable
gsValid	Valid, the pressure measurement is valid and interpretable
gsInvalid	Invalid, the pressure measurement is invalid and not interpretable

19.1.1.2 Enum GaugePressureLevel

The pressure range for a every gauge is divided into four regions (low.....high). Depending on the gauge, a change in pressure - and thus a switching from one level to another - may result in an action of the vacuum system. (Example: On the Tecnai a buffer cycle will start, when the gauge ‘P1’ reaches plGaugePressurelevelMediumHigh.)

Value	Description
plGaugePressurelevelUndefined	Gauge is not active
plGaugePressurelevelLow	Lowest pressure range
plGaugePressurelevelLowMedium	Low to medium pressure range
plGaugePressurelevelMediumHigh	Medium to high pressure range
plGaugePressurelevelHigh	Highest range

19.2 The StagePosition (utility object)

Property	Description
X	[Double], read/write Position of X-axis, in meters.
Y	[Double], read/write Position of Y-axis, in meters.
Z	[Double], read/write Position of Z-axis, in meters.
A	[Double], read/write Position of A-axis (alpha-tilt), in radians
B	[Double], read/write Position of B-axis (beta-tilt), in radians

Method	Description
GetAsArray ([out] Double[5] Position)	[void] Returns the properties of the StagePosition as an array of 5 doubles. The ordering is [X,Y,Z,A,B].
SetAsArray ([in] Double[5] Position)	[void] Sets the properties of the StagePosition from an array of 5 doubles. The ordering is [X,Y,Z,A,B].

19.3 The Vector (utility object)

This is a general object used to retrieve and set two-dimensional properties, the components of which have to be read or set simultaneously.

The vector object must be used for tilt, shift, and stigmator properties. X and Y must then be given in the appropriate units (ie radians, meters or logical).

Property	Description
X	[Double], read/write; The x value.
Y	[Double], read/write; The y value.

20 TEM constants

TEM-specific constants are defined in the following enumerations that are used by the objects in the right column of the table:

enum	object
AcqImageCorrection	CCDAcqParams object
AcqImageSize	CCDAcqParams object, STEMAcqParams object
CassetteSlotStatus	AutoLoader object
DarkFieldMode	Illumination object
GaugeStatus	Gauge object
GaugePressureLevel	Gauge object
HightensionState	Gun object
IlluminationMode	Illumination object
IlluminationNormalization	Illumination object
InstrumentMode	InstrumentModeControl object
LensProg	Projection object
MeasurementUnitType	StageAxisData object
MinicondenserMode	Illumination object
ProjDetectorShiftMode	Projection object
ProjectionDetectorshift	Projection object
ProjectionMode	Projection object
ProjectionNormalization	Projection object
ProjectionSubMode	Projection object
RefrigerantLevel	TemperatureControl object
ScreenPosition	Camera object
StageHolderType	Stage object
StageStatus	Stage object
StageAxes	Stage object
TEMScriptingError	All
VacuumStatus	Vacuum object

21 TEM error codes

There are TEM-specific error codes available for errors that are due to microscope specific error conditions and specific errors raised by the TEM software. There will also be support for retrieving more information about the error that occurred (descriptive string, source etc.). How this can be done is described in the chapter "Scripting in various languages". These error codes are the following:

21.1 Enum TEMScriptingError:

Value	Description
E_VALUE_CLIP	a parameter exceeded its possible range, it is clipped to its highest or lowest allowed value
E_OUT_OF_RANGE	a parameter exceeded its possible range. The command is ignored.
E_NOT_OK	another error occured. Query the error object for more information.

Only a few functions return these error codes. Mostly, standard Microsoft OLE-error codes are returned. However, they are used in a specific way. These standard codes are displayed below in hexadecimal format.
Scripting languages may translate the standard error values into others, as indicated in the table below. In Delphi apparently the error codes cannot easily be retrieved, but the standard describing string is well available. These descriptions are pretty much the same for all languages and close to what the meaning of the error in the TEM-context is. Only for E_UNEXPECTED you will get a message like "catastrophic failure", which does not quite fit to the error description given in the table.

Error	C++	JScript	VBScript	VB	Description
E_UNEXPECTED	8000FFFF	8000FFFF	8000FFFF	8000FFFF	This error is generally raised, if you do a function call that is not allowed in the current state of the microscope, and you could in principle have known about this because the information was available.
E_NOTIMPL	80004001	800A01BD	1BD	1BD	Function not yet implemented (Should not happen for TEM scripting functions).
E_INVALIDARG	80070057	800A0005	5	5	The arguments you supplied for a function call are of the wrong type or out of range
E_ABORT	80004004	80004004	80004004	11F	An action was started (i.e. an exposure), but then aborted
E_FAIL	80004005	80004005	80004005	80004005	Unspecified failure.
E_ACCESSDENIED	80070005	800A0046	46	46	If this error is raised a TEM component cannot be accessed, either because it is not there or because a hardware or software of this component is in an error state.

22 TEM-specific issues

When writing scripts for the TEM, it is of course important to keep in mind that there is a physical microscope involved. Changing a property of an adapter object mostly results in a physical action on the actual TEM. This also means that no matter how many objects of one type are created, be it in one or also in several scripts running in parallel, they finally all correspond to the same thing. And if somebody sitting at the microscope or another script decides to change microscope parameters, the properties of your objects may change without you knowing. Apart from this perhaps trivial -but easily forgotten- aspect there are some details about the TEM's inner structure that may be necessary to know when programming.

We already introduced the

- Microscope modes.

This concept had to be known in order to understand the behavior of some objects properties. Occasionally you may run into trouble, because the TEM microscope does some things for you that you are not aware of:

- Normalizations
- TEM sessions

It may also be worthwhile to have a look at the notion of

- Synchronous functions

Finally, consider

- Setting parameters out of range

22.1 Synchronous functions

Although adapter-functions are programmed to be synchronous, the action of setting a property to a certain value may in some special cases trigger further reactions of the microscope. These cases are:

22.1.1 Automatic normalization

Certain changes in lens excitations trigger a normalization procedure that can take up to two seconds.

22.1.2 Pole touch

During movement of the stage (especially with a microscope equipped with UTWIN objective lens) a pole touch may be detected. The stage will then try to solve this problem.

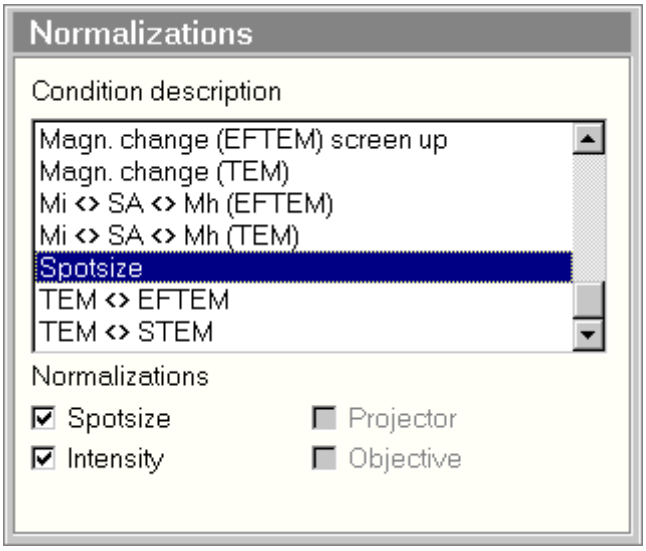
These are situations in which the microscope, or rather parts of it, sometimes behave as independent users themselves. This hampers the use of synchronous functions. But there are remedies. For the first situation they are explained in the section on Normalizations, the second situation can often be avoided by using ‘MoveTo’ in place of ‘GoTo’ in case you want to reach extreme positions of the stage.

22.1.3 Setting the high tension

Setting the high tension or switching it is considered done, when the command is delivered to the electronics (as is in fact also the case with most other commands). However, switching and setting the high voltage may need a considerable additional time. It also has to be checked whether the action was successful.

22.2 Normalizations

Many actions that change excitations of optical elements trigger normalization procedures to ensure reproducibility of the settings (since the optics contain magnetic lenses). You can examine the ‘Normalizations’ control panel in the TEM user interface to get information about when normalization procedures are automatically initiated (see image below).



On the Tecnai these settings are user-dependent (note the checkboxes in the control cluster that allow individual settings to be made). The example in the image shows that -for this user- automatic normalizations are invoked for C1 and C2 condenser lenses, if the spotsize is changed.

The automatic normalization procedures are done independently from any other action. They take about two seconds on Tecnai and five seconds on Titan. This disturbs the adapter's concept of using synchronous functions in order to allow a simple order of commands.

Remedy: In scripting applications it can be helpful to disable automatic normalizations and invoke the necessary normalizations via the script, using the (synchronous) ‘Normalize()’ methods of the ‘Illumination’ and ‘Projection’ objects. Do not forget to enable autonormalizations again before ending the application!

Example:

```
MyTem.AutoNormalizeEnabled = FALSE
var Illum = MyTem.Illumination
Illum.SpotsizeIndex = 3
Illum.Normalize(nmCondenser)
MyTem.AutoNormalizeEnabled = TRUE
```

Please note that entering the alignment procedures via the TEM user interface overrides the setting of the ‘AutoNormalizeEnabled’-property.

22.3 TEM sessions

The TEM software architecture supports the concept of sessions and user levels. If a user is logging on (starting a session), user-specific data (preferences, alignment settings etc.) are read from the registry. User rights are determined as well. In turn, some data are updated when the user logs off again. Logging on/off here means, that somebody is actually starting up/closing down the user interface of the TEM microscope, and is not to be confused with logging on/off in Windows NT.

You may have noticed, that already without a session started, that is without the TEM user interface running, your script can communicate with the TEM server. But because of the facts mentioned above, you may run into problems. So, for every application that does more than passive protocolling tasks, it is strongly recommended that a TEM session is initialized before the script is executed!

22.4 Setting parameters out of range

Generally, if you set a certain property to a value that is valid by type (i.e. a double), but exceeds the physically allowed limits (i.e. setting the magnification to 2x) this property should take its maximal or minimal value. Exceptions from this rule can be expected if the action takes a lot of time and is probably erroneous anyway. In these cases an error will be raised.
The stage, for example will not react on setting its x-position to +100mm, but will raise an error to indicate that something is wrong.

This behavior is not yet implemented consistently in the microscope server. You will probably find properties that will not react at all, if you feed out of range parameters. To make things a little more difficult, it is not always easy to tell what the allowed range of values for a property is, because it may depend on the alignment settings of the microscope.
One of the more complicated cases is the beam-deflection setting, for example. One single set of coils executes two different kinds of actions: beam shift and beam tilt. Both shift and tilt are further divided into an alignment-value (shift not accessible through the TEM scripting adapter; tilt = rotation center) and a user value (also accessible via the ‘Illumination’ object). Additionally the beam-deflection coils are affected by the alignment of the condenser stigmator. Shifts, tilts and stigmator adjustments are translated through pivot points into settings on the upper and lower deflection coils. These coils ultimately trigger an out-of-range response. This complexity makes it very difficult to predict what the actual limit is for a specific value. To determine the actual limit for a parameter that does not change on an attempt to set it to an out-of-range value, the following procedure can be used (in JScript code):

```
//Applies to setting between -1 and +1
//moving in positive direction
var step = 0.1;
var CurrentValue = ..... ;
      // get the current setting
do
    var CheckValue = CurrentValue;
    //try increase:
    Currentvalue = CurrentValue + step;
    if (Math.abs(Checkvalue - CurrentValue) < 1E-06)
    {
       //nothing happened - reduce stepsize
       step = step/2;
    }
    else
     {
       step = step*2;
    }
while (Math.abs(step)< 1E-04); //value arbitrarily chosen
```

It is advisable to rather check a property's value before relying on the success of an attempt to set it. Anyway, another script or human user could also in principle interfere at all times, so checking can never harm.

23 Property- and function- types

Throughout this help, the types of properties (and also the return types of functions) have been described in a more or less symbolic way (for example as ‘Boolean’, ‘String’, ..). These descriptions differ from programming language to programming language and may thus not be clear to everybody. The following table holds a description and a ‘translation’ to common scripting and programming languages, that use typing (JScript and VBScript doe not declare the types of variables).

‘Symbolic’ type used here	Visual Basic	Delphi	C++	Description
Boolean	Boolean	boolean	BOOL	a Boolean type (having values "TRUE" and "FALSE" that are actually transmitted as -1 and 0)
String	String	widestring	BSTR (or _bstr_t)	a wide, double-byte (Unicode) string on 32-bit Windows platforms
Long	Long	long	long	a 32 bit integer
Double	Double	double	double	a 64 bit real
Void	- (sub)	- (procedure)	void	keyword specifying that there is no return-type
Object	Object	ComObject	IDispatch*	a variable that refers to an object that has its own properties and methods. (Under water it contains a pointer to the default interface of something called a COM-object, where COM = component object model.)

24 TEM scripting in C++

For the explanations in this chapter, it is assumed that you are going to create a scripting client using Microsoft Developer Studio (MSDEV) and the Microsoft Foundation Classes (MFC), as it is done in the example program delivered with the adapter.

24.1 Import the dynamic link library

In order to be able to compile your project (at least if you use the hard types defined in the adapter) you have to include the typelib (or the DLL):

```
#import "stdscript.tlb" named_guids
using namespace TEMScripting;
```

The above statement assumes that the typelib is in the same directory as your source files. From the typelib, the compiler will generate the stdscript.tlh and .tli files that contain all necessary definitions and a number of wrapper-functions. You will then also get the full support of the ClassView and the IntelliSense mechanism.

In C++, the names of the ‘objects’ can be different from what was written in the remainder of this help, because unlike most scripting languages, C++ knows the difference between objects and interfaces implemented by them. Due to details in the adapter's implementation that are meant to make its use consistent in scripting languages, you see that most interfaces are named as are the ‘objects’ in the remainder of this help file, but that, as a C++ user, you have to replace ‘Instrument’ by ‘InstrumentInterface’ and Userbutton by ‘IUserButton’.
Your application must also initialize the COM-libraries. Using MFC, this is typically done by the following code

```
if (!AfxOleInit())
{
  AfxMessageBox(IDP_OLE_INIT_FAILED);
  return FALSE;
}
```

that you have to add to your Applications ::InitInstance() if you did not check the ‘automation’-check box, when creating it.

24.2 Create an ‘Instrument’ and get the secondary microscope object that you need

The following piece of code shows one possible way to get access to the projection system and the stage:

```
InstrumentInterfacePtr MyInstrument;
ProjectionPtr          MyProjection;
StagePtr               MyStage;
MyInstrument.CreateInstance(_T("TEMScripting.Instrument.1"));
MyProjection = MyInstrument->Projection;
MyStage      = MyInstrument->Stage;
```

Here we used smart, _com_ptr_t-derived pointers that are defined in the stscript.tlh and stdscript.tli files. (Their names all end with the postfix ‘`Ptr`’.) These pointers will do the interface reference-counting for you. We also made use of the ‘property-way’ of writing, as do most scripting languages. A ‘property’ appears in the ClassView as a public member variable. In effect it can be used as such, except that for example it may be ‘read only’, i.e. setting may not be permitted.

The above code is entirely equivalent to the use of the wrapper functions that is shown in the following piece of code:

```
MyProjection = MyInstrument->GetProjection();
MyStage      = MyInstrument->GetStage();
```

In fact these functions are also called under water in the first example. The advantage of the wrapper functions is a convenient treatment of errors. If an error occurs, these functions will raise an exception (of type `_com_error`) that you can catch (see below) and that allows easy access to detailed error information. The wrapper functions also encapsulate `BSTR` (and `VARIANT`) types into `_bstr_t`-(and `_variant_t`-)objects that are easier to handle. You can of course also choose to go the hard way and use the 'raw' functions, that finally implement the properties:

```
HRESULT hr;
hr = MyInstrument->get_Projection(&MyProjection);
hr = MyInstrument->get_Stage(&MyStage);
```

For handling of errors you would then have to check the value of hr. If there is extended error information, you could query it via the `IErrorInfo`-interface. (See any book on COM on how to do this).
In this last example, it does not matter, whether you use intelligent pointers or not, while in the first two examples you have to use them, otherwise you will experience difficulties with the reference count. Your objects might be destroyed due to the fact that the wrapper functions are designed in a way that results in a hidden `Release()`-call for the interface in question. This call is used to compensate for the `AddRef()` that is connected to the use of the '='-operator for intelligent pointers. (This also implies, that you have to use '=' and not the `.Attach()` method of the smart pointer class).

24.3 Manipulate and read microscope parameters, invoke microscope actions

To manipulate microscope parameters, you can in principle use all of the three different ways of writing assignment statements that are already described above (only that for simple properties you do not retrieve interface pointers here). In the following examples, we choose the way of writing that stresses the notion of 'properties', because it is the easiest to write and because it is most consistent with the remainder of this help. An example:

```
MyProjection->MagnificationIndex = 5;
```

This statement will set the magnification index to a new value (=5).

```
long MyLong;
MyLong = MyProjection->MagnificationIndex;
```

reads the current value of the magnification index into the variable MyLong.
'Compound' properties are read and set by using the TEM scripting adapter's 'utility objects'. Thus

```
IlluminationPtr  MyIllumination;
VectorPtr        MyShift;
double           MyShiftX;
MyIllumination = MyInstrument->Illumination;
MyShift        = MyIllumination->Shift;
MyShiftX       = MyShift->X;
```

would read the full 2D-beam shift first and then store its x-component into the variable MyShiftX. Be aware that MyShift contains a copy of the actual beam shift. Thus, if you want to get the new actual

beam shift at a later moment in time, you have to ask for MyIllumination->Shift again! Setting compound parameters then works as follows:

```
VectorPtr       MyNewShift;
MyNewShift    = MyIllumination->Shift;
MyNewShift->X = 0;
MyNewShift->Y = 0;
MyIllumination->Shift = MyNewShift;
```

The codes first gets a Vector-object by reading a 2D-property, assigns new values (0,0) and then copies it back, thus setting the beam shift to zero.

24.4 Use the collections

The adapter contains several collections, such as the ‘Gauges’ and the ‘UserButtons’ collections. Collection interfaces were originally designed for Visual Basic and are very convenient to use in scripting languages. The collections that come with the adapter are of fixed size, the script cannot add items. Reading items can be done as follows:

```
GaugesPtr  MyGauges;
GaugePtr   g;
CString    sMsg;
MyGauges = MyInstrument->Vacuum->Gauges;

for (long i; i<MyGauges->Count; i++)
{
    g     = MyGauges->GetItem(i);
    CString sGauge;
    sGauge.Format(_T("%s : %d\n"), g->Name, g->Pressure);
    sMsg  = sMsg + sGauge;
}
AfxMessageBox(sMsg);
```

loops over all gauges of the vacuum system and shows for each gauge the name and the pressure measured with it on a message box.
In C++, you cannot use Item as a property (although other languages use it as such), i.e.

```
g = MyGauges->Item(i); //does not work!
```

does not work. Some scripting languages allow the use of the handy abbreviation MyGauges(i), because ‘Item’ is the default property of a collection. This is also not allowed in C++.
Probably you will often not know the index of the gauge, but its name, that you can read from the vacuum display of the microscope. GetItem therefore also accepts the gauge name as identifier, for example it is possible to write

```
g = MyGauges->GetItem(_T("P1"));
```

g now contains the information about the buffer tank pressure. Remember that the ‘gauge’ objects are utility objects, so to get the new actual values, you have to ask for a new collection again, thus to repeat:

```
MyGauges = MyInstrument->Vacuum->Gauges;
```

24.5 Receive events from the user buttons

To receive the events connected with pressing user buttons of the TEM control pads (L1,..,L3 and R1,..,R3), you have to do some work in C++.
First, the class that implements the event interface should support OLE-automation. Thus the constructor should include the following statement:

```
EnableAutomation();
```

Also you should use the following macros to your classes header file:

```
DECLARE_DISPATCH_MAP()
DECLARE_INTERFACE_MAP()
```

The class wizard will generally add these for you.
In the corresponding macros that actually define the interface and dispatch maps in your classes *.cpp-file, you have to make some adjustments, however:

a) In the dispatch map macro, you have to assign the button event Pressed to a function (called OnPressed in this example) of your CCmdTarget-derived, event- receiving class (which is called CMyButton here).

```
BEGIN_DISPATCH_MAP(CMyButton, CCmdTarget)
     DISP_FUNCTION(CMyButton, "Pressed", OnPressed, VT_EMPTY, VTS_NONE)
END_DISPATCH_MAP()
```

VT_EMPTY indicates that the function has no return parameter and VTS_NONE specifies that it does not take any data.

b) The interface that contains the user button event has to be included into the INTERFACE_MAP macro:

```
BEGIN_INTERFACE_MAP(CMyButton, CCmdTarget)
     INTERFACE_PART(CMyButton, DIID_UserButtonEvent, Dispatch)
END_INTERFACE_MAP()
```

Furthermore you have to include code that handles the connection point mechanism used to receive COM-events. This might look as follows:

```
//Assume, we alreay have a pointer pMyButton,pointing to a UserButton interface
IConnectionPointPtr MyConnectionPoint;
IConnectionPointContainerPtr
        CPContainer = pMyButton;
DWORD MyCookie;
HRESULT hr = CPContainer->FindConnectionPoint (DIID_UserButtonEvent,
&MyConnectionPoint);
if (hr == S_OK)
    //pass pointer of eventsink interface for callback by server
    hr = MyConnectionPoint->Advise(GetIDispatch( TRUE), &MyCookie);
```

And, in the end (for example in the destructor of your event-receiving class), you have to disregister with the connection point again. (For this reason you had to remember MyCookie):

```
MyConnectionPoint->UnAdvise(MyCookie);
```

In order to make the above code work, you further have to overwrite the connection point's GetIID() function to retrieve the ID of the user button event-interface:

```
IID CMyButton::GetIID ()
 {
       return DIID_UserButtonEvent;
}
```

Having done all this,

```
void CMyButton::OnPressed()
{
   // Do whatever you want
}
```

will be called, whenever the corresponding button (the button "pointed to" by pMyButton) is pressed, provided you have activated it. Activation is done by assigning a string that will then also be displayed in the TEM UI:

```
_bstr_t bstrNewAssignment(_T("any new assignment"));
pMyButton->Assignment = bstrNewAssignment;
```

24.6 Receive Events from a remote microscope server

If you want your application to connect to a remote microscope server and also receive the userbutton events, you have to add a call to the function CoInitializeSecurity, that opens your application for calls from a remote system service. Just place the following code directly after the call to AfxOleInit (see above), because you have to call CoInitializeSecurity, before COM calls it under water for you:

```
HRESULT sc = ::CoInitializeSecurity(
           NULL,
           -1,
           NULL,
           NULL,
           RPC_C_AUTHN_LEVEL_NONE,
           RPC_C_IMP_LEVEL_IMPERSONATE,
           NULL,
           EOAC_NONE,
           NULL);
if (FAILED(sc))
{
      TRACE1("CoInitializeSecurity failed!  (returned 0x%X)\nno remote events will be
received", sc);
}
```

Also, the compiler switch `_WIN32_DCOM` has to be defined. Otherwise the relevant function declarations may not be included into your project. Thus if you get a compiler error, you might have to add the following line of code, preferably in stdafx.h:

```
#define _WIN32_DCOM
```

24.7 Errors and error handling

Some of the microscopes functions may return (raise) an error. This can happen due to a physical reason, i.e. if the requested action is not possible at that moment. For example, you cannot ask the stage to perform a `GoTo()`, when it is wobbling or already moving. The high tension may not be switched on or raised under certain conditions and so forth. You may also have given parameters that

are out of range (in that case the stage for example would not move either). Generally, calls to another process (i.e. from your script to the TEM server) may fail occasionally. So be aware and do the error handling - otherwise your application might die!

If an error is raised, then some global object may be filled with detailed information about the error. (In case of TEM errors that will be done for versions >1.0). The minimal information, that is always available, consists of an HRESULT error code and a system generated standard error message. Usually you will get an error code, a textual description and some more information as described below. The easiest way to obtain this information is to use the smart pointer classes and the wrapper functions that are defined in the stdscript.tli/tlh-files and that are automatically generated when the typelib or the dll is imported into your project. You can then use the C++ try/catch mechanism to catch the errors, as shown below. Suppose you want to move the stage to a new position that is calculated from the old one -just as an example, you could invent some code here:

```
try
 {
   StagePositionPtr OldPos;
   StagePositionPtr NewPos;
   StagePtr         MyStage;
   MyStage = MyInstrument->Stage;
   OldPos  = MyStage->Position;
   CalculateNewPosition(OldPos, &NewPos) // some function
   MyStage->Goto(NewPos, axisXY);
}
catch ( _com_error E)
{
   CString sDescription;
   if (E.Description().length() > 0)
   {
     sDescription.Format(E.Description());
   }
   else if (E.ErrorMessage() != NULL)
   {
    sDescription.Format(E.ErrorMessage());
   }
   AfxMessageBox(Description);
}
```

In case of an error, a message-box will show a description of the error that ocurred. This will be either the application-delivered one, if available, or the standard message for the error code in question. Among others, the exception object supports the following handy methods:

```
HRESULT       Error( )         // returns the HRESULT
_bstr_t       Description( )   // server generated description
_bstr_t       Source( )        // name of the source
const TCHAR * ErrorMessage( )  // the standard message
```

For more information check the _com_error's member functions.

25 TEM scripting in Delphi

25.1 Introduction and package installation

The Delphi version used for the example program was the Professional edition of Delphi 4.0, but the other variants (including Desktop) will work as well. Delphi 3.0 (in all variants, including Desktop) will work as well, but when the example program is opened in Delphi 3.0 it will complain about some (in Delphi 3.0) unsupported properties. Look for the complaints from Delphi 3.0, then load the frm file as text (e.g. in Notepad) and remove the offending statements.

25.2 Events

Delphi (at least up to version 4.0) provides no direct handling of events. Instead you either have to write your own handler or (to keep it simple) use the EventSinkImport utility written by Binh Ly, which was used here (can be found on the world wide web). The Event Sink Import utility creates two files, a standard TLB (as obtained when using the Delphi Import Type Library function) and a separate Events unit. The Events unit for TEM Scripting was compiled into a package that can be installed, creating a non-visual component under ActiveX.
Important note: The declaration of the Events component under the uses clause must always precede the declaration to the TLB itself.

To install the Scripting Events component for Delphi 4.0:
- Copy the TemScripting.bpl file to the Imports folder of the Borland\Delphi 4 folder.
- Start Delphi.
- Select Component, Install Packages.
- Press the Add button and select the TemScripting.bpl file.
- In the ActiveX tab of Delphi you will now find the UserButtonEvent component.

To install the Scripting Events component for Delphi 3.0 (or Delphi 5.0) and higher:
- Copy the TemScriptingEvents.pas and TemScriptingEvents.dcr file to the Imports folder of the Borland\Delphi 3 or 5 folder.
- Start Delphi.
- Select Component, Install Component.
- Select the Into new package tab.
- For unit file name, Browse to TemScriptingEvents.pas.
- For Package file name Browse to the Imports folder and enter TemScripting.dpk.
- For Package description enter Tem Userbutton Events
- Press OK. The package will be compiled and installed. Save the package.
- In the ActiveX tab of Delphi you will now find the UserButtonEvent component.

25.3 Using the dynamic link library

To use the type library, simply copy the TemScripting_TLB.pas file to the Imports folder of the Borland\Delphi folder. Then use Project, Add to project and select the TemScripting_TLB.pas.

25.4 Example program

The program has a form with a PageControl containing a number of TabSheets, each of which contains a logical group of TEM scripting functions (Optics, Stage, Vacuum, Camera). Since some of these groups do not have very many functions, their TabSheets are rather empty, but the overall size is determined by the fullest TabSheet.

The connection with the microscope is done in FormCreate for instrument and then per tab (checked against previous connection through the nil). This method makes startup faster and eliminates unnecessary connections.

Some units, such as GetVersion are very small and can easily be combined into a program itself. Since they may be used repeatedly for more than one application, they are separated off for easy importing.

25.4.1 Large fonts

The TEM microscopes often have Windows set to Large Fonts. When this is the case, the size of all controls changes (if your program was designed with the PC set on Small Fonts) but typically not the form size (and controls will often be partially hidden).

To correct for this, you can change the settings dependent on the font setting read at start up. The font size can be determined at run time by inspecting the PixelsPerInch property. It is 96 for Small Fonts and 120 for Large Fonts.
Note: The font in a statusbar may come out bold on Large Fonts when compiled in Small Fonts. Make sure you have the ParentFont property of the statusbar set to true and it will come out normal.

25.4.2 About box and Version number

The About box for the Exampler is located in the System Menu (under the icon of the Window). For displaying the version number, these settings are read from the program file itself (to make it only necessary to update the Version Info of the program and not have to keep two settings in sync. The About Box is called through the Windows function WMSysCommand. The Version number is read through the GetVersion unit.

25.5 Create an ‘Instrument’ and get secondary microscope objects

The following piece of code shows how to get access to the projection system and the stage:

In the declaration section (under **`var`**)

```
FTem       : Instrument;
FProj      : Projection;
FStage     : Stage;
```

In a section accessed before any calls to the microscope are made (e.g. the FormCreate). In the example program the subunits (Projection, Stage, etc.) are connected per tab of the PageControl (the connections take time, so this reduces the number of connections to those necessary and reduces start-up time).

```
try
  if FTem = nil then FTem := CoInstrument.Create;
  try { no sense in trying this if the previous didn't work }
    if FProj = nil then FProj := FTem.Projection;
  except
    on E : Exception do
      Application.Messagebox(
         pchar('Error - No connection to projection - '+E.Message),
         'Error',mb_OK);
  end;
  try
    if FStage = nil then FStage := FTem.Stage;
  except
    on E : Exception do
      Application.Messagebox(
```

```
        pchar('Error - No connection to stage - '+E.Message),
        'Error',mb_OK);
  end;
except
  on E : Exception do
    Application.Messagebox(
       pchar('Error - No connection to instrument - '+E.Message),
       'Error',mb_OK);
end;
```

25.6 Manipulate and read microscope parameters, invoke microscope actions

Many of the values of the microscope can manipulated in one of two ways, using **properties** or using **functions/procedures**.
If you use Delphi, you will be quite familiar with properties. Nevertheless, here is an example:

```
FTem.MagnificationIndex := 5;
```

'Compound' properties are read and set by using the TEM scripting adapter's 'utility objects'. Thus

```
var
  FShift : Vector;
  FX     : double;
```

In the code:

```
FShift := FProj.ImageShift;
FX     := FPos.X;
```

would read the full 2D-image shift and then store the x-component in the variable FX. Be aware that `FShift` contains a copy of the image shift. If you want to get the new actual image shift at a later moment in time, you have to ask for `FProj.ImageShift` again! For setting the multidimensional properties, you have to create an instance of the utility object first (by reading it, as above), then put the right coordinate values into it and then assign it to the property. Example:

```
var
  FShift : Vector;
```

In the code:

```
FShift           := FStage.ImageShift;
FShift.X         := 1e-6; { image shift is in meters }
FShift.Y         := -1e-6;
FProj.ImageShift := FShift;
```

creates a new Vector object that is initialized with the coordinates as given and is then used to set the image shift to these values. (An exception here is the position of the stage, that is read only, because for moving the stage you have to use the `Goto_` and `MoveTo` functions. The underscore on `Goto` is added by Delphi automatically because `Goto` is a reserved word).

25.7 Use the collections

The adapter contains several collections, such as the 'Gauges' and the 'UserButtons' collections. A collection is a convenient way to store things of the same kind. The collections that come with the adapter are of fixed size, the script cannot add items. Reading items can be done in several ways:

```
var
  FTem      : Instrument;
  FVacuum   : Vacuum;
  FGauges   : Gauges;
  FGauge    : Gauge;
  indx      : integer;
  td        : double;
  OleVar1   : OleVariant;
```

In the code:

```
try
  if FTem = nil then FTem := CoInstrument.Create;
  try { no sense in trying this if the previous didn't work }
    if FVacuum = nil then FVacuum := FTem.Vacuum;
    if FGauges = nil then FGauges := FTem.Vacuum.Gauges;
    for indx := 0 to FGauges.count-1 do
    { fill labels, see example program - better }
    begin
      OleVar1 := indx;
      FGauge := FGauges.Item[OleVar1];
      Label1.caption := FGauge.Name;
      td := FGauge.Pressure;
      with Label2 do
        if td > 100 then caption := RealToStr(td,0)
         { for RealToStr see example program }
        else if td > 1 then caption := RealToStr(td,2)
        else if td > 0.001 then caption := RealToStr(td,4)
        else caption := RealToStr(td,8);
      with Label3 do
        case FGauge.Status of
          gsUndefined : caption := 'Undefined';
          gsUnderflow : caption := 'Underflow';
          gsOverflow : caption  := 'Overflow';
          gsInvalid : caption   := 'Invalid';
          gsValid : caption     := 'Valid';
        end;
    end;
  except
    on E : Exception do
      Application.Messagebox(
         pchar('Error - No connection to vacuum - '+E.Message),'Error',mb_OK);
  end;
except
  on E : Exception do
    Application.Messagebox(
        pchar('Error - No connection to instrument - '+E.Message),'Error',mb_OK);
end;
```

loops over all gauges of the vacuum system. You can access single gauges via the ‘Item’-property of the collection (thus as `MyGauges.Item[3]` for example). Probably often you will not know the index of the gauge, but you can find out its name from the vacuum display of the microscope.

The ‘Item’ property therefore also accepts the gauge name as identifier, for example it is possible to write

```
  OleVar1 := 'P1';
  FGauge := FGauges.Item[OleVar1];
  { in Delphi you cannot use the name as in VB or JScript-
    thus you cannot write FGauges[OleVar1] }
```

25.8 Receive events from the user buttons

To receive the events connected with pressing user buttons of the TEM Control pads (L1,..,L3 and R1,..,R3), you have to do the following in addition to the code for connecting to the instrument. First put a UserButtonEvent control on the form (as many as you need, so if you want L1..L3 you'll need three):

```
FTem      : Instrument;
FUserButtons : UserButtons;
F_UB_L1      : UserButton;
try
  if FUserButtons = nil then
  begin
    FUserButtons := FTem.UserButtons;
    F_UB_L1      := FUserButtons.Item[1];
    UserButtonEvent1.Connect(IUnknown (F_UB_L1) );
  end;
except
  ...
end;
```

When you double-click on the UserButtonEvent control, the following procedure will be added to the code:

```
procedure TForm1.UserButtonEvent1Pressed (Sender: TObject; newval: Integer);
begin
  { whatever }
end;
```

The procedure `UserButtonEvent1Pressed` will be invoked when User button "L1" is pressed, provided you have activated the events by using the assignment-property of `F_UB_L1`.

25.9 Receive events from a remote microscope server

If you want your application to connect to a remote microscope server and also receive the userbutton events from that remote machine, you have to change the standard COM security setting. This has to be done, before COM automatically initializes security. The hard-work approach (doing everything yourself): Immediately after the first `CoInitialize` call you must call a function named `CoInitializeSecurity` as follows:

```
OleCheck (CoInitializeSecurity (nil,
                                -1,
                                nil,
                                nil,
                                RPC_C_AUTHN_LEVEL_NONE,
                                RPC_C_IMP_LEVEL_IMPERSONATE,
                                nil,
                                EOAC_NONE,
                                nil));
```

where

```
RPC_C_AUTHN_LEVEL_NONE = 1
RPC_C_IMP_LEVEL_IMPERSONATE = 3
EOAC_NONE = 0
```

When you are using the functionality from Binh Ly (comlib) you simply call

```
InitializeComSecurity (alNone,ilImpersonate);
```

In the examples, this call is made immediately in the FormCreate procedure.

25.10 Errors and error handling

Some of the microscopes functions may return (raise) an error. This can happen due to a physical reason, i.e. if the requested action is not possible at that moment. For example, you cannot ask the stage to perform a `GoTo_`, when it is wobbling or already moving. The high tension may not be switched on or raised under certain conditions, and so forth. You may also have given parameters that are out of range (in that case the stage for example would not move either).
Generally, calls to another process (from your script to the TEM server) may fail occasionally. So be aware and do the error handling, typically by a **`try except end;`**

If an error is raised, then `Exception`**`.Message`** will be filled with detailed information about the error, if available. You can use this information for your error handling. See the error handling in the code examples above.

26 TEM scripting in JScript

This chapter explains how to use the TEM Scripting component from JavaScript / JScript client code. JScript code can be run in a browser as part of an HTML page/s (as illustrated in the examples). This approach allows a graphical user interface to be constructed relatively easily using HTML tags.

An alternative approach would be to run a script using the Windows Scripting Host.

26.1 Create the Instrument Object and get a secondary Microscope Object

The following code demonstrates how to get access to the Projection System and the Stage. In this example, the code is included into HTML code. As an alternative, the JavaScript code could be located in a separate source file by using the SRC attribute of the HTML <SCRIPT> tag.

Regarding the style of the code snippets below, in general global variables (that is, variables that 'live' beyond just one function) are given a prefix of 'g_'.

```
<HTML>
...
<BODY ONLOAD="Connect()">
...
<SCRIPT LANGUAGE="JavaScript">
<!--// Variables used throughout the script
var g_myTem;
var g_myProjection;
var g_myStage;
...

function Connect()
{
   g_myTem        = new ActiveXObject("TEMScripting.Instrument");
   g_myProjection = g_myTem.Projection;
   g_myStage      = g_myTem.Stage;  ...}
...
-->
</SCRIPT>
</BODY>
</HTML>
```

Notice that the Connect method is called when the HTML page is loaded.

26.2 Using Microscope Parameters and invoking actions

Within JScript the desired actions are often carried out using properties. For example, the following code sets the magnification index to 5.

```
g_myProjection.MagnificationIndex = 5;
```

And the following copies the current magnification index into the variable 'myMagIndex'.

```
var myMagIndex;
myMagIndex = g_myProjection.MagnificationIndex;
```

Compound properties are read and set by using the TEM scripting 'utility objects'. Consider the following.

```
var myIllumination;
var myShift;
var myShiftX;

myIllumination = g_myTem.Illumination;
myShift        = myIllumination.Shift;
myShiftX       = myShift.X;
```

This reads the full 2D Beam Shift and then stores its x component in the variable ‘myShiftX’. Note that ‘myShiftX’ contains a copy of the actual Beam Shift. Thus, it is not necessarily the Beam Shift a few moments later – for that, the ‘Illumination must be retrieved again.

Setting compound parameters can be done as follows. First a ‘Vector’ type is defined by retrieving a ‘Vector’ constructor function object, then a new ‘Vector’ object is constructed and initialized to (0,0). This newly created object is then used to set the Beam Shift to 0.

```
var myNewShift;
var myVector;

myVector   = g_myTem.Vector;
myNewShift = new myVector(0,0);

g_myIllumination.Shift = myNewShift;
```

The following code is an alternative approach that same the same effect, basically the Vector object is being retrieved from Illumination object.

```
myNewShift           = g_myIllumination.Shift;
myNewShift.X         = 0;
myNewShift.Y         = 0;

myIllumination.Shift = g_myNewShift;
```

The above code is specific for JScript and works in the same fashion for the ‘StagePosition’ object.

26.3 Use the collections

TEM Scripting contains several useful collections that make some programming aspects easier. For example there are collections for Vacuum ‘Gauges’ and the ‘User Buttons’. A collection is a convenient way to store things. The collections that are of fixed size, e.g. items cannot be added.

Reading the items in a collection is typically done as follows.

```
var myGauges;
var g;
var sMsg = "";

myGauges = g_myTem.Vacuum.Gauges;

for (i=0; i<myGauges.Count; i++)
{
   g = myGauges(i);
   sMsg = sMsg + g.Name + ": " + g.Pressure;
}
alert(sMsg);
```

The above code loops over all gauges of the Vacuum system and then shows a message box containing the names of all gauges and the pressures measured by them.

A single gauge can be accessed via the ‘Item’ property of the collection (e.g. ‘myGauges.Item(3)’ for example). However, since ‘Item’ is the default property, you can use the array syntax as demonstrated in the above example (i.e. ‘myGauges(i)’).

Typically a Gauge will be indexed by name rather than index, the name being the same name that is seen on the vacuum display of the Microscope software (peoui). The ‘Item’ property therefore also accepts the gauge name as identifier, for example it is thus possible to write:

```
g = myGauges("P1");
```

The variable ‘g’ now contains the information about the Buffer Tank pressure. Remember that the ‘Gauge’ objects are utility objects, so to get the new actual values you have to ask for a new collection again.

```
myGauges = g_myTem.Vacuum.Gauges;
```

26.4 Receive events from the user buttons

The ‘Connect()’ function above (see section Create the Instrument Object and get a secondary Microscope Object) is an example of how to define event handlers that react to events originating from HTML elements.

To define event handlers for TEM user button-events is a bit more involved. The following code shows how to do it (for the ‘L1’ button).

```
...
var buttonL1;
...

function Connect()
{
   ...
   buttonL1 = g_myTem.UserButtons("L1");
   ...

   DefineEventHandlers();
   ...
   ...
}

function DefineEventHandlers()
{
   function buttonL1::Pressed()
   {
       //Do what you want to do here
       ...
   }
}
```

The above code is necessary to convince the Script Interpreter that ‘buttonL1’ is indeed a ‘UserButton’ object and that the subsequent definition of ‘buttonL1::Pressed()’ makes sense and implements the ‘UserButton’ event.

The main point is, that after assigning a ‘UserButton’ object to MyL1, the function ‘DefineEventHandlers()’ is called that contains the definition of the event handler. Simply defining ‘buttonL1::Pressed()’ inside or outside the ‘Connect()’ function would yield an error:
‘buttonL1 not an object’ or ‘buttonL1 undefined’).

‘buttonL1::Pressed()’ will be called whenever the corresponding TEM user button L1 is pressed, provided you have activated it for the script. This activation is done by assigning a string that will then also be displayed in the TEM user interface (peoui):

```
MyL1.Assignment = "my new assignment";
```

26.5 Errors and error handling

Some of the Microscope functions when called may return (raise) an error.

- This can happen due to a physical reason, i.e. if the requested action is not possible at that moment. For example, you cannot ask the Stage to perform a ‘GoTo()’, when it is wobbling or already moving. Or the High Tension may not be switched on or its value changed in certain situations.
- The function being called might also have been passed parameters that are out of range (in the case of the Stage this will result in it simply not moving).
- Also calls from client programs to the Microscope Server process might fail due to some software communication problem.

It is best to be aware of these possible problems and, depending on the objectives, program accordingly.

If an error occurs an ‘Error’ object will be generated. Technically this is referred to as ‘an exception being thrown’. If this occurs then any client code (JavaScript in this case) needs to determine if such exceptions should be caught, and if so what should be done.

The error contains an error code (a number) and a textual description. The JScript ‘try/catch’ mechanism can be used to catch such errors. The following example helps illustrate how.

Suppose the HTML page contains a form with a button named ‘cmdGoto’, and a text area named ‘TextErrors’. Pressing the button triggers movement of the stage to a new position that is calculated from the old one:

```
<FORM ID="theForm">
...
<INPUT TYPE=BUTTON NAME="cmdGoto" VALUE="Go" ONCLICK="OnGoto()">
...
<TEXTAREA NAME="TextErrors" COLS=.. ROWS=.. READONLY>
</TEXTAREA>
...
</FORM>
...
...

function OnGoto()
{
   try
   {
```

```
        var newPos;
        var oldPos;
        var myStage;

        myStage = g_myTem.Stage;
        oldPos  = mystage.Position;

        newPos  = new g_myTem.StagePosition(0,0,0,0,0);
        CalculateNewPosition(oldPos, newPos) // some function
        myStage.Goto(newPos, axisXY);
     }
     catch (Err)
     {
        var n;
        var sn;
        n  = new Number(Err.number);
        sn = (n + Math.pow (2,32)).toString(16); // hex number as string
        with (theForm.TextErrors)
        {
           value = value + sn + "\n";
           value = value + Err.description + "\n";
        }
     }
  }
```

If an error occurs when the code in the ‘try block’ is executed, the program will automatically enter the ‘catch block’. Here the error code (in hexadecimal format) and the error description will be written to the text area of the HTML page.

Note that some errors come from the TEM Scripting component itself and some come from the underlying software infrastructure (e.g. COM layers). As an example of the latter, should creation of the ‘Instrument’ object fail then the error code and message returned will be a general error message and not something specific to the TEM Scripting component itself. Unfortunately in some cases this general error message hides the more useful underlying message. For instance, JScript will miss the useful description ‘TEM scripting: Protection key missing!’ indicating that the Customer Dongle is missing...

26.6 Image Acquisition

This section explains how to acquire image data in JScript client code. This is done through a series of worked examples.

26.6.1 Handling Array Data

The TEM Scripting component uses Microsoft COM Technology to expose its interfaces and methods to clients. The primary reason for this is to allow TEM Scripting to be accessed from a variety of different languages (e.g. VBScript, JScript, Visual Basic, C++, Delphi, Python, C#).

However, one of the key differences between VBScript / JScript clients and the others is the manner in which array data is handled. ***This problem has been resolved in Tecnai 4.3 (or later) and Titan 1.4 (or later).*** If you require a fix for an earlier version please contact FEI (Dave.Karetnyk@fei.com) to determine a suitable workaround.

The TEM Scripting calls that are affected by this are outlined in the following sub-sections.

26.6.1.1 Get Acquired Image as VARIANT

For JavaScript / VBScript clients the ‘AsSafeArray’ property on the ‘AcqImage’ interface cannot be used. Instead the property ‘AsVariant’ should be used.

The following JavaScript code snippet shows how the property might be used. It retrieves the image data as a two dimensional array then displays information about the array along with the first few pixel values in a text box on the HTML page (see example in section CCD Acquisition Example below for more details).

```
// get the pixel data, use 'AsVariant' which returns the data as an SAFEARRAY of
// VARIANTS packed into a VARIANT.
g_imageData = g_acquiredImage.AsVariant;

with (_textAreaMessagesAndErrors) {
   value += "image array, # of dimensions is " + g_imageData.dimensions() + "\n";
   value += "first dimension, lower bound is " + g_imageData.lbound(1) + "\n";
   value += "first dimension, upper bound is " + g_imageData.ubound(1) + "\n";
   value += " second dimension, lower bound is " + g_imageData.lbound(2) + "\n";
   value += "second dimension, upper bound is " + g_imageData.ubound(2) + "\n";

   value += "first few pixel values are: ";
   for (var i = g_imageData.lbound(1); i <= 10; i++) {
      value += g_imageData.getItem(i, 0) + " ";
   }
   value += "\n\n";
}
```

The pixel data is returned as a two dimensional array with the first element starting a 0. For instance a 512x512 image will be returned as a two dimensional array with 262144 elements, so ***no*** header is returned with the image data. And the image rows and columns will be indexed from 0 to 511. The elements are of type 32 bit signed integer.

26.6.1.2 Get Camera Binnings as VARIANT

For JavaScript / VBScript clients the ‘Binnings’ property on the ‘CCDCameraInfo’ interface cannot be used. Instead the property ‘BinningsAsVariant’ should be used.

The following JavaScript code snippet shows how the property might be used. It retrieves the binnings for the current camera then displays them in a text box (see example in section CCD Acquisition Example below for more details).

```
// fetch and display the supported binnings, use 'BinningsAsVariant' which returns
// the data as an SAFEARRAY of VARIANTS packed into a VARIANT.
binnings = g_camera.Info.BinningsAsVariant;
with (_textAreaMessagesAndErrors) {
   value += "binnings array, # dimensions is " + binnings.dimensions() + "\n";
   value += "binnings, lower bound is " + binnings.lbound(1) + "\n";
   value += "binnings, upper bound is " + binnings.ubound(1) + "\n";
   value += "available binnings are ";
   for (var i = binnings.lbound(1); i <= binnings.ubound(1); i++) {
      value += binnings.getItem(i) + " ";
   }
value += "\n\n";
}
```

In principle the array that is returned can be multi-dimensional (see section Get Acquired Image as VARIANT above dealing with image data). But in this case the array will have only one dimension with the first element starting at 0. For example, a camera supporting binnings 1, 2, 4, and 8 will be returned as a one dimensional array with 4 elements, the first element at index 0 and the last at index 3.The elements are of type 32 bit signed integer.

26.6.1.3 Get CCD Shutter Modes as VARIANT

For JavaScript / VBScript clients the 'ShutterModes' property on the 'CCDCameraInfo' interface cannot be used. Instead the property 'ShutterModesAsVariant' should be used.

The property for returning the CCD shutter modes is pretty much identical to that used for the CCD binnings (see previous section Get Camera Binnings as VARIANT). So the modes will be returned as a one-dimensional array of 32-bit signed integers with the first element starting at 0. For a full example, see in section CCD Acquisition Example below.

26.6.1.4 Get STEM Binnings as VARIANT

For JavaScript / VBScript clients the 'Binnings' property on the 'STEMDetectorInfo' interface cannot be used. Instead the property 'BinningsAsVariant' should be used.

The property for returning the STEM binnings is pretty much identical to that used for the CCD binnings (see previous section Get Camera Binnings as VARIANT). So the binnings will be returned as a one-dimensional array of 32-bit signed integers with the first element starting at 0. For a full example, see in section CCD Acquisition Example below.

26.6.1.5 Get Acquired Image as File

The 'AsFile' method of the 'AcqImage' interface allows clients to request that an acquired image be saved in a specified format and in a specified location. TIFF (16-bit), JPG (8-bit), and PNG (16-bit) are currently supported. For a full example, see section CCD Acquisition and Display Example below.

This method can be used by any type of client, Scripting or otherwise. But it is particularly relevant for JScript / VBScript clients as an easier means of saving pixel data for either processing off-line or for Browser display.

The following JavaScript code snippet shows how the property might be used. For illustration is shows saving a previously acquired image in the three supported formats.

```
// file formats for saving images
var AcqImageFileFormat_TIFF = 0;
var AcqImageFileFormat_JPG = 1;
var AcqImageFileFormat_PNG = 2;

g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.tif", AcqImageFileFormat_TIFF);
g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.jpg", AcqImageFileFormat_JPG);
g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.png", AcqImageFileFormat_PNG);
```

Additionally, the following formats can in principle be used. Subject to proper testing and depending on feedback from customers these formats may be 'officially added' at a later date.

```
// these formats still under discussion/development, magic cookie of 123 needed
// to experiment with them
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.raw", 123);
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.ser", 123);
g_acquiredImage.AsFile("C:\\Temp\\Scripting\\SaveStemImage.mrc", 123);
```

The ***raw format*** is an internal format used by FEI; its most likely use is when trying to diagnose problems with images since it allows inspection of the image data 'exactly' as it is returned from the Microscope Server. The ***ser format*** is the officially published TIA format. And the ***mrc format*** is the

format used by several microscopy related packages; it does not currently include the FEI extended header since only a single image is returned from the scripting component. Any feedback on the merits or otherwise of these formats is welcome (please mail to Dave.Karetnyk@fei.com).

26.6.1.6 Normalizing the Acquired Image

The 'AsFile' method of the 'AcqImage' interface can also be used to 'normalize' acquired image data. For example, say an image is acquired which has minimum and maximum pixel values of 134 and 213 respectively. If this is saved as a 16-bit PNG file and then displayed in the Browser it will typically be seen as a completely black image.

For display purposes the image can be 'stretched out', e.g. using the previous figures the minimum intensity value (134) will be mapped to 0 and the maximum intensity value (213) mapped to 65535 when saving as a normalized 16 bit image. This will allow the image to be easily displayed in a Browser without any processing in the client code (hint – such processing is JavaScript / VBScript is not straightforward).

The normalize variation of 'AsFile' is illustrated by the following code fragment where the current acquired image is stored first as a PNG file exactly as it was acquired from the Microscope Server and then secondly it is stored as a normalized image.

```
// file formats for saving images
var AcqImageFileFormat_TIFF = 0;
var AcqImageFileFormat_JPG = 1;
var AcqImageFileFormat_PNG = 2;

// save the image as PNG without any modification
g_acquiredImage.AsFile("C:\\Temp\\SaveCcdImage.png", AcqImageFileFormat_PNG);

// save the image as a normalized PNG file by passing -1 for the optional third parameter
g_acquiredImage.AsFile("C:\\Temp\\SaveNormalizedCcdImage.png", AcqImageFileFormat_PNG, -1);
```

So the image will only be saved in normalized format if the optional third parameter is set to -1.

However, the technique should be viewed as purely a convenience. In principle such image data adjustment should be the responsibility of the client code.

And lastly, the normalize trick is of course available to all clients, not just Scripting clients. For a full example, see section CCD Acquisition and Display Example below.

26.6.2 Example Programs

26.6.2.1 CCD Acquisition Example

File 'CCDAcqJScript.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

Setup / preconditions:

- To run the program create the directory 'C:\Temp\Scripting'.
- Make sure all software including TIA is running and that a CCD image can be acquired in TIA manually.

The example illustrates how to:

- Do some JScript basics:

  - Embedding JavaScript and HTML code in one file.
  - Writing functions that respond to buttons on the HTML page being pressed.
  - Writing code to catch errors and display them on the user interface.
- Connect to the Microscope Scripting Component.
- Fetch the current camera and displays its characteristics, e.g. name, width, height, and available binning.
  - Fetching the available binnings and acquisition modes uses two of the properties added specifically for JScript / VBScript clients (properties 'BinningsAsVariant' and 'ShutterModesAsVariant' respectively).
- Acquire an image from the selected camera using specified acquisition parameters.
  - The raw pixel data is acquired via the property 'AsVariant'.
- Save the raw pixel data to text file. *Note – depending on the image size this might be very slow!*
- Use the property 'AsFile' to save the acquired image in three different formats:
  - JPG, 8 bit, single-channel.
  - PNG, 16 bit, single-channel.
  - TIFF, 16 bit, single-channel.

26.6.2.2 STEM Acquisition Example

File 'STEMAcqJScript.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

Setup / preconditions:
- To run the program create the directory 'C:\Temp\Scripting'.
- Make sure all software including TIA is running and that a STEM image can be acquired in TIA manually.

The example illustrates how to:
- Do some JScript basics:
  - Embedding JavaScript and HTML code in one file.
  - Writing functions that respond to buttons on the HTML page being pressed.
  - Writing code to catch errors and display them on the user interface.
- Connect to the Microscope Scripting Component.
- Fetch the first available STEM detector and display its characteristics, e.g. name, width, height, and available binning.
  - Fetching the available binnings uses the property specifically added for JScript / VBScript clients (properties 'BinningsAsVariant').
- Acquire an image from the selected detector using specified acquisition parameters.
  - The raw pixel data is acquired via the property 'AsVariant'.
- Save the raw pixel data to text file. *Note – depending on the image size this might be very slow!*
- Use the property 'AsFile' to save the acquired image in three different formats:
  - JPG, 8 bit, single-channel.
  - PNG, 16 bit, single-channel.
  - TIFF, 16 bit, single-channel.

26.6.2.3 CCD Acquisition and Display Example

File 'AcquireCCDSingleImage.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

Setup / preconditions:
- Make sure all software including TIA is running and that a CCD image on the camera to be used can be acquired in TIA manually.

- If running the script on the Tecnai/Titan simulation software, the bias/gain reference setting will need to be changed. It is set to 'AcqImageCorrection_Default' which is what would typically be desired on the Microscope. To run in simulation, change the value to 'AcqImageCorrection_Unprocessed' in the method 'SetupCameraAcqParams' in file 'shared_code.js'.

The example illustrates how to:
- Do some JScript basics:
  - Embedding JavaScript and HTML code in one file.
  - Organizing some common shared JScript code into a separate file and calling it ('shared_code.js').
  - Respond to user specified input (e.g. binning and image size) and convert that input to the appropriate TEM Scripting parameter.
- Acquire an image from the selected camera using specified acquisition parameters.
  - The raw pixel data is acquired via the property 'AsVariant'.
- Use the property 'AsFile' to save the acquired image as a file in the format specified by the user.
- Save a normalized version of the image to file and display that on the HTML user interface.

26.6.2.4 Multiple CCD Acquisition and Display Example

File 'AcquireCCDMultipleImages.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

Setup / preconditions:
- Make sure all software including TIA is running and that a CCD image on the camera to be used can be acquired in TIA manually.
- If running the script on the Tecnai/Titan simulation software, the bias/gain reference setting will need to be changed. It is set to 'AcqImageCorrection_Default' which is what would typically be desired on the Microscope. To run in simulation, change the value to 'AcqImageCorrection_Unprocessed' in the method 'SetupCameraAcqParams' in file 'shared_code.js'.

The example builds on the previous example but allows for a sequence of CCD images to be acquired. This sequencing is achieved by using Windows timers.

26.7 Example Programs

Installed under 'C:\Tecnai\Scripting\JScript' the following examples can be found.

26.7.1 Get/Set Image Beam Shift

File 'jscriptdebug2d.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

This example shows how to get and set the Image Beam Shift.

26.7.2 Executing JScript code fragments

File 'jscripter.htm' installed in 'C:\Tecnai\Scripting\JScript' or 'C:\Titan\Scripting\JScript'.

This example allows fragments of JScript code to be entered into the text box of an HTML page then executed.

26.7.3 Working with the Hand Panels

File ‘jscriptexample.htm’ installed in ‘C:\Tecnai\Scripting\JScript’ or ‘C:\Titan\Scripting\JScript’.

This is a small script that demonstrates how to work with the Hand Panels.

26.7.4 Reading the Magnification

File ‘magntest.htm’ installed in ‘C:\Tecnai\Scripting\JScript’ or ‘C:\Titan\Scripting\JScript’.

Uses a Windows timer to regularly read and display the current Microscope magnification.

26.7.5 Controlling the Stage

File ‘stagetest.htm’ installed in ‘C:\Tecnai\Scripting\JScript’ or ‘C:\Titan\Scripting\JScript’.

A small example that shows how to control the stage.

27 TEM scripting in VBScript

This chapter explains how to run a script in an HTML-page, as is also demonstrated in the example program. This way you can easily build a graphical user interface, using HTML input tags. An alternative approach would be to run a script using the windows scripting host.

27.1 Create an 'Instrument' and get the secondary microscope object that you need

The following code demonstrates how to get access to the projection system and the stage. In this example, the code is included into the HTML page (you could also specify the source-file using the SRC attribute of the <SCRIPT>-tag) and will be triggered, when the body of the HTML page is loaded:

```
<HTML>
 ...
 <BODY ONLOAD="Connect()">
 ...
 <SCRIPT LANGUAGE="VBScript">
<!--Option Explicit// Variables used throughout the script
Dim MyTem
Dim MyProjection
Dim MyStage
 ...
Sub Connect()
  MyTem            = CreateObject("TEMScripting.Instrument")
  Set MyProjection = MyTem.Projection
  Set MyStage      = MyTem.Stage
  ...
End Sub
 ...
-->
</SCRIPT>
</BODY>
</HTML>
```

27.2 Manipulate and read microscope parameters, invoke microscope actions

If you use VBScript, you will be quite familiar with properties. Nevertheless, here is an example:

```
MyProjection.MagnificationIndex = 5
```

This statement will set the magnification index to a new value (=5).

```
Dim MyLong;MyLong = MyProjection.MagnificationIndex
```

reads the current value of the magnification index into the variable MyLong.
Compound properties are read and set by using the TEM scripting adapters 'utility objects'. Thus

```
Dim MyIlluminationDim MyShiftXDim MyShift
Set MyIllumination = MyTem.Illumination
Set MyShift        = MyIllumination.Shift
MyShiftX           = MyShift.X
```

would read the full 2D-beam shift first and then store its x-component in the variable `MyShiftX`. Be aware that `MyShift` contains a copy of the actual beam shift, although it seems to be passed as a reference (using "Set"). The reason is, that the copy is made by the adapter and is then passed. Thus, if

you want to get the new actual beam shift at a later moment in time, you have to ask for `MyIllumination.Shift` again! Setting compound parameters then works as follows:

```
Dim MyNewShift as Tecnai.Vector
Set MyNewShift       = MyIllumination.Shift
MyNewShift.X         = 0
MyNewShift.Y         = 0
MyIllumination.Shift = MyNewShift
```

The code first gets a ‘Vector’ object by reading a 2D-property, then assigns new values (0,0) and then copies it back, thus setting the beam shift to zero (you cannot use "Set" here).

27.3 Use the collections

The adapter contains several collections, such as the ‘Gauges’ and the ‘UserButtons’ collections. A collection is a convenient way to store things of the same kind. The collections that come with the adapter are of fixed size, the script cannot add items. Reading items can be done in several ways:

```
Dim MyGauges
Dim g
Dim sMsgs
Msg = ""Set MyGauges = MyTem.Vacuum.Gauges
For Each g In MyGauges
  sMsg = sMsg & g.Name & ": " & g.Pressure & vbcrlf
next
Msgbox sMsg
```

loops over all gauges of the vacuum system and then shows a message box containing the names of all gauges and the pressures measured with them. You can access single gauges via the ‘Item’-property of the collection (thus as `MyGauges.Item(3)` for example), but since ‘Item’ is the default property, you can more easily use the array-type syntax

```
Set g = MyGauges(3)
```

Probably often you will not know the index of the gauge, but its name, that you can read from the vacuum display of the microscope. The ‘Item’ property therefore also accepts the gauge name as identifier, for example it is possible to write

```
Set g = MyGauges("P1")
```

g now contains the information about the buffer tank pressure. Remember that the ‘Gauge’ objects are utility objects, so to get the new actual values, you have to ask for a new collection again:

```
Set MyGauges = MyTem.Vacuum.Gauges
```

does the job.

27.4 Receive events from the user buttons

Since we did not succeed in recieving events in VBScript from objects that cannot be created directly, an additional DLL is delivered for VBScript. Its name is scriptevents.dll. It only delivers one object, the ‘TEMScriptingEvents’-object. Attention: Creating this object using

```
CreateObject("TEMScripting.TEMScriptingEvents")
```

does not work! This function does not connect to the events. You have to use the <object>-tag in your HTML-code:

```
<object id="MyTemEvents"
   classid="clsid:5E896A91-A0BF-11d3-A688-00C04F9D480A" >
</object>
```

The name of the id-parameter can be choosen arbitrarily and is the name under which you later address the 'TEMScriptingEvents'-object in your script. The 'TEMScriptingEvents'-object does not have any properties or methods, it only supplies one event: Pressed(ButtonName). This function has to be implemented in your script and will be called, whenever one of the User Buttons on the TEM handpanels is pressed. In contrast to the 'normal' way of receiving the events via the 'UserButton'-objects of TEM scripting, the VBScript-client will have to check whether the event is actually meant for him. He cannot make use of the intelligence that is built into the 'UserButton'-objects of TEM scripting. The following example shows how to do it:

```
Dim MyButtons 'need a "global" UserButtons-collection
...
'initialize collection in your "connect()" function
Set MyButtons = MyTem.UserButtons
...
'implement the event:
Sub MyTemEvents_Pressed(ButtonName)
   'In VBScript you have to check whether the event is meant for you
   Dim Button
   Set Button = MyButtons("" & Name)
       'Using "" & makes clear that we deal with a string
   if ( (Button.Label <> "")
       and (Button.Assignment = Button.Label) ) then
         ' we indeed control the button
         ....
   end if
End Sub
```

Remember: The 'UserButton'-Label is the Label of the Button as it is shown in the TEM user interface. If your script did an assignment to a button, then assignment and label coincide. Your assignment can (hopefully temporarily) be overwritten by another application (for example by the alignment procedures). You would then probably not want to react on the events.

27.5 Errors and error handling

Some of the microscopes functions may return (raise) an error. This can happen due to a physical reason, i.e. if the requested action is not possible at that moment. For example, you cannot ask the stage to perform a GoTo(), when it is wobbling or already moving. The high tension may not be switched on or raised under certain conditions and so forth. You may also have given parameters that are out of range (in that case the stage for example would not move either). Generally, calls to another process (ie from your script to the TEM server) may fail occasionally. So be aware and do the error handling - otherwise your application might die!

If an error is raised, then a global Error object (Err) will be filled with detailed information about the error, if available. You can use this information for your error handling. Usually you will get an error code (a number), a textual description and maybe a text string with the name of the source that raised the error. If you want to do the error handling, you have to insert the following statement into the code:

```
On Error Resume Next
```

This statement makes VBScript ignoring errors and is valid for the code to follow until the end of the function or subroutine. You then got the chance to ask the error object for its content.

An example:
Suppose your HTML page contains a form with a button named ‘cmdGoto’, and a text area named ‘TextErrors’. Pressing the button triggers movement of the stage to a new position that is calculated from the old one -just as an example, you could invent some code here:

```
<FORM ID="theForm">
 ...
<INPUT TYPE=BUTTON NAME="cmdGoto" VALUE="Go" ONCLICK="OnGoto()">
 ...
<TEXTAREA NAME="TextErrors" COLS=.. ROWS=.. READONLY>
</TEXTAREA>
 ...
</FORM>
 ...
 ...
Sub OnGoto()
  Dim NewPos
  Dim OldPos
  Dim MyStage
  On Error Resume Next
  Set MyStage = MyTem.Stage
  CheckError
  Set OldPos  = MyStage.Position
  CheckError
  CalculateNewPosition(OldPos, NewPos) // some function
  MyStage.Goto(NewPos, axisXY)
  CheckError
End Sub

Sub CheckError()
  If (Err.Number <> 0) Then
       With theForm.TextErrors
          .Value = .Value + "Error:" &
                   Hex(Error.Number) & vbCrLf _
                   & "Source: " & Error.Source & _
                   vbCrLf & Error.Description & vbCrLf
       End With
  End If
  Err.Clear 'Clears the error object
End Sub
```

In case of an error, the error code (in hexadezimal format), its source (if available) and the error description will be added to the text in the text area. If we would not clear the error object by using `Err.Clear`, we would handle the same error with the next check again.
In some cases, for example when the creation of the ‘Instrument’ object fails, VBcript insists on its own error code and error message, overwriting any description given by TEM scripting.

Note: For example you will miss the description "Tecnai scripting: Protection key missing!", that indicates that the protection dongle is missing.

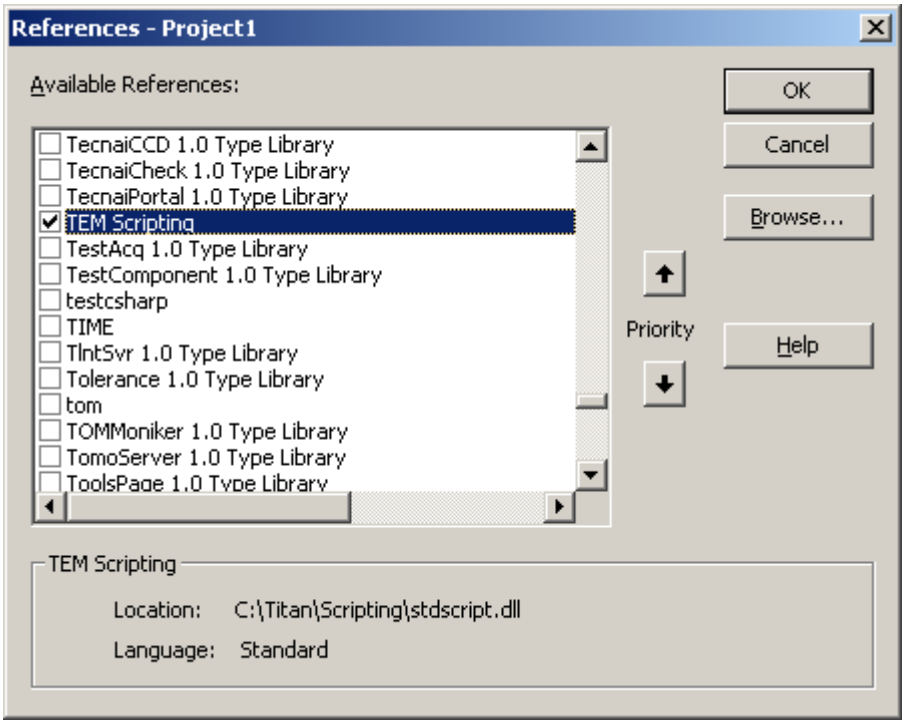
28 TEM scripting in Visual Basic

28.1 Import the dynamic link library

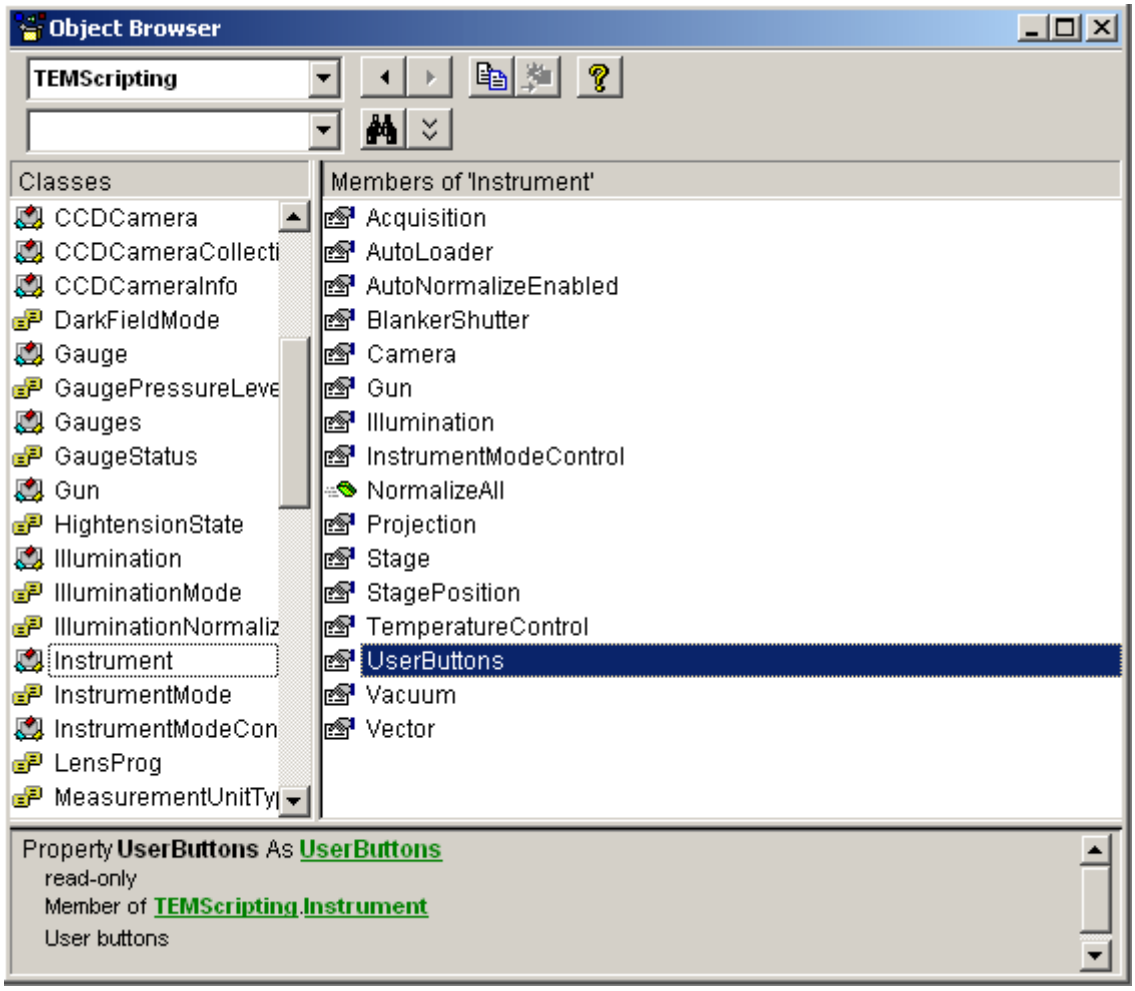
To get the support of Visual Basics IntelliSense mechanism and to be able to declare variables of the types that the adapter delivers (ie to use hard typing and early binding), you have to import the adapters dynamic link library into your project. This is easily achieved.

- create a VB Project if that is not already done
- reference the adapter :

Click on the menu item Project – References. The following window will pop up (the actual contents will vary):



Find the TEMScripting and check it. Now that the adapter is included into the project, you will see its objects, their properties and their methods appear in the object browser (click View – Object Browser or press F2):



For every item you will find a small description in the lower gray part of this window. Context sensitive help (try pressing F1) is also available, if you are working with VB6.0

28.2 Create an 'Instrument' and get the secondary microscope object that you need

The following piece of code shows how to get access to the projection system and the stage:

```
Dim MyTem        as TEMScripting.Instrument
Dim MyProjection as TEMScripting.Projection
Dim MyStage      as TEMScripting.Stage
Set MyTem     = new TEMScripting.Instrument
Set MyProjection = MyTem.Projection
Set MyStage      = MyTem.Stage
```

28.3 Manipulate and read microscope parameters, invoke microscope actions

If you use VB, you will be quite familiar with properties. Nevertheless, here is an example:

```
MyProjection.MagnificationIndex = 5
```

This statement will set the magnification index to a new value (=5).

```
Dim MyLong as Long
MyLong = MyProjection.MagnificationIndex
```

reads the current value of the magnification index into the variable MyLong.
‘Compound’ properties are read and set by using The TEM scripting adapters ‘utility objects’. Thus

```
Dim MyIl     as TEMScripting.Illumination
Dim MyShiftX as Double
Dim MyShift  as TEMScripting.Vector
Set MyIl    = MyTem.Illumination
Set MyShift = MyIL.Shift
MyShiftX    = MyShift.X
```

would read the full 2D-beam shift first and then store its x-component into the variable `MyShiftX`. Be aware that `MyShift` contains a copy of the actual beam shift, although it seems to be passed as a reference (using "Set"). The reason is, that the copy is made by the adapter and is then passed. Thus, if you want to get the new actual beam shift at a later moment in time, you have to ask for `MyIl.Shift` again! Setting compound parameters then works as follows:

```
Dim MyNewShift as TEMScripting.Vector
Set MyNewShift = MyIl.Shift
MyNewShift.X   = 0
MyNewShift.Y   = 0
MyIl.Shift     = MyNewShift
```

The codes first gets a ‘Vector’ object by reading a 2D-property, then assigns new values (0,0) and then copies it back, thus setting the beam shift to zero (you cannot use "`Set`" here).

28.4 Use the collections

The adapter contains several collections, such as the ‘Gauges’ and the ‘UserButtons’ collections. A collection is a convenient way to store things of the same kind. The collections that come with the adapter are of fixed size, the script cannot add items. Reading items can be done in several ways:

```
Dim MyTem as TEMScripting.Instrument
Dim MyGauges as TEMScripting.Gauges
Dim g        as TEMScripting.Gauge
Open "TESTFILE" For Output As #1
Set MyGauges = MyTem.Vacuum.Gauges
For Each g In MyGauges
    Print #1, g.name & ": " & g.Pressure
next
Close #1
```

loops over all gauges of the vacuum system and prints their names and the pressures measured with them to a file named "TESTFILE". You can access single gauges via the ‘Item’-property of the collection (thus as `MyGauges.Item(3)` for example), but since ‘Item’ is the default property, you can more easily use the array-type syntax

```
Set g = MyGauges(3)
```

Probably often you will not know the index of the gauge, but its name, that you can read from the vacuum display of the microscope. The ‘Item’ property therefore also accepts the gauge name as identifier, for example it is possible to write

```
Set g = MyGauges("P1")
```

g now contains the information about the buffer tank pressure. Remember that the ‘Gauge’ objects are utility objects, so to get the new actual values, you have to ask for a new collection again:

```
Set MyGauges = MyTem.Vacuum.Gauges
```

does the job.

28.5 Receive events from the user buttons

To receive the events connected with pressing user buttons of the TEM control pads (L1,..,L3 and R1,..,R3), you have to do the following (the example uses the user button "L1"):

```
Private WithEvents MyButton as TEMScripting.UserButton
Set MyButton = MyTem.UserButtons("L1")
...
'Activate events'and let a label be displayed in the TEM UI
MyButton.Assignment = "my new label"
...
Private Sub MyButton_Pressed()
     'Do what you want here
End Sub...
```

The procedure MyButton_Pressed will be invoked whenever the user button "L1" is pressed.

28.6 Receive events from a remote microscope server

If you want your application to connect to a remote microscope server and also receive the userbutton events from that remote machine, you have to add a call to the function CoInitializeSecurity that opens your application for calls from a remote system service. You have to do that, before VisualBasic does this for you under water, because this function call succeeds only once per application. What you have to do is to add a module to your application (Menue--Project--AddModule). In this module, add the following code:

```
Option Explicit
'see example VB application for more constants
Private Const RPC_C_AUTHN_NONE As Long = 0
Private Const RPC_C_AUTHN_LEVEL_NONE As Long = 1
Private Const RPC_C_IMP_LEVEL_IMPERSONATE As Long = 3
Private Const EOAC_NONE As Long = &H0
 ' Import function from ole32.dll
Private Declare Function CoInitializeSecurity Lib "OLE32.DLL" ( _
    pSD As Any, _
    ByVal cAuthSvc As Long, _
    asAuthSvc As Long, _
    pReserved1 As Any, _
    ByVal dwAuthnlevel As Long, _
    ByVal dwImpLevel As Long, _
    ByVal pAuthInfo As Long, _
    ByVal dwCapabilities As Long, _
    pvReserved2 As Any _) As Long
```

```
Sub Main()
  ' following code will fail in the IDE (debug mode).

  Dim lngHr As Long
  Dim lngAuthn As Long
  lngAuthn = RPC_C_AUTHN_NONE
  lngHr = CoInitializeSecurity(ByVal API_NULL, _
      -1, _
      lngAuthn, _
      ByVal API_NULL, _
      RPC_C_AUTHN_LEVEL_NONE, _
      RPC_C_IMP_LEVEL_IMPERSONATE, _
      API_NULL, _
      EOAC_NONE, _
      ByVal API_NULL)

   If (S_OK <>lngHr) Then
      MsgBox "CoInitializeSecurity failed with error code: 0x" _
      & Trim$(Str$(Hex(lngHr))) & vbCrLf & _
      "Ignore, if running in IDE - but you will not receive events " & _
      "from remote microscope server", _
      vbCritical, _
      "Application Initialization Failure"
      Exit Sub
  End If
  ' Any additional code you need here.
  frmYourframe.Show ' showing your starting form
End Sub
```

Now, in Project--Properties--General tab, change the startup object to Sub Main. Note that Sub Main has to have code to make your forms visible (such as `frmYourframe.Show`). The call to CoInitializeSecurity will fail, when you are working in the VB development environment (IDE), debugging your application. In that case, the IDE has already initialized COM security. This means, you will receive events only, if you run the compiled executable.

28.7 Errors and error handling

Some of the microscopes functions may return (raise) an error. This can happen due to a physical reason, i.e. if the requested action is not possible at that moment. For example, you cannot ask the stage to perform a GoTo(), when it is wobbling or already moving. The high tension may not be switched on or raised under certain conditions and so forth. You may also have given parameters that are out of range (in that case the stage for example would not move either).
Generally, calls to another process (from your script to the TEM server) may fail occasionally. There are also some failures that may be VisualBasic-specific (see below).

So be aware and do the error handling - otherwise your script might die!

If an error is raised, then a global Error object (`Err`) will be filled with detailed information about the error, if available. You can use this information for your error handling. Usually you will get an error code (a number), a textual description and maybe a text string with the name of the source that raised the error.

The following piece of code gives an example:

(Suppose you have a form with at least a button named ‘cmdGoto’ and a textbox named ‘txtDisplay’. As reaction on a button-click a new stage position is calculated -just as an example, you could invent some

code here- and the stage is requested to move to that position. In case of an error the content of the error object is shown in the textbox):

```
Private Sub cmdGoto_Click()
Dim NewPos  as TEMScripting.StagePosition
Dim OldPos as TEMScripting.Stageposition
Dim MyStage as TEMScripting.Stage
On Error GoTo ComError:
   Set MyStage = MyTem.Stage
   Set OldPos  = MyStage.Position
   CalculateNewPosition OldPos, NewPos
   MyStage.Goto NewPos, axisXY
Exit Sub
ComError:
   txtDisplay.Text = _
     txtDisplay.Text & "Error # " & Hex(Err.Number) & _
     " was generated by " & Err.Source & vbCrLf _
     & Err.Description & vbCrLf
   txtDisplay.SelStart = Len(txtDisplay.Text)
End Sub
```

At the end of the error handling you can either do nothing (as in the above example), that is leave the function or add either

```
Resume
```

which jumps back to the line in your code where the error occurred or

```
Resume Next
```

which jumps back to the next line in your code

```
Goto MyLabel
```

jump to a specified label, here named `MyLabel`.

```
Another possibility is to use the

On Error Resume Next
 ...
On Error Goto 0
```

statements. This allows you either to ignore errors or ask for the contents of the Err object after each relevant statement and do the error handling ‘in place’. If you handled an error you should then clear the content of the error object, using `Err.Clear`.
For more information see the Visual Basic documentation.

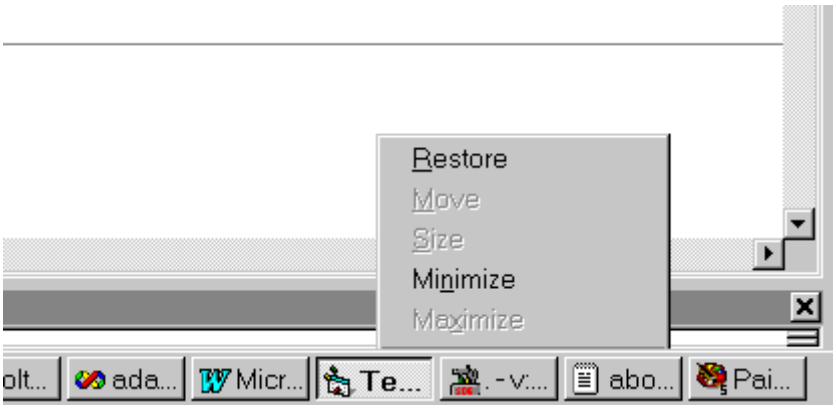
28.8 Visual Basic-specific Errors

At last, two examples of errors that you might encounter and which may be specific to Visual Basic 5.0 (but we have not yet checked in other scripting languages and the 6.0 version):

1) From a VB application it is not allowed to do more than one out-of-process call at a time. You might think that this is a rare situation, but you can easily encounter this case. Imagine the following: You have a timer running in your application that polls for certain microscope parameters of interest in regular time intervals. (Maybe you want to protocol the vacuum pressures or the stage position.) At the same time

your application has a user interface, through which the user can request actions by pressing a button (maybe start a pumping cycle or an exposure of some kind). Then, especially when you use functions that take a long time to complete, there is a chance that your application tries to issue a call from your timer loop while a first one (issued via a button command) did not yet return. This second call will then fail (error description: "illegal to call out while inside message filter"). The reason is the way the apartment threading and message loop are implemented in VB.

2) A right click on the applications button in the Windows button list will also cause the out of process calls to fail, as long as the small "Restore, Move, Size etc."-menu is open (see image below). We did not do any further research why this happens, but it may be related to 1).



29 Revision History

31st March 2012, Dave Karetnyk:

- Revised for Tecnai 4.3 software release.
- Interface additions to support JavaScript and VBScript clients retrieving image data.
- Additions to JScript chapter, examples on image acquisition in particular.
- Gauge Objects section: virtual/pseudo names are no longer supported.

20st August 2013, Max Otten

- Inserted note about no 64-bits scripting
- Inserted note about no remote acquisition possible