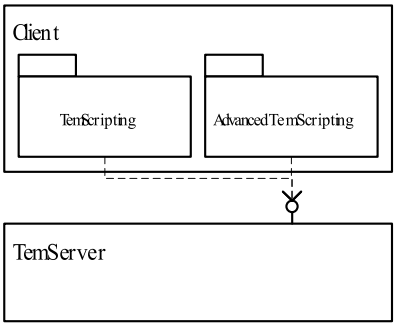
(Advanced) TEM Scripting User Guide

Document status: v2.4 proposal

Contents

Introduction

TEM Scripting is a scripting adapter component that enables customers and 3rd party applications to automate the operation of their TEM microscope. It provides an interface through which scripts can control all TEM subsystems and perform acquisitions with the available detectors.

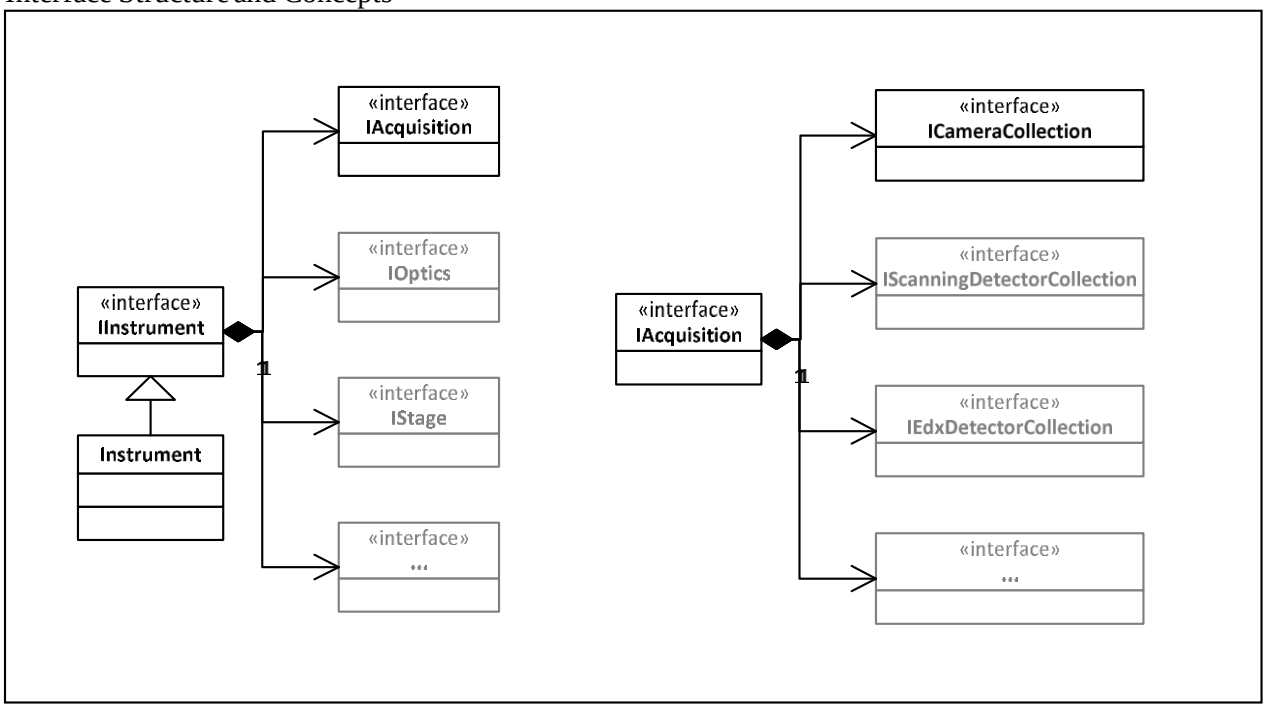


The interface is limited in scope such in order to make it stable across many TEM Server releases.

The existing TEM Scripting component does not properly support the Falcon cameras especially for handling the dose fractions that these cameras produce.

A new component is provided that directly interfaces with the TEM Server, providing access to single image acquisitions from the Falcon cameras. The component has been named AdvancedTemScripting.

Interface Structure and Concepts



The Advanced Scripting component provides a COM-based interface. COM is broadly supported in various scripting (and more general, application) environments. The set of interfaces is structured such as to allow easy usage from single-threaded clients (such as scripts): only calls, no events or callbacks, synchronous calls, polling for state, no concurrency.

Navigation

The main interface is IInstrument. From this, all other interfaces are reached by navigating through a tree-like structure. The Instrument object that implements IInstrument is creatable and the created instance is owned by the client (script) that created it. As long as the client holds on to the Instrument object, it can use it to navigate the subsystem tree. Discarding the Instrument object breaks the connection of the client to the server and renders any references obtained during navigation invalid.

```
Camera FindCameraByName(string name, CameraList cameras)
{
```

```
    foreach(Camera c in cameras)
        if (c.Name == name)
            return c;
    return null;
}

var instrument = new TEMAdvancedScripting.Instrument();
var csa = instrument.Acquisitions.CameraSingleAcquisition;
var allCameras = instrument.Acquisitions.Cameras;
var singleAcquisitionCameras = csa.SupportedCameras;
var falconCamera = FindCameraByName("BM-Falcon", singleAcquisitionCameras);
csa.Camera = falconCamera;
var cs = csa.CameraSettings;
```

Copy vs Reference

Simple objects are returned as copies (aka "data" objects"). Any changes to such objects are not applied to the underlying system. This currently only applies to objects like the FrameRange. For example:

```
FrameRangeList dfd = cs.DoseFractionsDefinition;
dfd[0].Begin = 2; // This will not modify the range value
```

Objects that are parts of the navigation tree are returned by reference and any action or change performed on them affects the state of the scripting component. For example, changing values on the CameraSettings object retrieved through the CameraSingleAcquisition will apply the changed values:

```
cs.ExposureTime = 0.25; // This will be applied
```

Objects that originate in capabilities are read-only values that can be selected from a list and applied. This currently refers to Binning:

```
BinningList binnings = cc.SupportedBinnings;
cs.Binning = binnings[0];
```

Capability-driven

To avoid an extensive system of interface inheritance the interfaces are designed to be capability-driven. This means that starting with a basic set of operations and properties on the main interface of e.g. a detector other sets of operations and properties (or features) that more specialized versions of the detector provide are organized into navigable subsystems or subsets of operations and properties.

For such cases test functions are provided (Has__, Is__, Can__, Supports__) that can be used before accessing the respective setting or operation.

Threading

The interfaces are not meant to be accessed from multiple threads concurrently.

Supported Cameras

The Acquisitions object contains all the use-cases and detectors available. The list of cameras returned through the Acquisitions object contains all the cameras that TEMAdvancedScripting supports. Currently, these are: Ceta 1, Ceta 2, Falcon 3 and Falcon 4.

Two use-cases are supported at this moment: the `CameraSingleAcquisition` and `CameraContinuousAcquisition`.

The list of cameras returned by `CameraSingleAcquisition` contains the subset of all the cameras that can participate in the use-case, which actually are all cameras.

```
var instrument = new TEMAdvancedScripting.Instrument();
var allCameras = instrument.Acquisitions.Cameras;
var singleAcquisitionCameras =
instrument.Acquisitions.CameraSingleAcquisition.SupportedCameras;
```

The list of cameras returned by `CameraContinuousAcquisition` contains the subset of all the cameras that can participate in the use-case, which actually is only Ceta2.

```
var instrument = new TEMAdvancedScripting.Instrument();
var allCameras = instrument.Acquisitions.Cameras;
var continuousAcquisitionCameras =
instrument.Acquisitions.CameraContinuousAcquisition.SupportedCameras;
```

Performing Camera Acquisitions

The example code for performing a simple single image acquisition looks like this:

```
Camera FindCameraByName(string name, CameraList cameras)
{
    foreach(Camera c in cameras)
        if (c.Name == name)
            return c;
    return null;
}

var instrument = new TEMAdvancedScripting.Instrument();
var csa = instrument.Acquisitions.CameraSingleAcquisition;
var allCameras = instrument.Acquisitions.Cameras;
var singleAcquisitionCameras = csa.SupportedCameras;
var falconCamera = FindCameraByName("BM-Falcon", singleAcquisitionCameras);
csa.Camera = falconCamera;
var cs = csa.CameraSettings;

// Perform any settings necessary; some examples:
cs.ExposureTime = cs.ExposureTimeRange.Max;
if (cs.SupportsDriftCorrection)
    cs.AlignImage = true;
// Here: Optionally, setup the dose fractions
dfd = cs.DoseFractionsDefinition;
dfd.Clear();
dfd.AddRange(fs1, fe1); dfd.AddRange(fs2, fe2);
//...
dfd.AddRange(fsn, fen);
assert(fsx < fex);
assert(fen <= cs.CalculateNumberOfFrames());

// Perform the acquisition; the final image is received as a return value
var img = csa.Acquire();
var metadata = img.Metadata;

// Here: Save or otherwise use the image and metadata
```

```
// ...

// Wait for the camera to become available for another acquisition csa.Wait();
// or
// while (csa.IsActive) sleep(some);

// Here: Access and process images available at cs.PathToImageStorage
```

Figure 1 Basic single-image camera acquisition

When the instrument object is released, all other objects become invalid references that can no longer be used.

To perform an acquisition with a camera, it has to be added to a use-case. In the example above, the use-case only accepts one camera. Then the acquisition is triggered by calling the Acquire method on the single acquisition use-case. Configuration of the camera takes place through the CameraSettings object.

Dose Fractions

A single acquisition returns one image called the "final image" which is the integrated image over the exposure time. The Falcon cameras can also provide intermediate images called "dose fractions". The dose fractions are temporarily stored on the camera device itself until they get downloaded.

When defining fractions the client needs to know the number of frames that the acquisition will produce. This depends on acquisition settings, especially on the exposure time. So it is wise to first configure all settings and leave dose fractions for last, when the number of frames has settled.

The TEM scripting component will download the dose fractions to either a local directory (hardcoded to: c:\OffloadData) or to a storage server (hardcoded to: \\192.168.10.2\OffloadData) before allowing the client to perform a new acquisition. The download itself takes place asynchronously so the script may perform other operations while the download is in progress. Before attempting to start a new acquisition the script should Wait() for the previous one to finish.

After the Wait() returns the images will be available at the location indicated by PathToImageStorage which can either be a local directory or a UNC path to a directory on the storage.

EER

The Falcon 4 camera supports storing every captured electron counted camera frame within the EER file format. EER is a "special" dose fraction scheme and downloading of the images (to the storage server) is handled the same as for Dose Fractions, only the file format and extension differs.

EER can only be enabled in case Electron Counting is enabled and can not be used in combination with dose fractions.

File naming

Please note that in the examples .mrc is shown as file extension, in case EER is enabled the extension will be .eer

When nothing else is specified, the intermediate images are saved in a file that is named with a timestamp composed of the current date and time: YYYYMMDDhhmmss.
The client can specify a pattern to be used for the file and subdirectory name:

```
cs.SubPathPattern = "dir\\file";
```

The pattern is given in the form of a relative path in which the last element is interpreted as the file name. Also, special tags can be used to be replaced with current values. The tags supported are:

- {ymd} – replaced with the current date in the format YYYYMMDD
- {hms} – replaced with the current time in the format hhmmss

Examples of valid formats:

- "first_acquisition" – a file will be created called first_acquisition.mrc
- "my_study\\first_acquisition" – a file will be created called first_acquisition.mrc inside the subdirectory called my_study
- "my_study\\" – file names will be formatted YYYYMMDDhhmmss.mrc and will be saved in subdirectory "my_study"
- "day_{ymd}\\acquisition_{hms}" – file names will be formatted as acquisition_hhmmss.mrc and placed in the subdirectory day_YYYYMMDD

Continuous Acquisition Recording

There are two ways to do a Continuous Acquisition and both are Recording Acquisitions. If no `RecordingDuration` is set in the `CameraSettings`, the recording will last till `Stop()` is called. If a `RecordingDuration` is set, `Wait()` can be used to wait till the set duration of the acquisition has been acquired.
The `RecordingDuration` set is the minimum amount of time the acquirement will take, as it will take as much complete frames with the set exposure time as is needed to get to the set RecordingDuration. E.g. if the exposure time is 0.5 and the RecordingDuration is 2.3, there will be an acquirement of 2.5 (5 frames).

Image and file formats

Dose Fractions

The Storage Server saves acquired images in the MRC file format with an associated XML file containing metadata information. It uses one MRC-XML pair for all the dose fractions and one for the final image.

The same strategy has been implemented in the Advanced Scripting too. Images saved locally also use the MRC-XML format.

The MRC format can only contain images of up to 16-bits per pixel, thus all images will internally be adapted to fit inside this format. The (down-)scaling factor used for the pixel values is multiplied into the PixelValueToCameraCounts metadata factor. For a client to get the actual pixel value as read by the camera (called CameraCounts) it must multiply each pixel value in the image by the PixelValueToCameraCounts factor in the metadata.

EER

The Storage Server saves acquired images in the EER file format, containing the metadata and every captured electron counted camera frame.

Final image

Final Images returned in memory are not converted, so the client must be able to accept 16-bit and 32-bit images. See the interface description for more details.

Metadata

Metadata information is returned together with the in-memory image. For the dose fractions, metadata is saved in an XML file with the same name as the MRC file containing the images.

Every metadata parameter can be read as a string or a COM variant.
The following metadata is expected to be present:

TimeStamp

- DetectorName
- AcquisitionUnit
- ImageSize.Width
- ImageSize.Height
- Encoding
- BitsPerPixel
- Binning.Width
- Binning.Height
- ReadoutArea.Left
- ReadoutArea.Top
- ReadoutArea.Right
- ReadoutArea.Bottom
- ExposureTime
- DarkGainCorrectionType
- Shutters[0].Type
- Shutters[0].Position
- PixelValueToCameraCounts
- AlignIntegratedImage
- DriftCorrected (optional, only for drift corrected images)
- DriftCorrectionConfidence (optional, only for drift corrected images)
- DriftCorrectionClipping (optional, only for drift corrected images)
- DriftCorrectionVectorX (optional, only for drift corrected images)
- DriftCorrectionVectorY (optional, only for drift corrected images)
- ElectronCounted
- CountsToElectrons (optional, only for electron counted images)

Failures

No license

If the feature requested is not licenced, the command will be refused and an error will occur.

Inconsistency in parameters

In some cases it is possible that the set of parameters chosen by the client is inconsistent. In such a situation the acquisition will fail and the exception text will indicate which parameters have been adjusted to make the set consistent.

Camera in use

A camera can only be used in one acquisition at one time. If another client (script or other) is using the camera `Acquire()` will fail. `Acquire()` can also fail when the camera isn't used in another acquisition but cannot be inserted due to a physical conflict (i.e. another camera is inserted) or a defect.

Camera offline

A camera may be present in the system configuration but may be unavailable due to e.g. a network cable disconnect or hardware being powered off. In such cases `Acquire()` will fail and the text of the exception will indicate the most likely cause. Other operations on the camera may also fail in such a situation.

Storage server unavailable

The Storage Server may become (temporarily) unavailable. The acquisition will fail in case images should be downloaded to the storage server (e.g. Dose Fractions or EER being enabled).

Camera Support

The scripting adapter component mainly supports the Falcon cameras for single acquisition. It also supports the Ceta and Ceta2 cameras but only for very limited single acquisition scenarios.

For continuous acquisition only Ceta2 Recording acquisitions are supported.

Supported Electron Sources

The scripting adapter component supports a limited set of operations on electron sources with the following emitter types:

- FEG (including High Brightness FEG and Shottky FEG)
- Cold FEG

The adapter provides both basic operations, which are suitable for all emitter types, and emitter-specific operations. Refer to the *Interface Description* section for details.

Interface Description

`IInstrument` interface

The entry into the AdvancedScripting interface.

Acquisitions

- Sort: property, read-only
- Type: Acquisitions
- Description: returns the list of all implemented use cases and detectors

Phaseplate

- Sort: property, read-only
- Type: Phaseplate
- Description: returns the phaseplate interface

Source

- Sort: property, read-only
- Type: IDispatch
- Description: returns the electron source interface, if supported for the current microscope configuration; otherwise, a non-initialized object is returned. See *Supported Electron Sources* for the list of supported configurations.

EnergyFilter

- Sort: property, read-only
- Type: IDispatch (EnergyFilter)
- Description: returns the energy filter interface, if supported for the current microscope configuration; otherwise, a non-initialized object is returned.

Acquisitions interface

Represents the list of all implemented use cases and the list of all implemented detectors available in the system. Only camera detectors are implemented at the moment.

Cameras

- Sort: property, read-only
- Type: CameraList
- Description: returns a list of all cameras supported by the AdvancedScripting component

CameraSingleAcquisition

- Sort: property, read-only
- Type: CameraSingleAcquisition
- Description: returns a CameraSingleAcquisition object that will be used to perform single acquisitions using cameras

CameraContinuousAcquisition

- Sort: property, read-only
- Type: CameraContinuousAcquisition
- Description: returns a CameraContinuousAcquisition object that will be used to perform continuous acquisitions using cameras

CameraSingleAcquisition interface

Represents the single acquisition (one image) use case performed with a camera. The Falcon camera's allow the acquisition of intermediate images.

SupportedCameras

- Sort: property, read-only
- Type: CameraList
- Description: returns a list of cameras that support the single acquisition use-case (which currently are all cameras

Camera

- Sort: property, write-only

- Type: Camera
- Description: associates one camera with this use-case; this camera will perform the use-case; if no camera has been associated with the use-case prior to starting the acquisition, the acquisition will fail

CameraSettings

- Sort: property, read-only
- Type: CameraSettings
- Description: this is a view on the camera where settings can be applied; when no camera is set in the use-case – through the Camera property – attempting to read this property will fail; also, settings will be remembered for each given camera, even after it has been taken out of the use-case

Acquire

- Sort: method/function
- Return type: AcquiredImage
- Description: triggers the start of the acquisition; fails if settings not coherent; returns when the final (integrated) image has been acquired

IsActive

- Sort: property, read-only
- Type: boolean (VARIANT_BOOL)
- Description: used to poll for the end of the complete acquisition; used as a non-blocking wait for the downloading or offloading of intermediate images; no Acquire can be done while IsActive is true

Wait

- Sort: method/function
- Return type: none
- Description: blocking wait for the end of the complete acquisition

CameraContinuousAcquisition interface

Represents the continuous acquisition (multiple images) use case performed with a camera. Currently only supports Ceta2 recording acquisitions.

SupportedCameras

- Sort: property, read-only
- Type: CameraList
- Description: returns a list of cameras that support the single acquisition use-case (which currently is only Ceta2)

Camera

- Sort: property, write-only
- Type: Camera

- Description: associates one camera with this use-case; this camera will perform the use-case; if no camera has been associated with the use-case prior to starting the acquisition, the acquisition will fail

CameraSettings

- Sort: property, read-only
- Type: CameraSettings
- Description: this is a view on the camera where settings can be applied; when no camera is set in the use-case – through the Camera property – attempting to read this property will fail; also, settings will be remembered for each given camera, even after it has been taken out of the use-case

Store

- Sort: method/function
- Return type: none
- Description: triggers the start of the acquisition; fails if settings not coherent; returns when the acquisition has started.

IsActive

- Sort: property, read-only
- Type: boolean (VARIANT_BOOL)
- Description: used to poll for the end of the complete acquisition; used as a non-blocking wait for the downloading or offloading of intermediate images; no Acquire can be done while IsActive is true

Wait

- Sort: method/function
- Return type: none
- Description: blocking wait for the end of the complete acquisition and offloading job completion, will throw when RecordingDuration has not been set, as there is nothing to wait for.

Stop

- Sort: method/function
- Return type: none
- Description: blocking wait which stops the acquisition and waits for offloading to be complete.

AcquiredImage interface

Represent the final image acquired by the CameraSingleAcquisition.

Width

- Sort: property, read-only
- Type: long
- Description: the width of the image

Height

- Sort: property, read-only
- Type: long
- Description: the height of the image

PixelType

- Sort: property, read-only
- Type: ImagePixelType
- Description: the PixelType of the image, can be ImagePixelType_UnsignedInt, ImagePixelType_SignedInt or ImagePixelType_Float

BitDepth

- Sort: property, read-only
- Type: long
- Description: the bit depth of the pixels in the image

Metadata

- Sort: property, read-only
- Type: KeyValuePairList
- Description: the metadata belonging to the image

AsSafeArray

- Sort: property, read-only
- Type: SAFEARRAY(long)
- Description: The image is returned as SAFEARRAY(long). The final image has been converted to long without any scaling. One can use the methods BitDepth and PixelType to retrieve the original format.

AsVariant

- Sort: property, read-only
- Type: VARIANT
- Description: The image is returned as VARIANT and is not converted, so the client must be able to accept 16-bit and 32-bit images. One can use the vt member of the VARIANT to get the datatype of the underlaying data.

SaveToFile

- Sort: method/function
- Input arguments: BSTR filePath, VARIANT_BOOL bNormalize
- Return type: none
- Description: save the image as raw image on disk. Filepath indicates the location where the image should be stored and bNormalize indicated if the image should be normalized. bNormalize has the default value VARIANT_FALSE

Camera interface

Represents a camera device and its use case independent properties.

Name

- Sort: property, read-only
- Type: string (BSTR)

- Description: the conventional name of the camera used to identify it throughout the TEM server

Width, Height

- Sort: property, read-only
- Type: long
- Description: the count of pixels along each axis

PixelSize

- Sort: property, read-only
- Type: PixelSize
- Description: the physical size of a pixel (in m) along each axis

Insert, Retract

- Sort: method/function
- Return type: none
- Description: inserts/retracts the camera; returns only when the camera is fully inserted of retracted; fails with an exception if problems encountered

IsInserted

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)
- Description: checks the insertion status of the camera; may fail with exception in case of (communication) problems

CameraSettings interface

Represents the set of use case dependent settings that are to be used in the acquisition for the camera.

Capabilities

- Sort: property, read-only
- Type: Capabilities
- Description: returns the set of use case dependent capabilities for the camera

PathToImageStorage

- Sort: property, read-only
- Type: string (BSTR)
- Description: returns the base directory where the intermediate images will be saved;

SubPathPattern

- Sort: property, read-write
- Type: string
- Description: sets and retrieves the subpath pattern for the storage of intermediate images; this subpath is to be appended to PathToImageStorage to get the full path to the images; it can be set before each acquisition to place intermediate images grouped by acquisition

ExposureTime

- Sort: property, read-write
- Type: double
- Description: sets and retrieves the exposure time for the acquisition (in s); the actual exposure time can be slightly higher or slightly lower that the one set

ReadoutArea

- Sort: property, read-write
- Type: enumeration: Full, Half, Quarter
- Description: sets and retrieves the area to be read from the camera sensor; the area is defined around the center of the sensor, horizontally as well as vertically

Binning

- Sort: property, read-write
- Type: Binning
- Description: sets and retrieves the binning factor to be used with the acquisitions; it has a horizontal and a vertical component; it must be one of the values retrieved by the SupportedBinnings property in CameraCapabilities

DoseFractionsDefinition

- Sort: property, read-only
- Type: FrameRangeList; a list of pairs of long values (Begin, End) where Begin is the first frame, End is the last frame plus 1
- Description: retrieves the list of frame ranges that define the intermediate images; the list can be cleared and filled with the ranges; for Ceta 1 & 2 it throws and exception

AlignImage

- Sort: property, read-write
  Type: boolean (BSTR)
- Description: sets and retrieves whether frame alignment (i.e. drift correction) is to be applied to the final image as well as the intermediate images

ElectronCounting

- Sort: property, read-write
- Type: Boolean (BSTR)
- Description: sets and retrieves whether electron counting mode is to be used in the acquisition; In case electron counting is not supported it will throw.

EER

- Sort: property, read-write
- Type: Boolean (BSTR)
- Description: sets and retrieves whether EER mode is enabled; In case EER is not supported it will throw. ElectronCounting should always be enabled when selecting EER.

CalculateNumberOfFrames

- Sort: method/function
- Return type: long
- Description: retrieves the number of frames that the acquisition produces; it is dependent on other settings so it is better called last; it is used to determine the ranges for the DoseFractionsDefinitions

RecordingDuration

- Sort: property, read-write
- Type: double
- Description: sets and retrieves the recording duration for the acquisition (in s); the actual recording duration can be slightly higher, the value set is the minimum. In case recording duration is not supported it will throw.

CameraAcquisitionCapabilities interface

Represents the use case dependent capabilities of the camera. Defines acceptable features and ranges of acceptable values for the settings.

SupportedBinnings

- Sort: property, read-only
- Type: BinningList
- Description: returns a list of binnings supported by the camera; one of the binnings in this list can be used to set the Binnning setting

ExposureTimeRange

- Sort: property, read-only
- Type: ITimeRange
- Description: returns the range of exposure times supported by the camera

SupportsDoseFractions

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)
  Description: returns whether the camera supports the acquisition of dose fractions; for cameras that don't support it, attempting to specify dose fractions will fail with an exception

MaximumNumberOfDoseFractions

- Sort: property, read-only
- Type: long
- Description: returns the maximum number of dose fractions that the camera supports;

SupportsDriftCorrection

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)

- Description: returns whether the camera supports drift correction; attempting to use drift correction (AlignImage) for cameras that don't support it will lead to an exception

SupportsElectronCounting

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)
- Description: returns whether the camera supports the electron counting mode; attempting to use the electron counting mode for cameras that don't support it will lead to an exception

SupportsEER

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)
- Description: returns whether the camera supports the EER; attempting to use EER for cameras that don't support it will lead to an exception

SupportsRecordingDuration

- Sort: property, read-only
- Type: Boolean (VARIANT_BOOL)
- Description: returns whether the camera supports Recording; attempting to use Recording for cameras that don't support it will lead to an exception. Setting the value to 0 or less will disable RecordingDuration.

Phaseplate interface

Represents the phaseplate aperture which resides on the Objective ApertureMechanism. The phaseplate is an aperture with a thin layer of material, which when hit with an electron beam only allows electrons with a high enough energy level pass. This material gets saturated at the point where the beam is located, and therefore it is required to periodically change over to a new location within the aperture. A select number of predefined locations is available, which can be looped over by calling a 'next' function.

Preconditions: In order to be able to successfully call these methods, the following must be true:

- The microscope contains a motorized objective aperturemechanism
- The server is running
- The objective aperturemechanism is enabled, and a phaseplate aperture is selected The appropriate license must be available

SelectNextPresetPosition

- Sort: method
  Return type: none
- Description: Goes to the next preset location on the aperture

GetCurrentPresetPosition

- Sort: property, read-only
- Type: long

- Description: returns the zero-based index of the current preset position.

Source interface

Feg

Represents either FEG or Cold FEG electron source. The following basic set of operations is supported for both types of FEG electron source.

State

- Sort: property, read-only
- Type: FegState : enumeration {FegState_NotEmitting, FegState_Emitting}
- Description: returns current state of FEG electron source (not emitting or emitting electrons).

ExtractorVoltage

- Sort: property, read-only
- Type: double
- Description: returns current extractor voltage setpoint [V].

FocusIndex

- Sort: property, read-only
- Type: FegFocusIndex : struct { Coarse : long, Fine : long }
- Description: returns current FEG focus index (a.k.a. “Gunlens index”), which consists of two parts – the Coarse setting and the Fine setting.
- Note: retrieving this property will fail if the current microscope configuration includes the Monochromator (as the Focus index concept is not supported for Monochromator).

The following subset of operations is supported for a Cold FEG electron source only.

BeamCurrent

- Sort: property, read-only
- Type: double
- Description: returns current actual beam current [A].

Flashing

- Sort: property, read-only
- Type: IDispatch
- Description: returns FegFlashing interface if the current electron source type is Cold FEG; otherwise, a non-initialized object is returned.

FegFlashing

The following are the members of FegFlashing interface.

IsFlashingAdvised

- Sort: method/function

- Input arguments: flashingType : FegFlashingType
- Return type: Boolean (VARIANT_BOOL)
- Description: returns the Flashing Advised status for the flashing type defined by ‘flashingType’ parameter.

PerformFlashing

- Sort: method/function
- Input arguments: flashingType : FegFlashingType
- Return type: none
- Description: executes Flashing of the type defined by ‘flashingType’ parameter. For High Temperature Flashing this succeeds only if this type of flashing is advised.

EnergyFilter interface

The EnergyFilter object gives access to the properties below.
In case the EnergyFilter does not support a property, a non-initialized object (null) is returned.
In case the EnergyFilter is not licensed, accessing these properties is refused and an error will occur.

Slit

- Sort: property, read-only
- Type: IDispatch
- Description: Gives access to the Slit interface.
  .

HighTensionEnergyShift

- Sort: property, read-only
- Type: IDispatch
- Description: Gives access to the HighTensionEnergyShift interface.

ZeroLossPeakAdjustment

- Sort: property, read-only
- Type: IDispatch
- Description: Gives access to the ZeroLossPeakAdjustment interface.

Slit

The following are the members of the Slit interface.

IsInserted

- Sort: property, read-only
- Type: boolean (VARIANT_BOOL)
- Description: returns the whether the Slit is currently inserted (VARIANT_TRUE) or retracted (VARIANT_FALSE).

Insert

- Sort: method/function

- Input arguments: none
- Return type: none
- Description: Inserts the Slit into the filter. The Slit will move to the current setting of Width property
  In case the Insertion of the Slit failed an error is returned.

Retract

- Sort: method/function
- Input arguments: none
- Return type: none
- Description: Retracts the Slit from the filter
  In case the Retraction of the Slit failed an error is returned.

WidthRange

- Sort: property, read-only
- Type: EnergyRange
- Description: Returns allowed value range for Slit Width property

Width

- Sort: property, read-write
- Type: double
- Description: get or set the slit width in [eV].
  On read, Slit inserted: current Width position
  On read, Slit retracted: Width position the Slit will move to when Insert is called
  On write, Slit inserted: Slit moves immediately to the requested width
  On write, Slit retracted: updates Width, slit will not move, only activated after Insert is called
  Set values shall be within the range as specified by the *WidthRange* property.
  In case an invalid (out of range) Width value is set an error is returned.

HighTensionEnergyShift

The following are the members of the Slit interface.

EnergyShiftRange

- Sort: property, read-only
- Type: EnergyRange
- Description: Returns allowed value range for EnergyShift property

EnergyShift

- Sort: property, read-write
- Type: double
- Description: get or set the High Tension EnergyShift in [eV].
  Set values shall be within the range as specified by the *EnergyShiftRange* property.
  In case an invalid (out of range) EnergyShift value is set an error is returned.

ZeroLossPeakAdjustment

Allows adjusting the Zero Loss Peak Adjustment.

EnergyShiftRange

- Sort: property, read-only
- Type: EnergyRange
- Description: Returns allowed value range for EnergyShift property

EnergyShift

- Sort: property, read-write
- Type: double
- Description: get or set the Zero Loss Peak EnergyShift in [eV].
  Set values shall be within the range as specified by the *EnergyShiftRange* property.
  In case an invalid (out of range) EnergyShift value is set an error is returned.

EnergyRange

Properties of this type contain a range of Energy values in electron volt.

Begin

- Sort: property, read-only
- Type: double
- Description: Returns lowest value of the energy range [eV]

End

- Sort: property, read-only
- Type: double
- Description: Returns highest value of the energy range [eV]

AutoLoader interface

The AutoLoader object gives access to the members below.
In case the AutoLoader is not licensed, accessing these members is refused and an error will occur.

NumberOfCassetteSlots

- Sort: property, read-only
- Type: long
- Description: returns total number of slots in the cassette

SlotStatus

- Sort: property, read-only
- Type: CassetteSlotStatus
- Description: returns the CassetteSlotStatus of the provided slot number.

  The slot status can be CassetteSlotStatus_Unknown, CassetteSlotStatus_Occupied, CassetteSlotStatus_Empty or CassetteSlotStatus_Error

Initialize

- Sort: method/function

- Input arguments: None
- Return type: None
- Description: Initializes / Recovers the Autoloader for further use.

`DockCassette`

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Moves the cassette from the capsule to the docker.

`UndockCassette`

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Moves the cassette from the docker to the capsule.

`PerformCassetteInventory`

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Perform an inventory of all cassette the slots for the presence of a cartridge.

`LoadCartridge`

- Sort: method/function
- Input arguments: slot number to load. Should be 0 < slot_number <= NumberOfCassetteSlots
- Return type: None
- Description: Loads the cartridge from a specified slot in the cassette to the stage.

`UnLoadCartridge`

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Unload cartridge from stage to a free slot in the cassette.

`BufferCycle`

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Synchronously runs the Autoloader buffer cycle.

TemperatureControl interface

The TemperatureControl object gives access to the members below.
In case the TemperatureControl is not licensed, accessing these members is refused and an error will occur.

AutoloaderCompartment

- Sort: property, read-only
- Type: IDispatch (`AutoloaderCompartment`)
- Description: Gives access to the AutoloaderCompartment interface.

ColumnCompartment

- Sort: property, read-only
- Type: IDispatch (`ColumnCompartment`)
- Description: Gives access to the ColumnCompartment interface.

IsAnyDewarFilling

- Sort: property, read-only
- Type: boolean (VARIANT_BOOL)
- Description: Returns whether any of the Dewars (Autoloader or Column) is busy filling.

RefillAllDewars

- Sort: method/function
- Input arguments: None
- Return type: None
- Description: Force the refrigerant refill of autoloader and column dewars.

AutoloaderCompartment

The following are the members of the AutoloaderCompartment interface.

CartridgeTemperature

- Sort: property, read-only
- Type: double
- Description: Returns Cartridge gripper temperature in Kelvins.

CassetteTemperature

- Sort: property, read-only
- Type: double
- Description: Returns Cassette gripper temperature in Kelvins.

DockerTemperature

- Sort: property, read-only
- Type: double
- Description: Returns Docker temperature in Kelvins.

Dewar

- Sort: property, read-only
- Type: IDispatch (`Dewar`)
- Description: Gives access to the Dewar interface. Provides access to the AutoLoader Dewar.

ColumnCompartment

The following are the members of the ColumnCompartment interface.

HolderTemperature

- Sort: property, read-only
- Type: double
- Description: Returns Holder temperature in Kelvins.

Dewar

- Sort: property, read-only
- Type: IDispatch (`Dewar`)
- Description: Gives access to the Dewar interface. Provides access to the Column Dewar.

Dewar

The following are the members of the Dewar interface.

RefrigerantLevel

- Sort: property, read-only
- Type: double
- Description: Returns the current refrigerant level of the dewar. Refrigerant level is specified in percentage (%). 0 means empty, 100 means full.

DewarRemainingTime

- Sort: property, read-only
- Type: double
- Description: Returns the dewar remaining time before it needs to be refilled. The remaining time is specified in seconds.

Client script examples

Image Acquisition and Phase Plate

```
# Sample python script to show usage of Advanced Scripting
#
import comtypes
import comtypes.client as cc

instrument = cc.CreateObject("TEMAdvancedScripting.AdvancedInstrument")

def TakeSingleAcquisition():
	csa = instrument.Acquisitions.CameraSingleAcquisition
	cams = csa.SupportedCameras
	csa.Camera = cams[[c.name for c in cams].index("BM-Falcon")]
	cs = csa.CameraSettings
	cs.DoseFractionsDefinition.Clear
	cs.DoseFractionsDefinition.AddRange(0, 1)
	cs.DoseFractionsDefinition.AddRange(1, 2)
	cs.DoseFractionsDefinition.AddRange(2, 3)
	cs.SubPathPattern = "Jupyter_{ymd}\\{hms}"
	ai = csa.Acquire()
	ai.SaveToFile("c:\\users\\factory\\desktop\\myimage.raw")
	array = ai.AsSafeArray

def PhaseplateSelectNext():
	print("PhaseplateSelectNext")
	pp = instrument.Phaseplate
	pp.SelectNextPresetPosition()
	print("PhaseplateSelectNext done")

def PhaseplateGetCurrent():
	print("PhaseplateGetCurrent")
	pp = instrument.Phaseplate
	print(pp.GetCurrentPresetPosition)
	print("PhaseplateGetCurrent done")
def PhaseplateGetCurrentAndSelectNext():
	print("PhaseplateGetCurrentAndSelectNext")
	pp = instrument.Phaseplate
	pp.SelectNextPresetPosition()
	print pp.GetCurrentPresetPosition
	print("PhaseplateGetCurrentAndSelectNext done")

def PhaseplateGetCurrentAndSelectNextAndGetCurrent():
```

```
      print("PhaseplateGetCurrentAndSelectNextAndGetCurrent")
      pp = instrument.Phaseplate
      for x in range(1,100):
      pp.SelectNextPresetPosition()
      print(pp.GetCurrentPresetPosition)
      print("PhaseplateGetCurrentAndSelectNextAndGetCurrent done")

print("----")
PhaseplateSelectNext() print("----")
PhaseplateGetCurrent()
print("----")
PhaseplateGetCurrentAndSelectNext()
print("----")
PhaseplateGetCurrentAndSelectNextAndGetCurrent()
print("----")
print("All Done")
```

FEG Source

```
# Sample python script to show usage of FEG Source interface
# Tested with Python v3.6.8
#
import sys
import comtypes
import comtypes.client as cc

from enum import IntEnum, unique

@unique
class FegState(IntEnum):
    NOT_EMITTING = 0
    EMITTING = 1

ADV_TEM_SCRIPTING_PROG_ID = "TEMAdvancedScripting.AdvancedInstrument"

#Get the interface to the Source interface
def getSourceInterface():
    try:
        instrument = cc.CreateObject(ADV_TEM_SCRIPTING_PROG_ID)
    except(OSError):
        print("Cannot retrieve interface of {}".format(ADV_TEM_SCRIPTING_PROG_ID))
        sys.exit(1);

    if instrument is None:
        print("Invalid Instrument")
        sys.exit(1);

    return instrument.Source


if __name__ == '__main__':
    source = getSourceInterface()
    if source is None:
        print("Invalid Source")
        sys.exit(1);

    fegState = FegState(source.State)
    print("FEG State is {}".format(fegState.name))

    print("Extractor Voltage is {:.0f} V".format(source.ExtractorVoltage))

    try:
        focusindex = source.focusindex
        print("focus index is {}.{}".format(focusindex.Coarse, focusindex.Fine))
    except(comtypes.COMError):
        print("Cannot retrieve focus index")
        sys.exit(1);
```

Cold FEG Source

```
# Sample python script to show usage of Cold FEG Source interface
# Tested with Python v3.6.8
#
import sys
import comtypes
import comtypes.client as cc

from enum import IntEnum, unique

@unique
class FegState(IntEnum):
    NOT_EMITTING = 0
    EMITTING = 1

@unique
class FegFlashing(IntEnum):
    LOW_T = 0
    HIGH_T = 1

advisedStr = {True: 'Advised', False: 'NOT Advised'}

ADV_TEM_SCRIPTING_PROG_ID = "TEMAdvancedScripting.AdvancedInstrument"

#Get the interface to the Source interface
def getSourceInterface():
    try:
        instrument = cc.CreateObject(ADV_TEM_SCRIPTING_PROG_ID)
    except(OSError):
        print("Cannot retrieve interface of {}".format(ADV_TEM_SCRIPTING_PROG_ID))
        sys.exit(1);

    if instrument is None:
        print("Invalid Instrument")
        sys.exit(1);

    return instrument.Source


if __name__ == '__main__':
    source = getSourceInterface()
    if source is None:
        print("Invalid Source")
        sys.exit(1);

    fegFlashing = source.Flashing
    if fegFlashing is None:
        print("Invalid FegFlashing")
        sys.exit(1);

    isLowTFlashAdvised = fegFlashing.IsFlashingAdvised(FegFlashing.LOW_T)
```

```
    print("Low T Flashing is {}".format(advisedStr[isLowTFlashAdvised]))

    isHighTFlashAdvised = fegFlashing.IsFlashingAdvised(FegFlashing.HIGH_T)
    print("High T Flashing is {}".format(advisedStr[isHighTFlashAdvised]))

    fegState = FegState(source.State)
    print("FEG State is {}".format(fegState.name))

    if fegState == FegState.EMITTING:
        print("Perform Low T Flashing")
        fegFlashing.PerformFlashing(FegFlashing.LOW_T)

    print("Beam Current is {:.2f} nA".format(source.BeamCurrent * 1E+9))
    print("Extractor Voltage is {:.0f} V".format(source.ExtractorVoltage))

    focusIndex = source.FocusIndex
    print("Focus Index is {}.{}".format(focusIndex.Coarse, focusIndex.Fine))
```

Energy Filter

C# example EnergyFilter navigation

```
using TEMAdvancedScripting;

// Create the instrument
instrument = new TEMAdvancedScripting.Instrument();
// Access the EnergyFilter
energyFilter = instrument.EnergyFilter as TEMAdvancedScripting.EnergyFilter;

// Access the Slit, HighTensionEnergyShift and ZeroLossPeakAdjustment interfaces
slit = m_energyFilter.Slit as Slit;
htShift = m_energyFilter.HighTensionEnergyShift as HighTensionEnergyShift;
zlpAdjust = m_energyFilter.ZeroLossPeakAdjustment as ZeroLossPeakAdjustment;
```

C# example Slit movements

```
if (slit != null)
{
    // Insert the slit into the filter
    slit.Insert()
    // Should print 'Slit is inserted'
    Console.WriteLine("Slit is " + slit.IsInserted ? "inserted" : "retracted");

    slit.Retract()
    // Should print 'Slit is retracted'
    Console.WriteLine("Slit is " + slit.IsInserted ? "inserted" : "retracted");
}
```

C# example Slit Width manipulation

```
// Start with a retracted slit
slit.Retract();

// Set the slit Width to the lowest value allowed
slit.Width = slit.WidthRange.Begin;
// Should print 'Slit is retracted' (slit has not moved yet!)
Console.WriteLine("Slit is " + slit.IsInserted ? "inserted" : "retracted");
```

```
// Now Insert the Slit, and it will move to the requested Width
slit.Insert();
// Should print 'Slit is inserted'
Console.WriteLine("Slit is " + slit.IsInserted ? "inserted" : "retracted");

// Increase the slit Width by 2.0 [eV]
slit.Width = Math.Min(slit.Width + 2.0, slit.WidthRange.End);
```

C# example Slit Width error

```
// Setting out of range Width value will throw
try
{
     slit.Width = slit.WidthRange.Begin - 0.1;
}
catch (System.ArgumentException e)
{
     Console.WriteLine("Requested Width is not allowed");
}
```

C# example HighTensionEnergyShift manipulation

```
if (htShift != null)
{
    // Increase the HT EnergyShift by 1.0 [eV]
    htShift.EnergyShift = Math.Min(htShift.EnergyShift + 1.0, htShift.EnergyShiftRange.End);
}
```

C# example HighTensionEnergyShift error

```
// Setting out of range HT EnergyShift value will throw
try
{
    htShift.EnergyShift = htShift.EnergyShiftRange.Begin – 0.1;
}
catch (System.ArgumentException e)
{
     Console.WriteLine("Requested HT EnergyShift is not allowed");
}
```

C# example ZeroLossPeakAdjustment manipulation

```
if (zlpAdjust!= null)
{
    // Increase the HT EnergyShift by 2.0 [eV]
    zlpAdjust.EnergyShift =
        Math.Min(zlpAdjust.EnergyShift+2.0, zlpAdjust.EnergyShiftRange.End);
}
```

C# example ZeroLossPeakAdjustment error

```
// Setting illegal ZeroLossPeakAdjustment EnergyShift value will throw
try
{
    zlpAdjust.EnergyShift = zlpAdjust.EnergyShiftRange.Begin – 0.1;
}
```

```
catch (System.ArgumentException e)
{
     Console.WriteLine("Requested ZLP Adjustment EnergyShift is not allowed");
}
```

AutoLoader and TemperatureControl

```
# Sample python script to show usage of AutoLoader and TemperatureControl interface
# Tested with Python v3.6.8
#
# Copyright (c) 2021 by Thermo Fisher Scientific
# All rights reserved. This file includes confidential and proprietary information
of Thermo Fisher Scientific.

import comtypes
import comtypes.client as cc
import sys
import time

from enum import Enum
class SlotStatus(Enum):
    Unknown = 0
    Occupied = 1
    Empty = 2
    Error = 3

from enum import IntEnum
class ErrorCode(IntEnum):
    E_NOT_ALLOWED = -2147155969,

    # Pre-conditions:
    #   - temserver is up and running
    #   - there should be Autoloader installed
    #   - there should be a valid license for this feature
class AutoLoaderAdvScript:
    Instrument = cc.CreateObject("TEMAdvancedScripting.AdvancedInstrument")
    Autoloader = Instrument.AutoLoader
    TemperatureControl = Instrument.TemperatureControl

    def is_autoloader_installed(self):
        installed = False
        if self.Autoloader is not None:
             installed = True
        else:
            print('\n AutoLoader is not supported')
        return installed

    def is_temp_control_installed(self):
        installed = False
        if self.TemperatureControl is not None:
             installed = True
```

```
        else:
            print('\n Temperature Control is not supported')
        return installed

    def handle_com_error(self, com_error, context):
        if(int(getattr(com_error, 'hresult')) == ErrorCode.E_NOT_ALLOWED):
            print('\n {0} already performed or is not allowed.'.format(context))
        else:
            print('\n Error in {0}.'.format(context))
            print('\n Exception : ', sys.exc_info()[1])

    def get_number_of_cassette_slots(self):
        slots = 0
        try:
            slots = self.Autoloader.NumberOfCassetteSlots
            print('\n Number Of CassetteSlots : {0}'.format(slots))
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting Number Of CassetteSlots')
        return slots

    def is_slot_occupied(self, slot_number):
        slotStatus = SlotStatus.Unknown
        try:
            slotStatus = SlotStatus(self.Autoloader.SlotStatus[slot_number])
            print('\n Status of slot {0} : {1}'.format(slot_number, slotStatus))
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting slot status')
        return (slotStatus == SlotStatus.Occupied)

    def initialize(self):
        print('\n Initializing Autoloader.')
        try:
            self.Autoloader.Initialize()
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Initializing')
        else:
            print('\n Initializing Autoloader completed.')

    def dock_cassette(self):
        print('\n Docking cassette in Autoloader.')
        try:
            self.Autoloader.DockCassette()
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Docking cassette')
        else:
            print('\n Docking cassette in Autoloader completed.')

    def perform_inventory(self):
        print('\n Inventorying the cassette.')
        try:
            self.Autoloader.PerformCassetteInventory()
```

```
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Inventorying cassette')
        else:
            print('\n Inventorying the cassette completed.')

    def load_cartridge(self, slot_number):
        print('\n Loading cartridge from slot : {0}'.format(slot_number))
        try:
            self.Autoloader.LoadCartridge(slot_number)
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Loading cartridge')
        else:
            print('\n Loading cartridge from slot : {0}
completed.'.format(slot_number))

    def unload_cartridge(self):
        print('\n Unloading cartridge from stage.')
        try:
            self.Autoloader.UnloadCartridge()
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Unloading cartridge')
        else:
            print('\n Unloading cartridge from stage completed.')

    def get_docker_temperature(self):
        temp = 0.0
        try:
            temp = self.TemperatureControl.AutoloaderCompartment.DockerTemperature
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting Docker temperature')
        else:
            print('\n Docker Temperature : {0} K'.format(temp))
        return temp

    def get_cassette_gripper_temperature(self):
        temp = 0.0
        try:
            temp = self.TemperatureControl.AutoloaderCompartment.CassetteTemperature
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting Cassette Gripper temperature')
        else:
            print('\n Cassette Gripper Temperature : {0} K'.format(temp))
        return temp

    def get_cartridge_gripper_temperature(self):
        temp = 0.0
        try:
            temp =
self.TemperatureControl.AutoloaderCompartment.CartridgeTemperature
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting Cartridge Gripper
temperature')
```

```
        else:
            print('\n Cartridge Gripper Temperature : {0} K'.format(temp))
        return temp

    def get_holder_temperature(self):
        temp = 0.0
        try:
            temp = self.TemperatureControl.ColumnCompartment.HolderTemperature
        except comtypes.COMError as com_error:
            self.handle_com_error(com_error, 'Getting Holder temperature')
        else:
            print('\n Holder Temperature : {0} K'.format(temp))
        return temp

def wait_for_docking_temperatures():
    # This time out value is purely indicative.
    # On the actual system it can take much longer to go to cryo temperatures.
    timeout = 10
    while((script_al.get_docker_temperature() > DOCKER_TEMP or
           script_al.get_cassette_gripper_temperature() > CASSETTE_TEMP) and
          timeout > 0):
        time.sleep(1)
        timeout -= 1
    handle_timeout(timeout, 'docking')

def wait_for_inventory_temperatures():
    # This time out value is purely indicative.
    # On the actual system it can take much longer to go to cryo temperatures.
    timeout = 10
    while(script_al.get_cartridge_gripper_temperature() > CARTRIDGE_TEMP and timeout
> 0):
        time.sleep(1)
        timeout -= 1
    handle_timeout(timeout, 'inventory')

def wait_for_loading_temperatures():
    # This time out value is purely indicative.
    # On the actual system it can take much longer to go to cryo temperatures.
    timeout = 10
    while((script_al.get_cartridge_gripper_temperature() > CARTRIDGE_TEMP or
script_al.get_holder_temperature() > HOLDER_TEMP) and timeout > 0):
        time.sleep(1)
        timeout -= 1
    handle_timeout(timeout, 'loading')

def handle_timeout(timeout, context):
    if(timeout == 0):
        print('\n Timeout occured while waiting for temperatures before
{0}.'.format(context))
        sys.exit()
    else:
        print('\n Temperatures for {0} reached.'.format(context))
```

```
if __name__ == '__main__':
    print('PreConditions for this script to run')
    print(' - temserver is up and running')
    print(' - there should be Autoloader installed')
    print(' - there should be a valid license (Scr. Sample Loading) for this
feature')
    print(' - there should be a capsule placed with a few cartridges')
    slot = int(input('\n Enter slot number to load : '))
    script_al = AutoLoaderAdvScript()
    DOCKER_TEMP = 100 # Kelvin
    CASSETTE_TEMP = 100 # Kelvin
    CARTRIDGE_TEMP = 100 # Kelvin
    HOLDER_TEMP = 100 # Kelvin
    if(script_al.is_autoloader_installed() and
script_al.is_temp_control_installed()):
        if(slot<=script_al.get_number_of_cassette_slots() and slot > 0):
            print('\n Valid slot number provided : {0}'.format(slot))
            # 1. Initialize the Autoloader
            script_al.initialize()
            # 2. Wait until the required temperatures are reached for docking.
            wait_for_docking_temperatures()
            # 2. Dock the cassette into the Autoloader.
            script_al.dock_cassette()
            # 3. Wait until the required temperatures are reached for inventory.
            wait_for_inventory_temperatures()
            # 4. Perform Inventory
            script_al.perform_inventory()
            # 5. Check the slot is occupied
            if(script_al.is_slot_occupied(slot)):
                #6. Wait until the required temperatures are reached for loading.
                wait_for_loading_temperatures()
                #6. Load the Sample on the stage
                script_al.load_cartridge(slot)
                #7. Unload the Sample from the stage
                script_al.unload_cartridge()
            else:
                print('\n Invalid slot specified as slot is not occupied.')
        else:
            print('\n Invalid slot specified')
    else:
        print('\n Autoloader or Temperature Control not supported.')
```