CAS Spectrometer Driver

SDK

User Manual

4.10.2

Instrument Systems Optische Messtechnik GmbH
Kastenbauerstr. 2
D-81677 Munich, Germany
www.instrumentsystems.com
07/2020

Phone: +(49) 89-454943- 0
Fax: +(49) 89-454943-11
webinfo@instrumentsystems.com

Chapter 1

Introduction

This manual provides information about the functions and procedures of the CAS DLL.

Note

> The documentation uses the static .NET class CAS4DLL to provide a reference for all methods and constants provided and used by the CAS DLL.

As a first step you should use the program CASDLLTEST.EXE to test if your spectrometer works and to get a first impression of the capabilities of the CAS DLL.

The manual is complemented by the examples provided in the subdirectories of the DLL installation directory. Here you find examples for different programming languages:

- A fully functional example written with C++ using *Microsoft Visual C++ 2013* can be found under *VC2013*. You will find the example source code, the CAS4.h header and the lib files for Win32 and Win64.
- In the subdirectory *CSharp* you find the C# solution CAS4DLLTest which uses the class CAS4DLL.
- The subdirectory *Delphi* contains a Delphi sample project that demonstrates basic usage of the CAS DLL.

The DLL itself is installed to the system directory (usually C:\Windows\System32 or C:\Windows\SysWOW64 on Win64 systems).

The lib and header files for the different programming languages are in the directories of their respective examples (see above). If you are missing lib files for other compilers, use their toolchain to generate them from the DLL itself. One example would be C++ Builder, where you use implib.exe for Win32 and mkexp.exe for Win64.

Note

> As a special feature the CAS DLL provides a demo mode interface. This interface enables you to test your application without having a spectrometer connected to your PC. The demo mode interface creates artificial noisy sinus spectra.

Chapter 2

To-Do List

This chapter gives an overview about the important topics that should be covered when integration the CAS DLL. It is intended as a check list that can be used to ensure that no vital aspect of the CAS DLL integration are forgotten.

Essential

Check	Chapter in this manual
	Device Handles and Interface Types
	Device Parameter and Measurement Parameter
	Thread Safety
	Error Handling
	Configuration and Initialization
	Density Filter
	Dark Current
	Performing a Measurement
	Getting the Spectrum and Results

Recommended

Check	Chapter in this manual
	Using Serial Numbers
	CCD Temperature Monitoring
	Synchronization

Optional

Check	Chapter in this manual
	Triggered Measurements
	AutoRange Measurements
	Working with Parameter Sets
	MultiTrack Measurements
	Transmission Measurement

Chapter 3

What has changed

This topic lists major changes that have been made to the interface.

casPerformActionEx

A new methode casPerformActionEx has been introduced to support actions which required additional parameter. The old casPerformAction method has been marked deprecated and should no longer be used for new code and new actions.

If you are porting an application which previously used CAS DLL version 3.4 or earlier there are a few things to consider:

Integer Types

Throughout the DLL interface now only one 32bit integer type is used: int in C, Long in Visual Basic and Integer in Delphi. Note that this also applies to the 64bit version of the DLL which is called CAS4x64.DLL on Windows.

Unified Methods for Accessing Device and Measurement Parameter

Instead of introducing a new set of get- and set-methods for every new measurement or device parameter, generic methods have been introduced. See chapters Device Parameter and Measurement Parameter.

Enhanced Support for Triggered Measurements

The CAS DLL can now return the triggering capabilities of the device - refer to the dpidTriggerCapabilities device parameter. If supported by the device, the current state of the acquisition now can also be signaled via DigitalOut ports - refer to the mpidTriggerOptions measurement parameter.

Chapter 4

Tasks

4.1 Interfaces Types and Options

Two basic aspects of a spectrometer instance within the DLL are the interface type and its option. The interface type defines how the spectrometer is connected to the computer and therefore by which means it is accessed (USB, PCI card etc.). For most interface types, the interface option defines a specific spectrometer that is connected to the PC with this interface type. However, for some interface types like PCI, the interface option only identifies the hardware interface card, not a specific spectrometer.

4.1.1 Device discovery

For the traditional interface types like PCI, PCIe and USB, connected spectrometers are automatically discovered in a synchronous fashion that happens implicitly when the interface options are enumerated. However, some interface types, like Ethernet, can only be discovered asynchronously. This discovery process can be initiated by calling paSearchForDevices. Typically this is called very early during startup of your application and when new devices should be listed, for example whenever the user is presented with a user interface to select interface type and option as described below.

4.1.2 Letting the user select interface type and option

Typically the user is presented with two drop-down boxes where the interface type is selected first and then the available options are listed in the second drop-down.

Retrieve the number of interfaces with casGetDeviceTypes and their display names with casGetDeviceTypeName. Ignore interface types with an empty display name, as they are unavailable. The number of options for a given interface is returned by casGetDeviceTypeOptions, the value of the actual option with casGetDeviceTypeOption and the display name with casGetDeviceTypeOptionName.

To get the interface type and option for a given CASID use the dpidInterfaceType and dpidInterfaceOption device parameter. If the device associated with a given interface type and option is not available, methods like casInitialize will return ErrorAdrControl.

Typically the interface type and option properties are stored and restored as part of your applications settings, so that you can start working with the device right away after restarting.

To change the interface type and or option call casChangeDevice.

4.1.3 Solutions without a UI

Some interface type constants are considered constant and it is not very likely that they change in future versions, the most prominent example beeing InterfaceUSB. So to work with the first available USB spectrometer, you could retrieve the first available interface option for InterfaceUSB create a CAS ID for it with the following code:

[Delphi]

```
//before getting the interface option with index 0, ensure that the
//interface has enough options
intOptionCount = casGetDeviceTypeOptions(InterfaceUSB);
CheckError(intOptionCount);
if intOptionCount = 0 then
  raise Exception.Create('No interface options found for USB');
intOption:= casGetDeviceTypeOption(InterfaceUSB, 0);
...
CASID:= casCreateDeviceEx(InterfaceUSB, intOption);
```

Note

This approach will not work under all circumstances. It also does not detect if a different spectrometer has been connected since the last start (which would need other calibration files etc.). The best solution is still to produce an UI where the user can select interface type and option and then store and restore these values in configuration settings.

Warning

Be aware that interface options are not identical to spectrometers in all cases (e.g. for PCI) and that interfaces might also contain options for spectrometers that are no longer connected to the PC! This can have unexpected consequences when using the approach outlined above.

4.2 Device Handles (a.k.a. CASIDs) and Interfaces Types

When working with the CAS DLL you create objects within the DLL which store information about all aspects of the spectrometer, including hardware information as well as measurement parameters, the measured spectrum etc. These objects are identified via a device handle, which is often called a CASID.

Device Creation and Destruction

A device handle is typically created with casCreateDeviceEx where interface type and option are passed (see chapter Interfaces Types and Options).

Once you no longer need access to the spectrometer, you release the device handle with casDoneDevice.

4.3 Thread Safety

Warning

The CAS DLL is not thread safe. Applications using the DLL must ensure that no two threads call a D↩ LL method at the same time! Otherwise device operation might cause unpredicted behavior and/or result in unusual errors!

Typically thread-safety is ensured by the application with protecting every DLL method call by acquiring a global critical section, but any similar synchronization mechanism can be used.

Since every device handle also stores an error code, which is reset at the beginning of each method call, the error handling must also be done while the critical section is acquired.

4.3.1 Global settings

There are a few settings which are stored globally and therefore affect all devices. This might cause extra trouble in a multi-threaded application which operates several devices. These settings are:

- mpidCRIMode used when calling casCalculateCRI
- mpidObserver used when calling casColorMetric, casCalculateCRI with criCIE13_3_95, cmXYToDominantWavelength and casGetCCT

If these settings really need to be changed while several devices are operated, one possible approach would be to protect the above mentioned calls with a separate critical section to avoid that they are called simultaneously.

4.4 Error Handling

Most methods directly return an error code. For those who do not, the method casGetError can be called directly afterwards.

This error code can be translated into a error message by calling casGetErrorMessage.

The following code contains a generic error handling method in Delphi which will raise an exception in case of an error. Different usage scenarios are also shown.

[Delphi]

```
procedure CheckError(AErrorCode: Integer);
var
  ach: array[0..255] of AnsiChar;
begin
  if AErrorCode < ErrorNoError then
  begin
    casGetErrorMessage(AErrorCode, ach, 255);
    raise ECASDLLException.CreateFmt('CAS DLL Error (%d): %s', [AErrorCode, StrPas(ach)]);
  end;
end;

...

//Usage with return value:
CheckError(casMeasureDarkCurrent(CasID));

//Usage with casGetError:
factor:= casGetCalibrationFactors(CasID, 25, 36);
CheckError(casGetError(CasID));

//Usage with casGetError too because return value cannot be considered an error code under all
       circumstances
triggerOptions:= Round(casGetMeasurementParameter(CasID, mpidTriggerOptions));
CheckError(casGetError(CasID));
```

4.5 Device Parameter (dpid...)

Device parameter are properties or settings of the device and generally things which are not affected by or changed between measurements.

There is a set of four methods to set and retrieve device dependent parameter for a given spectrometer, two for numerical parameter and two for string parameter. The AWhat parameter defines the device parameter which should be retrieved or set. The possible values for AWhat are constants starting with dpid.

- casGetDeviceParameter
- casSetDeviceParameter
- casGetDeviceParameterString
- casSetDeviceParameterString

[Ansi C]

```
char buf[256];
int ret = casGetDeviceParameterString(CASID, dpidConfigFileName, buf, 255);
if (ret != ErrorNoError)
  goto fail;

...

char buf[256];
buf = ...
int ret = casSetDeviceParameterString(CASID, dpidConfigFileName, buf);
if (ret != ErrorNoError)
  goto fail;
```

4.6 Measurement Parameter (mpid...)

There is a set of two methods to set and retrieve numerical measurement parameter. These are either settings on how the device should measure, but can also document the measurement conditions.

- casGetMeasurementParameter
- casSetMeasurementParameter

Warning

The error handling shown in the example below is only possible, because mpidIntegrationTime is defined as a non-negative integer. With other mpids, it is necessary to call casGetError separately, to perform error checking, because the return value cannot safely be interpreted as an error code! For example mpidFlash↩ DelayTime might return -14 as a valid negative delay time of 14ms and does not indicate ErrorAdrControl!

[Ansi C]

```
double ret = casGetMeasurementParameter(CASID, mpidIntegrationTime);
long intTime = lround(ret);
if (intTime < ErrorNoError)
  goto fail;
```

4.7 Configuration and Initialization

After setting up the device type and option it is essential that the device is configured. At the very least a configuration file (.ini) and a calibration file (.isc) have to be provided with the dpidConfigFileName and dpidCalibFileName device parameter. These files typically come in pairs with the same name.

Some devices have configuration and calibration files stored on the device, see chapter Working with Calibration files on the device.

When the configuration and calibration files are set, the device can be initialized with casInitialize. Measurement parameter should only be set after casInitialize has been called, since only then the configuration has been loaded, which might affect parameter ranges etc.

4.8 Working with Calibration files on the device

You can download the files using dpidGetFilesFromDevice. The path you set this dpid to should be an empty directory where the files will be downloaded to. If the device does not have files or doesn't support this at all, dpidGetFilesFromDevice will return errors like ErrorInvalidEEPromType or ErrorNoFilesOnIdentKey.

Each downloaded calibration will consist of a configuration file (.ini), a calibration file (.isc) and possibly additional files. These files need to be used for Configuration and Initialization. The chapter Using Serial Numbers describes techniques to ensure that the correct calibration is used together with the correct accessory.

4.8.1 Checking for updated calibrations

If a device is recalibrated, it is necessary to redownload the updated calibration files. To check whether the device has newer calibration files, the CAS library supports 2 methods:

- the check is performed during casInitialize, if the option coCheckConfigUpToDate is enabled. Should an updated calibration be available, casInitialize would return ErrorConfigUpToDate
- you can perform the check manually at any time which is suitable, after the device has been initialized, by calling paCheckConfigUpToDate, which will also return ErrorConfigUpToDate, if there's an updated calibration

4.9 Density Filter

The density filter which should be used for the next measurement is defined by mpidNewDensityFilter. Before every measurement, it is essential, that you check the dpidNeedDensityFilterChange device parameter. If it returns true mpidDensityFilter should be set to mpidNewDensityFilter which will turn the filter wheel to the correct position.

[Visual Basic]

```
Public Function MeasureCAS(ByVal ACasID As Integer)

  If casGetDeviceParameter(ACasID, dpidNeedDensityFilterChange) <> 0 Then
    Call casSetMeasurementParameter(ACasID, mpidDensityFilter, casGetMeasurementParameter(ACasID,
       mpidNewDensityFilter))
  End If

  ...

  MeasureCAS = casMeasure(ACasID)
End Function
```

Note

This ensures that all filter options and parameter are taken into account and that the filter wheel is not turned within a measurement cycle, which might interfere with timing restrictions or even destroy the DUT.

If you want to cater for spectrometers without a filter wheel, check whether the configuration supports a manual density filter. This corresponds to dpidFilterType = 1. In this case your application should display a user message when mpidDensityFilter is changed (see example above). You might want to use casGetFilterName to display a user-friendly name for the filter.

Monitoring Filterwheel and Shutter operation

To determine if a given device supports shutter and/or filterwheel monitoring, use casGetOptions and check for the coGetFilter and coGetShutter bits.

Should shutter or filterwheel operation fail, the following actions may signal an error either via their return value or by setting error information (see casGetError).

- reading mpidCurrentDensityFilter
- setting mpidDensityFilter
- casGetShutter
- casSetShutter
- casMeasure
- casMeasureDarkCurrent

If you want to check for shutter and filter wheel errors at any given time, read mpidCurrentDensityFilter or call casGetShutter.

The ForceFilter option is automatically enabled for devices which support shutter/filter-monitoring. This causes dpidNeedDensityFilterChange to always return true, so the filter should be set before each measurement (see section above).

4.10 Dark Current

Before a spectrum can be measured, it is necessary to measure the dark current (or DC) so it can be subtracted from the spectra later. This is done with casMeasureDarkCurrent. Changing the integration time and or averages might make it necessary to measure a new dark current. To find out whether this is the case, check dpidDCRemeasure←Reasons. You can also set up a time after which a DC is marked invalid. Refer to the coAutoRemeasureDC option (see casGetOptions) and mpidRemeasureDCInterval.

Use casSetShutter to close the shutter before calling casMeasureDarkCurrent and open it afterwards. If you want to display a user message for devices without a shutter, check the coShutter option with casGetOptions.

Warning

It is absolutely *essential* to perform proper error handling for any of the above method calls!

casMeasureDarkCurrent only measures the dark current for the active parameter set. paMeasureParamSetsDC provides a convenient and efficient way to measure the dark current for some or all parameter sets - after closing the shutter and before opening it as described above. Refer to chapter Working with Parameter Sets for more information.

Warning

Before the dark current measurement is performed, it is necessary to move the filterwheel to the correct position! This helps avoiding overexposure for spectrometers where the filter wheel also functions as a shutter. For more information refer to the chapter Density Filter.
It is also essential that at least one dark current measurement has been performed, before any spectrum is measured. Otherwise the resulting spectrum would be incorrect, one reason being that the AmpOffset is determined during a dark current measurement.

An existing dark current can be removed by calling casClearDarkCurrent, but that is usually not necessary. Note that clearing the dark current will not clear the AmpOffset mentioned above.

4.10.1 Dark current array

A dark current array is supported for all device types. It contains the dark current spectra for various integration times which are measured at once, so that switching to a different integration time does not require a new dark current measurement. This is especially useful for AutoRange measurements and for spectrometers without a shutter.

Note

With most spectrometers now having a shutter and with introduction of a new AutoRange variant which requires less DC measurements (see AutoRange for various intensities), the dark current array has less and less advantages. With the introduction of the new paMeasureParamSetsDC (refer to the first part of this chapter), there is now also no longer a need for using the dark current array, even if you have a large number of parameter sets.

The dark current for a given integration time is interpolated from the dark current array. To activate the dark current array, set the coUseDarkcurrentArray option (see casGetOptions). Use casGetCalibrationFactors to access the actual dark current array. Measuring a dark current array is done the same way as a normal dark current measurement: close the shutter, call casMeasureDarkCurrent and then open the shutter.

Warning

In rare cases, measuring a dark current array can result in ErrorDarkArray. This indicates that one of the dark current array spectra could not be measured with exactly the desired integration time, resulting in an invalid array. In such cases the dark current array can not be used and coUseDarkcurrentArray should be switched off.

Note

If the dark current array has been successfully measured, dpidDCRemeasureReasons will no longer signal a required dark current measurement when changing the integration time or switching the current parameter set. This might however still happen, if the array hasn't been measured or the range of the dark current array does not cover the integration time to be measured.

Warning

Before the dark current measurement is performed, it is necessary to move the filterwheel to the correct position. This helps avoiding overexposure for spectrometers where the filter wheel also functions as a shutter. For more information refer to the chapter Density Filter.

4.11 Performing a Measurement

A spectrum can be measured by calling casMeasure. This method only returns after the acquisition has been completed.

The following preconditions should be met before casMeasure is called:

- The device handle has been created, see Device Handles (a.k.a. CASIDs) and Interfaces Types
- The device has been configured and initialized, see Configuration and Initialization
- The integration time has been set, see mpidIntegrationTime, or AutoRange has been enabled
- The density filter wheel has been moved if necessary, i.e. setting mpidDensityFilter to mpidNewDensityFilter if dpidNeedDensityFilterChange returns true.
- The dark current has been measured, if dpidDCRemeasureReasons is not 0. Refer to chapter Dark Current
- casPerformActionEx has been called with paPrepareMeasurement

A lot of measurement setups require that the CAS is synchronized with other equipment like a handler or a sourcemeter. For an overview of possible synchronization setups, see chapter Synchronization.

Warning

Validate the measurement by checking that the spectrometer wasn't oversaturated: mpidMaxADCValue must not be larger than dpidADCRange. If it reaches dpidADCRange, this should be considered a hard error. Another recommended validation is to check mpidRelSaturation against dpidRelSaturationMin and dpidRelSaturationMax to make sure it was sufficiently saturated. If this check fails, it does not have to be considered a hard error, but rather a good/bad indicator.
Note that checking mpidRelSaturation alone is not sufficient to check for over-saturation, especially when avering is used!
Do *not* calculate a relative signal level yourself using mpidMaxADCValue as this will be a misleading indicator! mpidRelSaturation takes the DC into account and is therefore better suited.
Obviously error handling is essential to detect hardware failures and other problems. Without error checking there would be no indication that the measurement failed and the previously measured spectrum would be used.
It is highly recommended to retrieve all measurement parameter after the measurement and document them with your measurement data, because some measurement parameter might have been adjusted. The most prominent example would be mpidIntegrationTime which is affected by AutoRange, mpidIntTimeAlignPeriod, mpidIntTimeResolution and for some spectrometers also dpidIntTimeMax.

4.12 Getting the Spectrum and Results

After a spectrum has been measured, the intensity values can be retrieved by calling casGetData for every visible pixel. The data unit for these values can be retrieved with dpidCalibrationUnit once casInitialize has been called. casGetXArray provides the corresponding wavelength values and casGetDarkCurrent returns the dark current per pixel. Make sure that dpidDeadPixels and dpidVisiblePixels are taken into account, i.e. start with dpidDeadPixels and end with dpidDeadPixels + dpidVisiblePixels - 1 (since the pixel index is 0-based).

4.12.1 ColorMetric results

The CAS SDK can calculate a wide range of colormetric results from a previously measured spectrum. These calculations only use the part of the spectrum defined by the mpidColormetricStart and mpidColormetricStop wavelength range.

Warning

The default values for mpidColormetricStart and mpidColormetricStop always have been 380nm and 780nm. It is highly recommended to adjust these values as they might constrain the calculation unexpectedly. Set both of these to zero to use the full range of the spectrum for calculation.

Starting with version 4.10 of the CAS SDK, a new mpidColormetricType parameter has been introduced. It allows to optionally switch to the more accurate but also more time-consuming colormetric calculation of SpecWin Pro. mpidColormetricType only applies to the given CAS ID but is independent of its active parameter set.

casColorMetric performs a calculation of several colormetric results from the measured spectrum which can then be retrieved with the following methods.

The photometric and radiometric integrals can be retrieved with casGetPhotInt and casGetRadInt.

casGetCentroid returns the centroid wavelength whereas casGetPeak and casGetWidth return results related to the peak.

Use casGetColorCoordinates to retrieve the CIE color coordinates and casCalculateLambdaDom to calculate dominant wavelength and purity.

To get the correlated color temperature (CCT) call casGetCCT and to get the tristimulus values call casGetTriStimulus.

Further results can be retrieved with casGetExtendedColorValues, which include the VIS, UVA, UVB and UVC integrals.

If you need the color rendering indices (CRI), call casCalculateCRI to calculate them and casGetCRI to retrieve them. Note that the mode of the CRI calculation can be set globally with mpidCRIMode.

4.13 Synchronization

In most measurement setups some kind of synchronization is needed to ensure that the CAS measurement takes place at the right point in time. Most likely because of a sourcemeter or handler powering a DUT.

Note

For reliable measurement results, it is essential that the CAS measurement happens only after the DUT is powered on and that the DUT is powered on long enough.

The following table lists possible synchronization methods with their advantages and disadvantages. It starts with the ideal but challenging method and ends with the easiest but most problematic method.

Method	Who triggers whom	Advantages	Disadvantages
• trigger handshake recommended!	• CAS triggers sourcemeter start • Sourcemeter triggers CAS start • optional: CAS triggers sourcemeter power-off	• most accurate timings and therefore results • minimal cycle time • minimal power-on time for DUT • trigger delay and settling time possible	• challenging requirements
• triggered CAS might be OK	• sourcemeter triggers CAS	• easier to set up	• CAS might not be ready for trigger[1] • longer cycle time • risk of DUT powering off too early
• software triggering not recommended	• no triggers	• easiest to set up	• longest cycle time • longest power-on time for DUT • risk of DUT damage [2] • a varying settling time is unavoidable and might influence results

1. It might be OK to work around this by adding a generous delay before the sourcemeter triggers the CAS. Some CAS types also support the toAcceptOnlyWhenReady trigger option which would at least help detect a missed trigger.

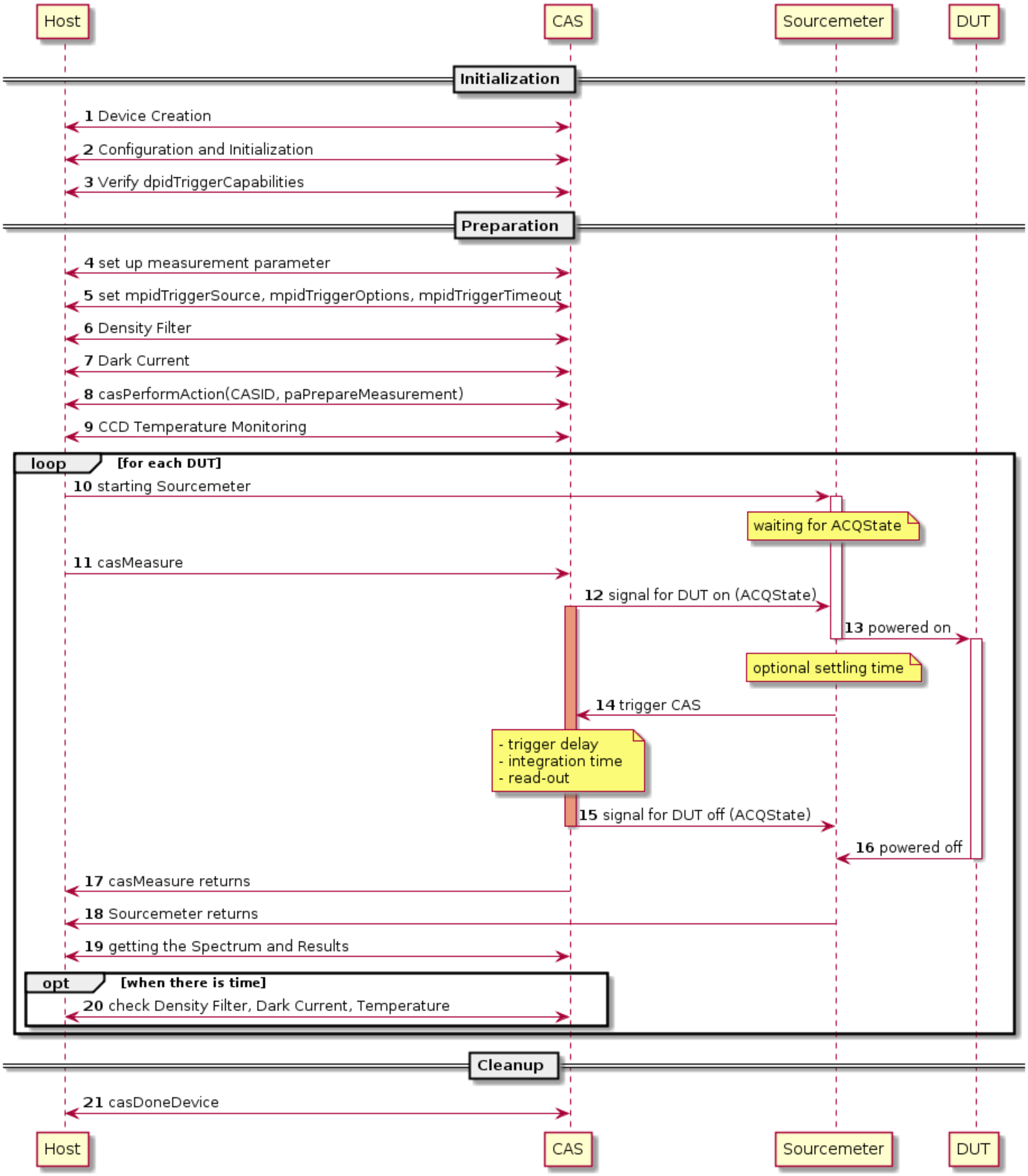
2. Risk of DUT damage: if preparation of measurement has not been done properly or if error handling is missing, the DUT might be powered on for too long, which might cause damage.

4.14 Trigger handshake sequence

The following diagram illustrates a full trigger handshake sequence as described in the synchronization chapter.

Most steps in the sequence refer to chapters in this documentation with the same name.

See chapter triggered measurements for details on trigger options and capabilities.



4.14.1 For this sequence to work

- the dpidTriggerCapabilities **has to** include tcoCanTrigger

- similarly the toShowACQState bit of mpidTriggerOptions has to be set, toAcceptOnlyWhenReady is optional but recommended
- the ACQState line settings should be verified: by checking mpidACQStateLine and mpidACQStateLine↩ Polarity. **Not all CAS types support modifying these mpid's!**

4.14.2 Several measurements per DUT

- if several measurements per DUT are needed, use parameter sets
- for each parameter set, set dpidCurrentParamSet and then call casMeasure (step 11 to 17)
- after all measurements have been done, again for each parameter set, set dpidCurrentParamSet and then get spectrum and results (step 19)
- whether you need to start the sourcemeter for each parameter set depends on the sourcemeter and it's capabilities
- note that the parameter sets need to be defined together with setting the measurement parameter (step 4)
- the Dark Current also needs to be measured for all parameter sets

Remarks

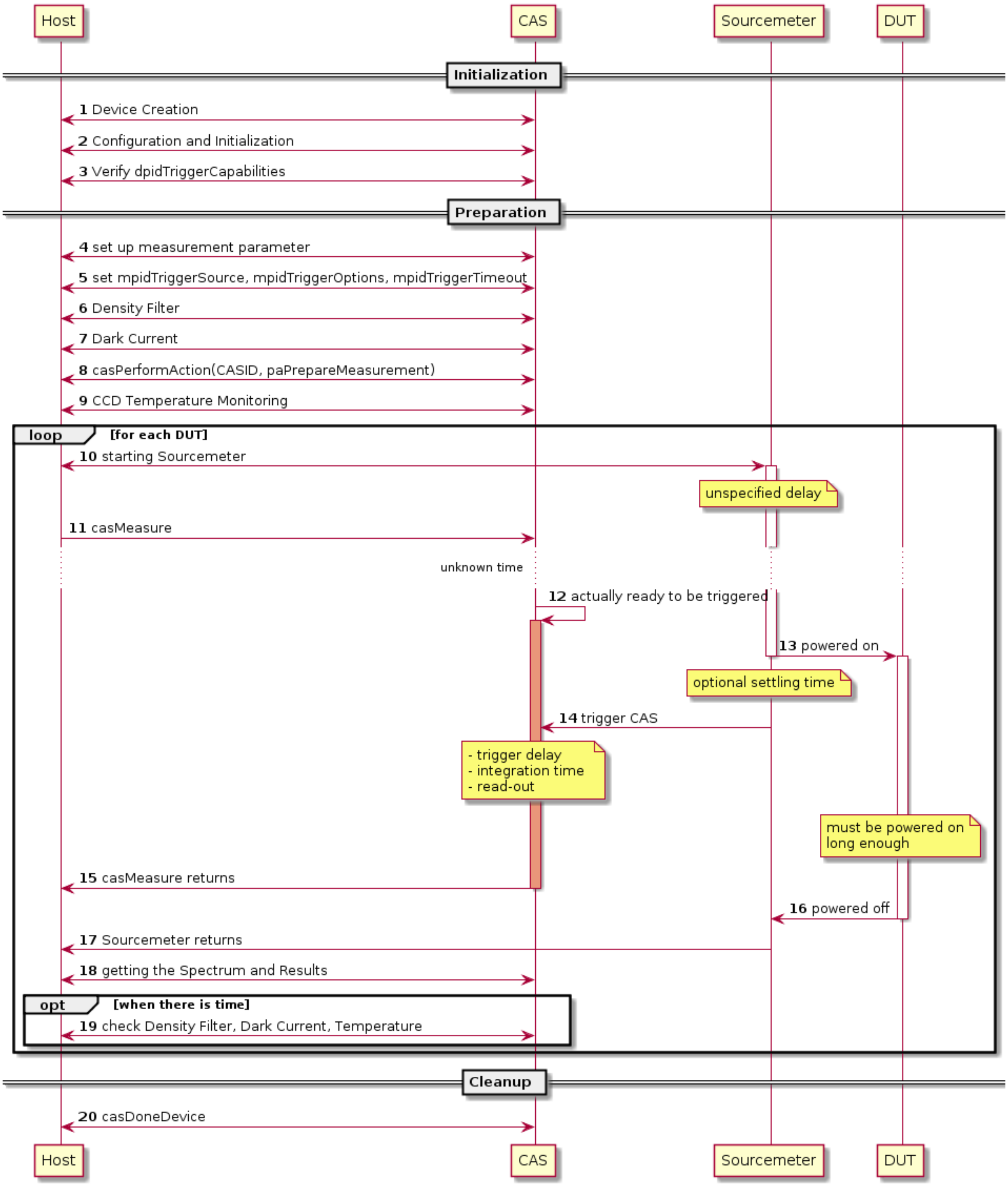
- if the sourcemeter does not support being switched off by the CAS (step 15 in the sequence diagram), a possible workaround is to switch off the DUT after a given time. This time - which should be verified by tests - should cover mpidIntegrationTime $*$ mpidAverages plus a few milliseconds extra time. The same approach is taken in the trigger CAS sequence.
- in the sequence diagram, the red rectangle on the CAS line shows the time during which the ACQ State line has the polarity defined by mpidACQStateLinePolarity. As soon as the DUT can be switched off, it changes back to the opposite polarity.

4.15 Trigger CAS sequence

The following diagram illustrates the trigger CAS sequence as described in the synchronization chapter.

Most steps in the sequence refer to chapters in this documentation with the same name.

See chapter triggered measurements for details on trigger options and capabilities.



4.15.1 For this sequence to work

- the dpidTriggerCapabilities **has to** include tcoCanTrigger
- the toAcceptOnlyWhenReady bit of mpidTriggerOptions should be set, but some CAS types do enforce this bit automatically. It is recommended to verify this bit afterwards by reading mpidTriggerOptions

4.15.2 Several measurements per DUT

- if several measurements per DUT are needed, use parameter sets

- for each parameter set, set dpidCurrentParamSet and then call casMeasure (step 11 to 15)

- after all measurements have been done, again for each parameter set, set dpidCurrentParamSet and then get spectrum and results (step 18)

- whether you need to start the sourcemeter for each parameter set depends on the sourcemeter and it's capabilities

- note that the parameter sets need to be defined together with setting the measurement parameter (step 4)

- the Dark Current also needs to be measured for all parameter sets

Remarks

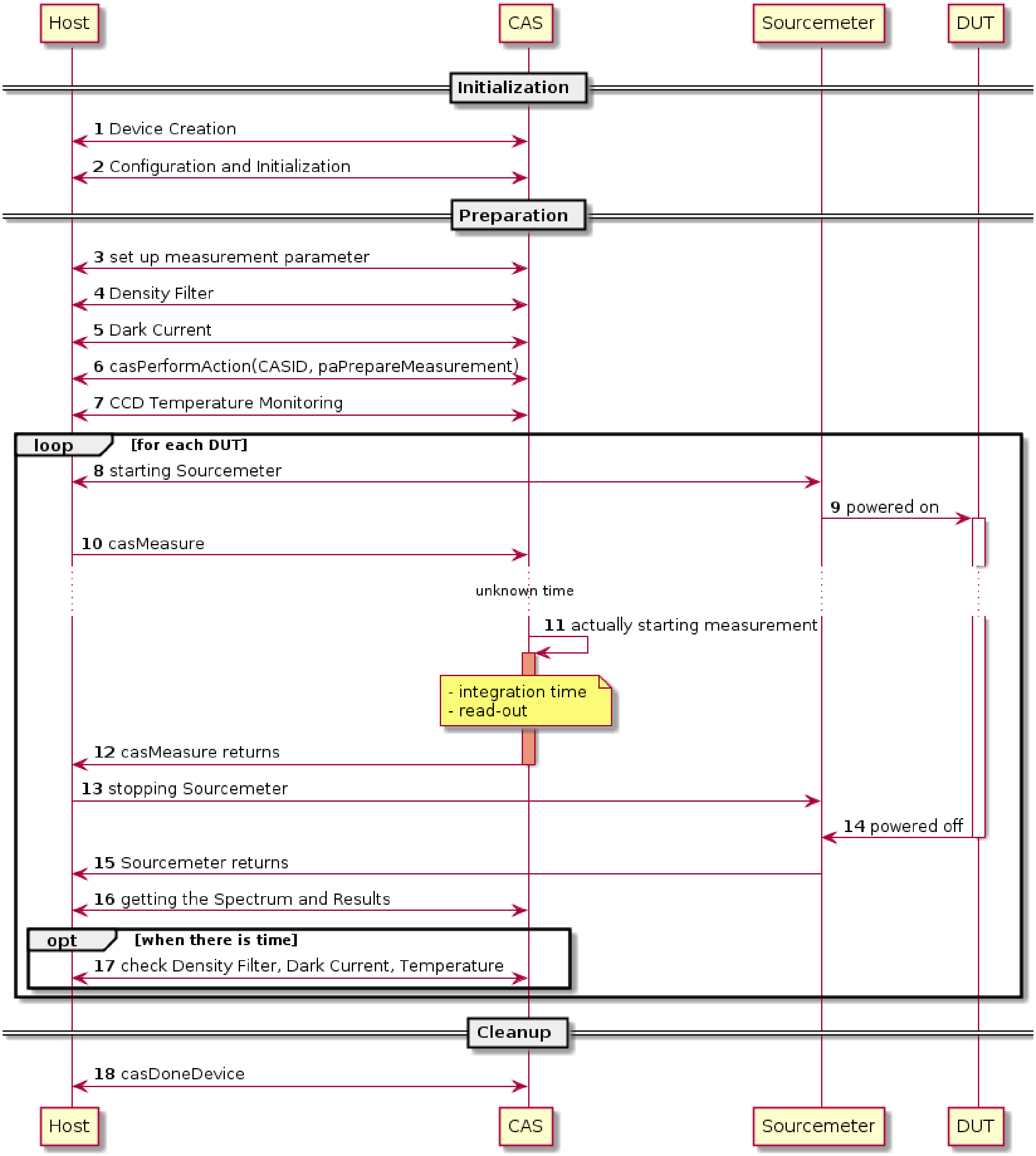
- the Sourcemeter delay time (between steps 10 and 11 in the diagram) should be verified by tests. Increase it until you no longer get missed triggers, i.e. ErrorTriggerTimeout.

- the DUT power on time (between steps 13 and 16) should also be verified by tests. Start with mpidAutoFlowTime as this covers mpidIntegrationTime $*$ mpidAverages. You might need to add some time until it no longer influences CAS measurement results.

4.16 Software trigger sequence

The following diagram illustrates the software trigger sequence as described in the synchronization chapter.

Most steps in the sequence refer to chapters in this documentation with the same name.



Remarks

- proper preparation of the measurement (steps 4 to 7 in the diagram) is even more essential here, because an incomplete preparation could extend the unknown time between steps 10 and 11 significantly
- to avoid the risk of DUT damage because of a longer than expected power on time, the DUT could be switched off after a given time, similar to the approach shown in the trigger CAS sequence. But this power on time now also needs to cover the unknown time before measurement starts (between steps 9 and 10), so it must be chosen and verified carefully.

4.17 Triggered Measurements

Triggered measurements using the CAS DLL are done by setting the mpidTriggerSource parameter to trgFlipFlop. A delay between trigger and acquisition can be set with mpidTriggerDelayTime. The trigger timeout should be set appropriately with mpidTriggerTimeout.

Some spectrometer types support signaling their state via DigitalOut ports which can help triggering the current source - see mpidTriggerOptions and the Synchronization chapter.

Warning

For some interface types a call to casPerformActionEx with paPrepareMeasurement is essential for triggered measurements to work!
Make sure to test whether the spectrometer supports the trigger aspects you want to use! Most importantly check the dpidTriggerCapabilities, but also things like dpidTriggerDelayTimeMin and dpidTriggerDelayTimeMax.
Be aware that the DigitalOut ports can change their state during the preparation phase! For example, as soon as the tcoShowACQState trigger option is activated, the mpidACQStateLine reflects the current state of mpidACQStateLinePolarity. So if the option is activated before the polarity is set, the level will change, which might result in an unwanted trigger. The safest approach is to only arm a sourcemeter triggered by the CAS *after* all the preparation tasks have been completed. This is especially important since dark current measurements could also result in unwanted triggers.

The following example checks if the device supports signaling the ACQ state via DigitalOut ports:

[Visual Basic]

```
Dim Capabilities As Long

Capabilities = Round(casGetDeviceParameter(CASID, dpidTriggerCapabilities))

If (Capabilities And tcoShowACQState) <> 0 Then
  MsgBox "Device supports signaling ACQ state"
Else
  MsgBox "Device does NOT support signaling ACQ state"
End If
```

4.18 AutoRange measurements

For some measurement tasks it might be appropriate to adjust the integration time of the CAS automatically. The CAS Library provides an AutoRange feature that determines the proper integration time and optionally can also change to the appropriate density filter, if necessary. After the AutoRange measurement, these parameter might have been changed and should be retrieved to document the measurement conditions.

Starting with version 4.7 the CAS Library supports two different ways of AutoRange. The one which has been supported for many years now, is aimed at measuring samples with similar intensities. The new AutoRange method is more suitable for various intensities and can significantly reduce the time and number of measurements which have to be made before suitable settings are found.

4.18.1 AutoRange parameter

Use mpidAutoRangeMinLevel, mpidAutoRangeMaxLevel and mpidAutoRangeMaxIntTime to specify limits for AutoRange. You can query dpidAutoRangeFilterMin and dpidAutoRangeFilterMax to check the density filter range which will possibly be used.

Warning

mpidAutoRangeMinLevel and mpidAutoRangeMaxLevel are simple percentages of the complete dpidADCRange. When doing an AutoRange measurement, mpidMaxADCValue is checked against this range to verify that a sufficient integration time has been found. This means that the DC is not taken into account. So if you specify a low minimal range of e.g. 10%, these 10% - for long integration times - might actually be reached just by the DC itself, so there's no real signal. This is contrary to mpidRelSaturation where the DC is taken into account, so the percentages of mpidAutoRangeMinLevel and mpidAutoRangeMaxLevel cannot be compared with mpidRelSaturation.

4.18.2 AutoRange for similar intensities

To enable AutoRange for similar intensities, use the command casSetOptionsOnOff and enable the option co↩ AutorangeMeasurement and, if desired, coAutorangeFilter.

To perform the actual AutoRange measurement, just call casMeasure like you would for a normal measurement. This will start a dark-current measurement, if necessary, and then a normal measurement with the current measurement parameter. If the results meet the criteria given via the AutoRange parameter, casMeasure returns.

So for similar intensities, once suitable parameter have been found during the initial AutoRange measurement, the following calls to casMeasure can return after the first acquisition.

If the first measurement does not meet the AutoRange criteria, the integration time and density filter are changed accordingly, a new DC is measured (or calculated from the DC array, see warning below) and then another measurement is made. This will repeat several times until the criteria are met or AutoRange gives up. Note that this procedure will be especially slow if a high number of averages are set up.

Warning

An AutoRange measurement for similar intensities might include necessary dark current measurements. For devices without a shutter, it is therefore mandatory to enable the dark current array (see chapter Dark Current). Otherwise the AutoRange measurement might measure a dark current without the optical path being closed! For the dark current array to actually avoid shutter action, the mpidAutoRangeMaxIntTime must also stay within the bounds of the dark array. The integration times of the dark current array can be retrieved with casGetCalibrationFactors using AWhat = gcfDarkArrayValues.

Note

If you're measuring DUTs which vary greatly you should consider using the new AutoRange method for various intensities. Another approach is to speed up AutoRange measurements by setting mpidIntegrationTime and Density Filter to low values before measuring. That way you are avoiding a lengthy and over-exposed first acquisition if the previous AutoRange measurement changed these parameter because the DUT had a very low intensity.

Warning

AutoRange is only supported for synchronous measurements using casMeasure, but not for asynchronous measurements with casStart!

4.18.3 AutoRange for various intensities

For AutoRange with various intensities, simply call casPerformActionEx with paAutoRangeFindParameter before doing the actual measurement.

paAutoRangeFindParameter will start measurements, beginning with dpidIntTimeMin and dpidAutoRangeFilterMin, but to save time, without averages and without measuring any dark current. Once sufficient integration time and density filter have been found, paAutoRangeFindParameter will return (or fail with an error). After that you should do a dark current measurement if necessary and then perform the measurement like you would normally do, typically by calling casMeasure.

To allow paAutoRangeFindParameter to change the density filter, enable the option coAutorangeFilter using cas↩ SetOptionsOnOff. It is NOT recommended to enable coAutorangeMeasurement, because a subsequent measurement would then try to find suiting parameter again.

4.19 Working with Parameter Sets

The CAS DLL provides so called parameter sets. By using parameter sets it is possible to quickly change between different sets of measurement parameter. Since each parameter set holds it's own dark current, spectrum and results, this makes it possible to quickly measure several spectra with varying parameter without the need of dark current measurements in between.

The following parameters can be changed for each defined parameter set:

- density filter
- integration time
- averages
- trigger delay time
- flash delay time
- flash duration
- CheckStart and CheckStop
- ColormetricStart and ColormetricStop
- GetWidth Level
- TOP aperture
- TOP distance
- PulseWidth

The following commands and parameter are used to manage parameter sets

- dpidParamSets - defines the number of parameter sets. When increasing dpidParamSets, new parameter sets are created by copying the currently active parameter set. If the currently active parameter set is deleted because dpidParamSets was decreased, the first parameter set will become active.
- dpidCurrentParamSet is the currently active parameter set. This device parameter has to be set prior to every action that is related to measurement parameters or measurement data. It is often used inside a loop to cycle through all parameter sets (see Example below)
- casDeleteParamSet is used for deleting a specific parameter set, since decreasing the number of parameter sets only deletes the sets at the end

Starting with CAS SDK version 4.10 the new paMeasureParamSets API can be used to conveniently measure a range of parameter sets. Some spectrometer types even support performing this set of measurements completely autonomously, a feature that is sometimes referred to as recipe mode.

For dark current measurements use paMeasureParamSetsDC. As with normal dark current measurements, the shutter should be closed before calling the API and opened afterwards.

Note

paMeasureParamSetsDC avoids measuring multiple DCs with the same settings by copying the DCs into the appropriate parameter sets instead of measuring again.
Both paMeasureParamSets and paMeasureParamSetsDC are synchronous, i.e. they only return once the operation has been completed or an error occured. In case of an error, use mpidMeasured to determine whether a given parameter set was measured successfully or not.

4.20 Using Serial Numbers

For many years now, all spectrometers by Instrument Systems have their serial number stored internally so that it can be retrieved by software.

Additionally a lot of accessories have their own serial number. If these are connected to the spectrometer correctly, these accessory serial numbers can be retrieved by software as well. To check whether a spectrometer supports reading accessory serial numbers, check the coGetAccessories bit with casGetOptions.

The general serial number dpidSerialNo can consist of up to 3 serial numbers divided by a semicolon. First the serial of the spectrometer, then that of additional accessory and finally the serial number of a TOP, if it is required by the current configuration/calibration. To retrieve the individual serial numbers, use casGetSerialNumberEx.

Warning

> You should use the function casPerformActionEx with paCheckAccessories to ensure that the currently used calibration and configuration is intended for this spectrometer and optional accessory. If you want this check to be performed automatically, set the option coCheckCalibConfigSerials using the command casSetOptionsOnOff. If this option is enabled, casInitialize may also return error code ErrorConfigSerialMismatch or ErrorCalibSerialMismatch indicating that either the configuration or the calibration file doesn't match the serial number(s) of the device. Additionally the CAS DLL can now verify the serial of a transmission file (dpidTransmissionFileName). To perform this verification call casPerformActionEx with paCheckAccessoriesTransmission. In case of a mismatch it will return ErrorTransmissionSerialMismatch.

Note

> All the relevant files from Instrument Systems contain the serial number(s) they are made for. If these files don't contain the intended serial number(s), the paCheckAccesories and paCheckAccessoriesTransmission mentioned above can never fail.

The ID Key is only read during casInitialize, when validating the serials manually with casPerformActionEx and paCheckAccessories or paCheckAccessoriesTransmission, when reading dpidAccessorySerial or when calling casGetSerialNumberEx with casSerialAccessory.

Warning

> If you plug in the ID Key after the device has been initialized, it is not re-read when getting dpidSerialNo or calling casGetSerialNumberEx with casSerialComplete. Instead use one of the methods mentioned above.

4.21 MultiTrack measurements

The CAS DLL is able to perform so called MultiTrack measurements. These are very fast continuous measurements without any acquisition gaps. This is possible because the acquired raw values are copied into an internal buffer and any correction or calculation can be performed afterwards.

Note

Not all interface types support MultiTrack. Interfaces that do support MultiTrack will include the coCanMultiTrack flag in casGetOptions.
Starting with CAS DLL 4.5 MultiTrack supports more interface types - even the Demo mode now supports it. Note that the necessary calls have changed (see below). The old way of doing MultiTrack, which involved calling casMultiTrackReadData and casMultiTrackCopySet is deprecated now and will only work with the PCI interface for the time being.

To perform a MultiTrack measurement these tasks have to be done in the order listed below:

1. All preconditions that are mentioned in Performing a Measurement are met.
2. Call casMultiTrackInit to initialize the internal buffer and specify the number of MultiTrack measurements. This number must not exceed dpidMultiTrackMaxCount.
3. You may check the actual size of the buffer with dpidMultiTrackCount.
4. Start the MultiTrack measurement by calling casPerformActionEx with paMultiTrackStart. This will return once all measurements have been made. Triggering the first track is supported.
5. The MultiTrack measurements can be saved to a file with casMultiTrackSaveData and loaded using casMultiTrackLoadData.
6. For calculating colormetric and other results, call casMultiTrackCopyData for each track. This reloads the raw spectrum from the MultiTrack buffer and all other methods use it as if it would have just been measured. After that, proceed as described in the chapter Getting the Spectrum and Results.
7. Finally call casMultiTrackDone to release the memory that was allocated for the MultiTrack buffer.

Important notes about MultiTrack files:

- We recommend using the .swm file extension
- Before calling casMultiTrackLoadData it is essential that the same configuration and calibration files were used to initialize the device! It is perfectly fine to use a demo mode CAS ID, i.e. for loading the spectrometer hardware does not need to be present.
- After loading and working with MultiTrack data, a call to casMultiTrackDone is necessary to release the MultiTrack buffer.
- Some MultiTrack specific results are mpidMultiTrackAcqTime and mpidCMTTrackStart.

4.22 CCD Temperature Monitoring

Various spectrometer types support measuring the CCD temperature, indicated by returning the coGetTemperature flag when calling casGetOptions.

For these spectrometers, it is highly recommended that you start a temperature measurement from time to time by querying mpidCurrentCCDTemperature. Since measuring the temperature might take a few milliseconds, choose a time which is convenient for you, for example after a measurement has been performed and the DUT has been disconnected.

This will ensure that hardware defects are detected (which would cause an ErrorCCDTemperatureFail) and also that, if necessary, a new dark current measurement will be requested (via dpidDCRemeasureReasons, as described in chapter Dark Current).

If you want to log the last measured temperature, without actually causing a temperature measurement, query mpidLastCCDTemperature. Note that this caching only works if the last temperature measurement is not too old (time frame varies by spectrometer type). Otherwise the temperature is actually measured, which might cause a delay.

Whenever a dark current is measured, the CCD temperature is also acquired. You can retrieve it by querying mpidDCCCDTemperature.

4.23 Transmission measurement

This section describes how to perform a transmission measurement where the reference spectrum is measured (as opposed to being loaded from a file etc.).

Warning

> This topic should not be confused with taking the transmission of additional optical equipment into account! That can be done using dpidTransmissionFileName and enabling coUseTransmission with casSetOptionsOnOff.

Preparations

Since a calibration file is required before calling casInitialize, it is recommended to use a 1-calibration, typically called "one.isc". Alternatively use the original calibration file and after casInitialize, call casClearCalibration with gcfSensitivityFunction like in the example provided below. Then set up the measurement parameter and perform a dark current measurement.

Measuring the reference and applying it as a calibration

The reference measurement itself can be started like any normal measurement, either synchronously (casMeasure) or asynchronously (casStart with casFIFOHasData and casGetFIFOData).

To automatically divide future transmission measurements by the reference spectrum, it is stored as the sensitivity calibration using casSetCalibrationFactors:

[Delphi]

```
//CasID holds the device handle
//the device must have been configured and initialized

//clear the sensitivity to make sure a previous
//reference is not applied
casClearCalibration(CasID, gcfSensitivityFunction);
CheckError(casGetError(CasID));

//set density filter if necessary
if Round(casGetDeviceParameter(CasID, dpidNeedDensityFilterChange)<>0 then
  CheckError(casSetMeasurementParameter(CasID, mpidDensityFilter, casGetMeasurementParameter(CasID,
       mpidNewDensityFilter)));

//dark current measurement if necessary
if Round(casGetDeviceParameter(CasID, dpidDCRemeasureReasons)<>0 then
begin
  casSetShutter(CasID, 1);
  CheckError(casGetError(CasID));

  CheckError(casMeasureDarkCurrent(CasID));

  casSetShutter(CasID, 0);
  CheckError(casGetError(CasID));
end;

//perform the reference measurement
casMeasure(CasID);
CheckError(casGetError(CasID));

//set sensitivity calibration to reference spectrum
//loop trough all Pixels

for i:= 0 to Round(casGetDeviceParameter(CasID, dpidPixels))-1 do
begin
  casSetCalibrationFactors(CasID, gcfSensitivityFunction, i, 0, casGetData(CasID, i));
  CheckError(casGetError(CasID));
end;

//make sure the new "calibration" is taken into account
casPerformActionEx(CasID, paUpdateSpectralCalibration, 0, 0);
CheckError(casGetError(CasID));
```

Transmission measurements

After the reference spectrum has been assigned as sensitivity calibration, every subsequent measurement has the reference spectrum automatically applied. To clarify: the CAS DLL divides the spectrum with gcfSensitivityFunction, so the spectrum will be divided by the reference, resulting in a transmission spectrum.

Note

A new initialization (see casInitialize) or paUpdateCompleteCalibration will restore the original calibration from the calibration file.

Replacing the reference spectrum

To replace a previous reference spectrum, either re-initialize the device (see note above) or call casClearCalibration with gcfSensitivityFunction. This is done at the beginning of the example above, to make sure the old reference spectrum is not taken into account when a new reference is measured.

Chapter 5

Namespace Index

5.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

Namespace Documentation

7.1 InstrumentSystems Namespace Reference

Namespaces

- namespace CAS4

  The InstrumentSystems namespace.

7.2 InstrumentSystems.CAS4 Namespace Reference

The InstrumentSystems namespace.

Classes

- class CAS4DLL

  The CAS DLL interface class

7.2.1 Detailed Description

The InstrumentSystems namespace.

Chapter 8

Class Documentation

8.1 InstrumentSystems.CAS4.CAS4DLL Class Reference

The CAS DLL interface class

Static Public Member Functions

- static int casGetError (int ADevice)

    Return error code for a given device/CASID
- static IntPtr casGetErrorMessage (int AError, StringBuilder ADest, int AMaxLen)

    Translates a given error code into a readable error message
- static int casCreateDevice ()

    Deprecated method. Creates a device context aka CASID within the CAS DLL
- static int casCreateDeviceEx (int AInterfaceType, int AInterfaceOption)

    Creates a device context aka CASID within the CAS DLL
- static int casChangeDevice (int ADevice, int AInterfaceType, int AInterfaceOption)

    Change the interface type and/or option of a device / CASID
- static int casDoneDevice (int ADevice)

    Release resources used by the device
- static int casAssignDeviceEx (int ASourceDevice, int ADestDevice, int AOption)

    Assigns properties and or parameters from one device handle to another
- static int casGetDeviceTypes ()

    Retrieves the number of interface types the CAS DLL supports
- static IntPtr casGetDeviceTypeName (int AInterfaceType, StringBuilder ADest, int AMaxLen)

    Retrieves the name of the given interface type
- static int casGetDeviceTypeOptions (int AInterfaceType)

    Retrieves the number of options a given interface type currently supports
- static int casGetDeviceTypeOption (int AInterfaceType, int AIndex)

    Returns the value of the interface option for the given interface type and option index
- static IntPtr casGetDeviceTypeOptionName (int AInterfaceType, int AInterfaceOptionIndex, StringBuilder A↵ Dest, int AMaxLen)

    Retrieves the name of the given interface option
- static int casInitialize (int ADevice, int Perform)

    Initializes the hardware of the device after loading the configuration and calibration files
- static double casGetDeviceParameter (int ADevice, int AWhat)

Retrieves a float representing a device parameter

- static int casSetDeviceParameter (int ADevice, int AWhat, double AValue)

  Sets a numerical device parameter
- static int casGetDeviceParameterString (int ADevice, int AWhat, StringBuilder ADest, int AMaxLen)

  Retrieves a string representing a device parameter
- static int casSetDeviceParameterString (int ADevice, int AWhat, string AValue)

  Sets a string device parameter
- static int casGetSerialNumberEx (int ADevice, int AWhat, StringBuilder ADest, int AMaxLen)

  Retrieves serial numbers of the specified device and/or additional information.
- static int casGetOptions (int ADevice)

  Returns the features and options the device currently supports or performs.
- static void casSetOptionsOnOff (int ADevice, int AOptions, int AOnOff)

  This method can set or clear several options for the device
- static void casSetOptions (int ADevice, int AOptions)

  This method sets and clears all device options for the device
- static int casMeasure (int ADevice)

  Performs a measurement for the given device using the measurement parameter which have been previously set
- static int casStart (int ADevice)

  Starts a measurement for the given device and returns immediately
- static int casFIFOHasData (int ADevice)

  Use this method to check if the spectrometer has data available which can be read
- static int casGetFIFOData (int ADevice)

  This method reads the acquired spectrum from the FIFO and stores it internally
- static int casMeasureDarkCurrent (int ADevice)

  Use casMeasureDarkCurrent to perform a dark current measurement for the given device.
- static int casPerformActionEx (int ADevice, int AActionID, int AParam1, int AParam2, IntPtr AParam3)

  Generic method which performs one of various actions with the specified device. This method replaces the now deprecated method casPerformAction.
- static int casPerformAction (int ADevice, int AID)

  Deprecated. Use casPerformActionEx instead!
- static double casGetMeasurementParameter (int ADevice, int AWhat)

  Returns a numeric measurement parameter for a given spectrometer
- static int casSetMeasurementParameter (int ADevice, int AWhat, double AValue)

  Sets a numeric measurement parameter for a given spectrometer
- static int casClearDarkCurrent (int ADevice)

  Clears a previously measured dark current of the given device
- static int casDeleteParamSet (int ADevice, int AParamSet)

  Deletes a specific parameter set for the given device.
- static int casGetShutter (int ADevice)

  Returns the current shutter position of the given device
- static void casSetShutter (int ADevice, int OnOff)

  Sets the shutter position of the given device.
- static IntPtr casGetFilterName (int ADevice, int AFilter, StringBuilder Dest, int AMaxLen)

  Translates a filter index into a user-readable density filter name.
- static int casGetDigitalOut (int ADevice, int APort)

  Returns the state the specified digital output port was set to by a former call to casSetDigitalOut.
- static void casSetDigitalOut (int ADevice, int APort, int OnOff)

  Sets the state the specified digital output port.
- static int casGetDigitalIn (int ADevice, int APort)

  Returns the state of the specified digital input port.
- static void casCalculateCorrectedData (int ADevice)

Applies the spectral correction to the previously acquired raw spectrum.
- static void casConvoluteTransmission (int ADevice)

  Applies the spectral and transmission correction to the previously acquired raw spectrum.
- static double casGetCalibrationFactors (int ADevice, int AWhat, int AIndex, int AExtra)

  Returns various details about the configuration/calibration of the given device.
- static void casSetCalibrationFactors (int ADevice, int AWhat, int AIndex, int AExtra, double AValue)

  Changes various details about the configuration/calibration of the given device effectively overriding information that normally comes from configuration/calibration files.
- static void casUpdateCalibrations (int ADevice)

  Updates the calibration information for the given device. Deprecated! Use paUpdateSpectralCalibration.
- static void casSaveCalibration (int ADevice, string AFileName)

  Saves the calibration of the given device to a file.
- static void casClearCalibration (int ADevice, int AWhat)

  Clears the specified calibration part of the given device.
- static double casGetData (int ADevice, int AIndex)

  Returns the intensity of the previously acquired spectrum for the given pixel.
- static double casGetXArray (int ADevice, int AIndex)

  Returns the wavelength of the spectrum for the given pixel.
- static double casGetDarkCurrent (int ADevice, int AIndex)

  Returns the intensity of the previously measured/calculated dark current for the given pixel.
- static void casGetPhotInt (int ADevice, out double APhotInt, StringBuilder AUnit, int AUnitMaxLen)

  Retrieves the previously calculated photometric integral and it's data unit.
- static void casGetRadInt (int ADevice, out double ARadInt, StringBuilder AUnit, int AUnitMaxLen)

  Retrieves the previously calculated radiometric integral and it's data unit.
- static double casGetCentroid (int ADevice)

  Retrieves the previously calculated centroid wavelength.
- static void casGetPeak (int ADevice, out double x, out double y)

  Retrieves the previously calculated peak wavelength and intensity.
- static double casGetWidth (int ADevice)

  Retrieves the peak width in nm.
- static double casGetWidthEx (int ADevice, int AWhat)

  Calculates and retrieves various aspects about the peak width (e.g. full width half maximum aka FWHM aka 50% bandwidth).
- static void casGetColorCoordinates (int ADevice, ref double x, ref double y, ref double z, ref double u, ref double v1976, ref double v1960)

  Retrieves CIE color coordinates of a previously measured spectrum.
- static double casGetCCT (int ADevice)

  Calculates the correlated color temperature CCT of a previously measured spectrum.
- static double casGetCRI (int ADevice, int Index)

  Retrieves one of the previsouly calculated the color rendering indices.
- static void casGetTriStimulus (int ADevice, ref double X, ref double Y, ref double Z)

  Retrieves the previously calculated X, Y and Z tristimulus values.
- static double casGetExtendedColorValues (int ADevice, int AWhat)

  Retrieves results and conditions for previously performed colormetric calculations.
- static int casColorMetric (int ADevice)

  Calculates colormetric results for the previously measured spectrum.
- static int casCalculateCRI (int ADevice)

  Calculates the color rendering indices (CRI) of a previously measured spectrum.
- static int cmXYToDominantWavelength (double x, double y, double IllX, double IllY, ref double LambdaDom, ref double Purity)

Calculates dominant wavelength aka LambdaDom and purity from a given color coordinate and illuminant reference. This method is deprecated because it does not take the new mpidColormetricType into account. Use casCalculate↩ LambdaDom instead.

- static int casCalculateLambdaDom (int ADevice, double IllX, double IllY, ref double LambdaDom, ref double Purity)

  Calculates dominant wavelength aka LambdaDom and purity for the specified illuminant reference and the color coordinates that have been calculated by casColorMetric. This method replaces the now deprecated cmXYTo↩ DominantWavelength and does take the new mpidColormetricType into account.
- static IntPtr casGetDLLFileName (StringBuilder Dest, int AMaxLen)

  Retrieves the complete path and file name of CAS library
- static IntPtr casGetDLLVersionNumber (StringBuilder Dest, int AMaxLen)

  Retrieves the version of CAS library
- static int casSaveSpectrum (int ADevice, string AFileName)

  Saves a previously measured spectrum to an .ISD file.
- static double casGetExternalADCValue (int ADevice, int AIndex)

  Obsolete. Use mpidCurrentCCDTemperature. Used to retrieve the CCD temperature.
- static void casSetStatusLED (int ADevice, int AWhat)

  Method to control the status LED of the spectrometer
- static int casNmToPixel (int ADevice, double nm)

  Converts a wavelength into the corresponding CCD pixel index.
- static double casPixelToNm (int ADevice, int APixel)

  Converts a CCD pixel index into the corresponding wavelength.
- static int casCalculateTOPParameter (int ADevice, int AAperture, double ADistance, ref double ASpotSize, ref double AFieldOfView)

  Calculates spot size and field of view of the TOP.
- static int casMultiTrackInit (int ADevice, int ATracks)

  Initializes the MultiTrack buffer
- static int casMultiTrackDone (int ADevice)

  Releases a previously allocated MultiTrack buffer
- static int casMultiTrackCount (int ADevice)

  Deprecated method.
- static int casMultiTrackCopyData (int ADevice, int ATrack)

  Releases a previously allocated MultiTrack buffer
- static int casMultiTrackSaveData (int ADevice, string AFileName)

  Saves a MultiTrack buffer to a .SWM file.
- static int casMultiTrackLoadData (int ADevice, string AFileName)

  Loads a MultiTrack buffer from a .SWM file.
- static void casLoadTestData (int ADevice, string AFileName)

  Loads a spectrum from an .ISD file into the CASID

Public Attributes

- const int ErrorNoError = 0

  No error, i.e. function was successful.
- const int ErrorUnknown = -1

  An unknown or unexpected error. Retrieve the corresponding error message via casGetErrorMessage for additional info.
- const int ErrorTimeoutRWSNoData = -2

  A timeout occurred while waiting for the spectrum data.
- const int ErrorInvalidDeviceType = -3

  Invalid InterfaceType constant passed to casCreateDeviceEx or casChangeDevice

- const int ErrorAcquisition = -4

  Acquisition failed, i.e. an error occurred while the spectrometer was measuring.
- const int ErrorAccuDataStream = -5

  Averaging failed, increase mpidIntegrationTime or make sure PC is not too busy.
- const int ErrorPrivilege = -6

  no longer used
- const int ErrorFIFOOverflow = -7

  It was detected that the FIFO of the interface card or the spectrometer was overflowing. Spectrum should be discarded.
- const int ErrorTimeoutEOSScan = -8

  The spectrometer FW reported a timeout while waiting for EOS of a spectrum scan.
- const int ErrorTimeoutEOSDummyScan = -9

  The spectrometer FW reported a timeout while waiting for EOS of a dummy scan.
- const int ErrorFifoFull = -10

  It was detected that the FIFO of the interface card or the spectrometer was full. Spectrum should be discarded.
- const int ErrorPixel1FinalCheck = -11

  After reading spectrum data, the FIFO did not contain a pixel marked as first pixel. Spectrum should be discarded.
- const int ErrorCCDTemperatureFail = -13

  The temperature of the CCD is outside of the allowed range. Temperature control failed or ambient temperature too high.
- const int ErrorAdrControl = -14

  The spectrometer hardware associated with the used Device/CASID could not be found or accessed. See chapter Interfaces Types and Options.
- const int ErrorFloat = -15

  Floating point error while calculating calibrated spectrum.
- const int ErrorTriggerTimeout = -16

  Timeout reached while waiting for the trigger. See mpidTriggerTimeout
- const int ErrorAbortWaitTrigger = -17

  While waiting for the trigger, the operation was aborted using dpidAbortWaitForTrigger
- const int ErrorDarkArray = -18

  An error occured while measuring the Dark current array.
- const int ErrorNoCalibration = -19

  Invalid calibration filename specified for dpidCalibFileName
- const int ErrorInterfaceVersion = -20

  The firmware version of the interface card (PCI with CAS140 CTS) or the driver version is too old to work with this device.
- const int ErrorCRI = -21

  Error calculating CRI (color rendering indices). For more info see casCalculateCRI
- const int ErrorNoMultiTrack = -25

  Old way of doing MultiTrack not supported for this interface type. See chapter MultiTrack measurements.
- const int ErrorInvalidTrack = -26

  no longer used
- const int ErrorDetectPixel = -31

  An error occurred while trying to detect the number of pixels the spectrometer has.
- const int ErrorSelectParamSet = -32

  Error while activating a parameter set dpidCurrentParamSet. See chapter Working with Parameter Sets.
- const int ErrorI2CInit = -35

  Initializing I2C bus failed.
- const int ErrorI2CBusy = -36

  I2C operation failed because the I2C bus was busy.
- const int ErrorI2CNotAck = -37

I2C operation was not acknowledged.

- const int ErrorI2CRelease = -38

  no longer used

- const int ErrorI2CTimeOut = -39

  I2C operation timed out.

- const int ErrorI2CTransmission = -40

  I2C transmission failed in an unspecified way.

- const int ErrorI2CController = -41

  The I2C controller responded in an unexpected way.

- const int ErrorDataNotAck = -42

  no longer used

- const int ErrorNoExternalADC = -52

  The temperature ADC should be read, but the spectrometer does not have such an ADC.

- const int ErrorShutterPos = -53

  Positioning the shutter or querying it's state failed. See chapter Density Filter.

- const int ErrorFilterPos = -54

  Positioning the filterwheel or querying it's position failed. See chapter Density Filter.

- const int ErrorConfigSerialMismatch = -55

  Configuration file not intended for this spectrometer and possible accessories. The serial number stored in the file dpidConfigFileName does not match the current dpidSerialNo. Refer to chapter Using Serial Numbers.

- const int ErrorCalibSerialMismatch = -56

  Calibration file not intended for this spectrometer and possible accessories. The serial number stored in the file dpidCalibFileName does not match the current dpidSerialNo. Refer to chapter Using Serial Numbers.

- const int ErrorInvalidParameter = -57

  Returned if there was an invalid "AWhat" parameter.

- const int ErrorGetFilterPos = -58

  no longer used

- const int ErrorParamOutOfRange = -59

  Returned by many methods indicating that a parameter was out of range; does not apply for ADevice parameter which results in errCasDeviceNotFound

- const int ErrorDeviceFileChecksum = -60

  Returned by dpidGetFilesFromDevice if there was a checksum error, so either the device is corrupt or the communication failed.

- const int ErrorInvalidEEPromType = -61

  Returned by dpidGetFilesFromDevice if the device does not support file storage.

- const int ErrorDeviceFileTooLarge = -62

  Not enough storage on the device.

- const int ErrorNoCommunication = -63

  The communication with the spectrometer ended unexpectedly.

- const int ErrorNoFilesOnIdentKey = -64

  Returned by dpidGetFilesFromDevice if no files were found on the device.

- const int ErrorExtraCalibFileInvalid = - 66

  Additional files of the calibration are either missing or corrupt.

- const int ErrorFeatureNotSupported = -68

  The requested feature is not supported by the device.

- const int ErrorConfigUpToDate = -70

  Config/calib files do not match the ones on the device. Returned by coCheckConfigUpToDate and casInitialize if option coCheckConfigUpToDate is set.

- const int ErrorCommunicationTimeout = -73

  The communication with the spectrometer reached a timeout.

- const int ErrorTransmissionSerialMismatch = -74

Transmission file not intended for this spectrometer and possible accessories. The serial number stored in the file dpidTransmissionFileName does not match the current dpidSerialNo. Refer to chapter Using Serial Numbers.

- const int errCASOK = ErrorNoError

  just an alias for ErrorNoError
- const int errCASError = -1000

  not returned; base for a few other errors like errCasDeviceNotFound
- const int errCasNoConfig = errCASError-3

  Invalid filename specified for dpidConfigFileName
- const int errCASDriverMissing = errCASError-6

  no longer used
- const int errCasDeviceNotFound = errCASError-10

  Invalid ADevice / CASID parameter.
- const int InterfaceISA = 0

  No longer used. Deprecated ISA interface constant.
- const int InterfacePCI = 1

  PCI interface constant. For use with e.g. casCreateDeviceEx. See chapter Interfaces Types and Options.
- const int InterfaceTest = 3

  Demo mode interface constant. For use with e.g. casCreateDeviceEx. See chapter Interfaces Types and Options.
- const int InterfaceUSB = 5

  USB interface constant. For use with e.g. casCreateDeviceEx. See chapter Interfaces Types and Options.
- const int InterfacePCIe = 10

  PCIe interface constant. For use with e.g. casCreateDeviceEx. See chapter Interfaces Types and Options.
- const int InterfaceEthernet = 11

  Ethernet interface constant. For use with e.g. casCreateDeviceEx. See chapter Interfaces Types and Options.
- const int aoAssignDevice = 0

  Only device specific properties are copied. This is rarely necessary to assign separately. Examples: current filter wheel state, flag whether data is present etc.
- const int aoAssignParameters = 1

  Only parameters are assigned. This includes all measurement parameters but also configuration and calibration as well as dark current. Pretty much everything which is not related to the state of the hardware.
- const int aoAssignComplete = 2

  Assigns everything, i.e. equal to calling casAssignDeviceEx with aoAssignDevice and then with aoAssignParameters.
- const int InitOnce = 0

  casInitialize: load config and calib, and only initialize the hardware if not done before
- const int InitForced = 1

  casInitialize: load config and calib, and always initialize the hardware even if it was initialized before
- const int InitNoHardware = 2

  casInitialize: load config and calib, but do not initialize the hardware
- const int dpidIntTimeMin = 101

  float: minimum integration time in ms as supported by the device; check when setting mpidIntegrationTime
- const int dpidIntTimeMax = 102

  float: maximum integration time in ms as supported by the device; check when setting mpidIntegrationTime
- const int dpidDeadPixels = 103

  number of dead pixels at the beginning of the pixel array; see casGetData
- const int dpidVisiblePixels = 104

  number of visible pixels after dead pixels in the pixel array that form the spectrum; see casGetData
- const int dpidPixels = 105

  the total number of pixels in the pixel array; defined by the configuration file and/or the device - should not be modified; see casGetData
- const int dpidParamSets = 106

  total number of parameter sets defined for this device - see Working with Parameter Sets; when set to 0, all are cleared but a new one is added immediately

- const int dpidCurrentParamSet = 107

  0-based index of the currently active parameter set - see Working with Parameter Sets
- const int dpidADCRange = 108

  the ADC level (in Counts) where saturation effects may start, i.e. if mpidMaxADCValue is larger than dpidADCRange, the measurement was saturated
- const int dpidADCBits = 109

  the bit resolution of the ADC - for information purposes only. To check for saturation, use dpidADCRange
- const int dpidSerialNo = 110

  the complete serial number of spectrometer and accessories if applicable. See Using Serial Numbers for an overview
- const int dpidTOPSerial = 111

  deprecated: the serial number of a TOP200 that is required by the current calibration. Rather use dpidTOPSerialEx and refer to Using Serial Numbers for an overview
- const int dpidTransmissionFileName = 112

  Returns or sets the path of thetransmission correction file (typical extension .isa); see coUseTransmission. Note that an invalid filename will not raise an error. The file is loaded during casInitialize or when paUpdateCompleteCalibration is called.
- const int dpidConfigFileName = 113

  Returns or sets the path of the currently used configuration file (extension .ini). This file is required and loaded during casInitialize or errCasNoConfig will occur.
- const int dpidCalibFileName = 114

  Returns or sets the path of the currently used calibration file (extension .isc). This file is required and loaded during casInitialize or ErrorNoCalibration will occur.
- const int dpidCalibrationUnit = 115

  the calibration unit as defined by the calibration file dpidCalibFileName
- const int dpidAccessorySerial = 116

  serial number of the ID key connected to the spectrometer. See Using Serial Numbers for an overview
- const int dpidTriggerCapabilities = 118

  Returns a bit-set describing the trigger capabilities of the spectrometer. tcoCanTrigger and others. For more info, see task Triggered Measurements.
- const int dpidAveragesMax = 119

  Returns maximum number of averages supported by the device/configuration. Use for validating mpidAverages
- const int dpidFilterType = 120

  Returns the density filter type: 0 = none, 1 = manual filter, 2,3 = filter wheel. Refer to Density Filter for an overview.
- const int dpidRelSaturationMin = 123

  Returns the minimum relative saturation in % which is still considered good for this device. Compare against mpid↩ RelSaturation
- const int dpidRelSaturationMax = 124

  Returns the maximum relative saturation in % which is still considered good for this device. Compare against mpid↩ RelSaturation
- const int dpidInterfaceVersion = 125

  Returns an optional version information for the device/configuration. Currently only the PCI-Interface supports an interface version, returning the CPLD version of the used PCI-Card (divide by 100, i.e. 123 for version 1.23), all other interface types return 0.
- const int dpidTriggerDelayTimeMax = 126

  float: maximum trigger delay time in ms supported by the device/configuration. Use to check user input for the mpidTriggerDelayTime measurement parameter.
- const int dpidSpectrometerName = 127

  Returns the name of the spectrometer type as specified in the configuration file.
- const int dpidNeedDarkCurrent = 130

  Deprecated, rather check dpidDCRemeasureReasons and refer to chapter dark current for an overview.
- const int dpidNeedDensityFilterChange = 131

  Returns whether the density filter needs to be moved (all values <> 0). Refer to chapter density filer for an overview.
- const int dpidSpectrometerModel = 132

Returns an ID which identifies the spectrometer type as specified in the configuration file. Rather use dpid↵ SpectrometerName

- const int dpidLine1FlipFlop = 133

  Returns the state of the trigger flipflop on line 1 (CAS140B and CAS140CT only, refer to the hardware manual and the tcoGetFlipState trigger capability). The value is 0 for low state or 1 for high. When setting this value, the flipflop is reset, regardless of the value which is passed.

- const int dpidTimer = 134

  Returns the internal timer. Use not recommended. Rather use mpidTimeSinceScanStart

- const int dpidInterfaceType = 135

  Returns the interface type for the specified device (or a negative error). See chapter Interface type and Options for an overview.

- const int dpidInterfaceOption = 136

  Returns the interface option for the specified device (or a negative error). See chapter Interface type and Options for an overview.

- const int dpidInitialized = 137

  Returns whether the device has been correctly initialized (see casInitialize) or not. A value bigger than 0 indicates it has been initialized, 0 that it hasn't and a negative value indicates an error.

- const int dpidDCRemeasureReasons = 138

  Returns a set of flags which indicate why a dark current measurement is necessary. Values include todcrrNeed↵ DarkCurrent and todcrrCCDTemperature. See chapter dark current for an overview.

- const int dpidAbortWaitForTrigger = 140

  Not recommended - setting dpidAbortWaitForTrigger to non-zero will abort a triggered acquisition if the trigger has not yet occurred. Only supported by PCI interface.

- const int dpidGetFilesFromDevice = 142

  Set to the path of an existing directory and immediately a download of the files on the device will start. This might take considerable time, i.e. several seconds. The device does not have to be initialized, but interface type and option have to be set. Might return ErrorInvalidEEPromType or ErrrorDeviceFileCheckSum

- const int dpidTOPType = 143

  Returns the type of TOP which is required by the current configuration. Possible values start at ttNone

- const int dpidTOPSerialEx = 144

  Returns the serial number of the TOP which is required by the current configuration or an empty string if no TOP is required. See Using Serial Numbers.

- const int dpidAutoRangeFilterMin = 145

  Identifies the lowest density filter index for mpidDensityFilter that should be used for AutoRange measurements with the coAutorangeFilter option. This constant is part of the configuration file and should not be changed.

- const int dpidAutoRangeFilterMax = 146

  Identifies the highest density filter index for mpidDensityFilter that should be used for AutoRange measurements with the coAutorangeFilter option. This constant is part of the configuration file and should not be changed.

- const int dpidMultiTrackMaxCount = 147

  Returns the maximum number of tracks which are supported for MultiTrack measurements with this device.

- const int dpidSLCFileInfo = 148

  Returns file path and timestamp for a straylight calibration file, if present. Otherwise an empty string is returned.

- const int dpidCheckConfigFileSerial = 149

  Write-only string device parameter. Call with dpidSerialNo from another CASID to check whether this CASID uses the correct configuration file for this accessory.

- const int dpidCheckCalibFileSerial = 150

  Write-only string device parameter. Call with dpidSerialNo from another CASID to check whether this CASID uses the correct calibration file for this accessory.

- const int dpidExtraTransmissionsFileInfo = 152

  Returns file information about additional transmission files the calibration uses or an empty string if it doesn't.

- const int dpidMultiTrackCount = 153

  Returns the number of MultiTracks which have been initialized/allocated for MultiTrack measurements with this device.

- const int dpidIntTimePossibleResolutions = 154

Returns a string containing the supported values for the mpidIntTimeResolution, separated by semicolons. The string is formatted using the current system locale, times are in microseconds. Examples: "1000;100;25", but might also be "1000" if only 1ms is supported.

- const int dpidCalibDate = 155

  Returns a string in ISO 8601 format containing the calibration date. Only valid after a call to casInitialize.
- const int dpidTriggerDelayTimeMin = 159

  float: minimum trigger delay time in ms supported by the device/configuration. Use to check user input for the mpidTriggerDelayTime measurement parameter.
- const int dpidWavelengthCalibrationType = 160

  string: returns the type of wavelength calibration for the given CAS ID and it's INI/ISC. Only valid after casInitialize has been called successfully.
- const int dpidCheckReferenceSpectrumFile = 162

  string, write-only: pass the complete file path of an ACS reference spectrum to check whether that spectrum has matching settings, so it can be compared with spectra of this CAS ID. Returns ErrorInvalidCalibration if the spectrum file could not be loaded, ErrorInvalidParameter if the settings of the spectrum do not match that of the CAS ID and ErrorNoError if they do match.
- const int dpidFlashDurationMin = 163

  float, read-only: minimum supported mpidFlashDuration in milliseconds. Only applicable if dpidTriggerCapabilities contains tcoFlashHardwareDuration.
- const int dpidDebugLogFile = 204

  complete filename and path where debug log should be written to. Set to empty path to stop logging. This dpid does not require that you pass valid CASID when getting or setting this parameter (any value will be accepted).
- const int dpidDebugLogLevel = 205

  integer level describing what to log, each higher level includes everything the levels below log, plus the things in this level. See constants starting with DebugLogLevelErrors. This dpid does not require that you pass valid CASID when getting or setting this parameter (any value will be accepted).
- const int dpidDebugMaxLogSize = 206

  integer file size, in bytes, the logfile should not exceed. Once this size is reached, a backup file with the same name but a .bak extension of the log is created and a new logfile starts. An already existing backup file will be deleted. This dpid does not require that you pass valid CASID when getting or setting this parameter (any value will be accepted).
- const int tcoCanTrigger = 0x00000001

  bit for dpidTriggerCapabilities. Spectrometer can be triggered externally, i.e. mpidTriggerSource supports trgFlipFlop
- const int tcoTriggerDelay = 0x00000002

  bit for dpidTriggerCapabilities. Spectrometer supports a trigger delay (mpidTriggerDelayTime).
- const int tcoTriggerOnlyWhenReady = 0x00000004

  bit for dpidTriggerCapabilities. Spectrometer supports changing the toAcceptOnlyWhenReady trigger option (mpidTriggerOptions)
- const int tcoAutoRangeTriggering = 0x00000008

  bit for dpidTriggerCapabilities. Spectrometer supports the toForEachAutoRangeTrial trigger option (mpidTriggerOptions)
- const int tcoShowBusyState = 0x00000010

  bit for dpidTriggerCapabilities. Spectrometer supports the toShowBusyState trigger option (mpidTriggerOptions)
- const int tcoShowACQState = 0x00000020

  bit for dpidTriggerCapabilities. Spectrometer supports the toShowACQState trigger option (mpidTriggerOptions)
- const int tcoFlashOutput = 0x00000040

  bit for dpidTriggerCapabilities. Spectrometer supports flash output in general, i.e. mpidFlashType. Rather check tcoFlashHardware for ftHardware and tcoFlashSoftware for ftSoftware
- const int tcoFlashHardware = 0x00000080

  bit for dpidTriggerCapabilities. Spectrometer supports flash hardware, ftHardware for mpidFlashType
- const int tcoFlashHardwareForEachAverage = 0x00000100

  bit for dpidTriggerCapabilities. Spectrometer can output a flash signal for each averaged spectrum (foEveryAverage option of mpidFlashOptions, ftHardware)
- const int tcoFlashSoftwareDelay = 0x00000200

  bit for dpidTriggerCapabilities. Spectrometer supports flash delay (mpidFlashDelayTime), for ftSoftware

- const int tcoFlashSoftwareDelayNegative = 0x00000400

  bit for dpidTriggerCapabilities. Spectrometer supports a negative flash delay (mpidFlashDelayTime) for ftSoftware
- const int tcoFlashSoftware = 0x00000800

  bit for dpidTriggerCapabilities. Spectrometer supports flash type ftSoftware (mpidFlashType)
- const int tcoGetFlipFlopState = 0x00001000

  bit for dpidTriggerCapabilities. Spectrometer supports reading the flip flop state using dpidLine1FlipFlop
- const int tcoQueryHasData = 0x00002000

  bit for dpidTriggerCapabilities. Spectrometer supports querying the FIFO state using casFIFOHasData
- const int tcoACQStatePolarity = 0x00004000

  bit for dpidTriggerCapabilities. Spectrometer supports changing the polarity of the ACQ state line via mpidACQ↩ StateLinePolarity
- const int tcoBusyStatePolarity = 0x00008000

  bit for dpidTriggerCapabilities. Spectrometer supports changing the polarity of the Busy state line via mpidBusy↩ StateLinePolarity
- const int tcoFlashHardwareDelay = 0x00010000

  bit for dpidTriggerCapabilities. Spectrometer supports flash delay (mpidFlashDelayTime), for ftHardware
- const int tcoFlashHardwareDelayNegative = 0x00020000

  bit for dpidTriggerCapabilities. Spectrometer supports a negative flash delay (mpidFlashDelayTime) for ftHardware
- const int tcoFlashHardwareDuration = 0x00040000

  bit for dpidTriggerCapabilities. Spectrometer supports an adjustable flash duration (mpidFlashDuration) for ftHardware
- const int todcrrNeedDarkCurrent = 0x0001

  bit for dpidDCRemeasureReasons. No valid dark current measurement present. This flag is identical to the now obsolete dpidNeedDarkCurrent.
- const int todcrrCCDTemperature = 0x0002

  bit for dpidDCRemeasureReasons. A new dark current measurement should be done, since the CCD temperature has changed too much (varies depending on device; mpidDCCCDTemperature and mpidLastCCDTemperature are compared).
- const int ttNone = 0

  dpidTOPType constant for no TOP required
- const int ttTOP100 = 1

  dpidTOPType constant for TOP 100 required
- const int ttTOP200 = 2

  dpidTOPType constant for TOP 200 required
- const int ttTOP150 = 3

  dpidTOPType constant for TOP 150 required
- const int ttTOPLumiTOP = 4

  dpidTOPType constant for LumiTOP required
- const int DebugLogLevelErrors = 1

  dpidDebugLogLevel constant, only log errors
- const int DebugLogLevelSaturation = 2

  dpidDebugLogLevel constant, only log errors and CCD saturation warnings
- const int DebugLogLevelHardwareEvents = 3

  dpidDebugLogLevel constant, additionally log any hardware events
- const int DebugLogLevelParameterChanges = 4

  dpidDebugLogLevel constant, additionally log any mpid oder dpid changes
- const int DebugLogLevelAllMethodCalls = 10

  dpidDebugLogLevel constant, log every API call - warning, this may slow things down and create a big backlog of things which still need to be logged
- const int casSerialComplete = 0

  casGetSerialNumberEx constant to retrieve complete serial number string, identical to dpidSerialNo
- const int casSerialAccessory = 1

casGetSerialNumberEx constant to retrieve the serial number of additional equipment, if present, otherwise "N/A".; identical to dpidAccessorySerial

- const int casSerialExtInfo = 2

  casGetSerialNumberEx constant to retrieve extended information, i.e. a multiline string with CAS type, ADC bits, serial number, No. of Pixels, ActivePixels and additional information depending on type and version of the firmware.
- const int casSerialDevice = 3

  casGetSerialNumberEx constant to retrieve the serial number read from the device; note that this might differ from case casSerialComplete without accessories or a TOP, since casSerialComplete falls back to the configuration file or its file name for a serial number (see Using Serial Numbers for more details)
- const int casSerialTOP = 4

  casGetSerialNumberEx constant to retrieve the serial number of a TOP if required by the calibration; identical to dpidTOPSerialEx
- const int coShutter = 0x00000001

  casGetOptions bit: device has a shutter; see Dark Current
- const int coFilter = 0x00000002

  casGetOptions bit: device has a density filter; see Density Filter
- const int coGetShutter = 0x00000004

  casGetOptions bit: device can check the position of the shutter (aka shutter control)
- const int coGetFilter = 0x00000008

  casGetOptions bit: device can check the position of the filter wheel (aka filter control)
- const int coGetAccessories = 0x00000010

  casGetOptions bit: device can retrieve serial number from additional equipment, see dpidAccessorySerial
- const int coGetTemperature = 0x00000020

  casGetOptions bit: device can measure CCD temperature, see mpidCurrentCCDTemperature
- const int coUseDarkcurrentArray = 0x00000040

  casGetOptions bit: option to enabled dark current array, see Dark Current
- const int coUseTransmission = 0x00000080

  casGetOptions bit: option to automatically apply a transmission correction when measuring as defined by dpid←TransmissionFileName
- const int coAutorangeMeasurement = 0x00000100

  casGetOptions bit: option to automatically detect good integration time, i.e. perform a AutoRange measurements
- const int coAutorangeFilter = 0x00000200

  casGetOptions bit: option to use density filter during AutoRange measurements
- const int coCheckCalibConfigSerials = 0x00000400

  casGetOptions bit: option to automatically check if config and calibration files are intended for this device; see Using Serial Numbers
- const int coTOPHasFieldOfViewConfig = 0x00000800

  casGetOptions bit: indication that configuration/calibration includes field of view information for the TOP, so mpidT←OPFieldOfView measurement condition is valid
- const int coAutoRemeasureDC = 0x00001000

  casGetOptions bit: option for turning automatic dark current measurements on/off. The interval after which the dark current is marked as invalid is defined by mpidRemeasureDCInterval
- const int coCanMultiTrack = 0x00008000

  casGetOptions bit: device supports MultiTrack measurements; depends on interface type
- const int coCanSwitchLEDOff = 0x00010000

  casGetOptions bit: device can switch off status LEDs during measurement (see option below)
- const int coLEDOffWhileMeasuring = 0x00020000

  casGetOptions bit: option for switching off status LEDs during measurements. Only supported if coCanSwitchLEDOff is present
- const int coCheckConfigUpToDate = 0x00040000

  casGetOptions bit: option to check whether the Ident-Key contains newer configuration/calibration files; happens during casInitialize and might result in ErrorConfigUpToDate
- const int coCanAverageOnDevice = 0x00080000

casGetOptions bit: device supports averaging on device, see mpidAveragingOnDevice

- const int coCanMultiTrackSensorTemp = 0x00100000

  casGetOptions bit: device supports sensor temperature for MultiTrack, see mpidCMTSensorTemp

- const int paPrepareMeasurement = 1

  casPerformActionEx ID: Prepares the spectrometer for the next measurement. This is intended to avoid wasting time within a measurement cycle and should be called before time-critical measurements are started. See chapter Performing a Measurement for other prerequisites.

- const int paLoadCalibration = 3

  casPerformActionEx ID: Reloads the calibration file. Might be useful after certain calibration parts have been modified with casClearCalibration or casSetCalibrationFactors.

- const int paCheckAccessories = 4

  casPerformActionEx ID: Performs a check whether the current calibration and config files are suitable for the used spectrometer and optional optical probes equipped with an ID Key. The ID key is re-read during the method call. See chapter Using Serial Numbers for more info.

- const int paMultiTrackStart = 5

  casPerformActionEx ID: Starts a MultiTrack measurement. The traditional way of starting MultiTrack with casStart is not supported by most interface types. See chapter MultiTrack measurements for an overview.

- const int paAutoRangeFindParameter = 7

  casPerformActionEx ID: performs AutoRange measurements to find suitable measurement parameter. See chapter AutoRange measurements for details.

- const int paCheckConfigUpToDate = 12

  casPerformActionEx ID: Performs a check whether the device has more recent configuration/calibration files. Returns ErrorConfigUpToDate if that is the case and files should be downloaded using dpidGetFilesFromDevice

- const int paSearchForDevices = 13

  casPerformActionEx ID: Starts a search for devices (i.e. interface options) for one or all interface types (see below). For some interface types like USB, this search is synchronous, for others like Ethernet, the search is started asynchronously. See chapter Interfaces Types and Options. This action does not require that you pass a valid CASID when calling casPerformActionEx. If you want to search only for a specific interface type, pass the interface type as AParam1 to casPerformActionEx - which may result in ErrorInvalidParameter if the interface ID is not valid. If AParam2 is 1, some interface types, like USB, perform a full reset and search twice, which may fix problems in console apps where CAS might not appear, which first need a firmware upload.

- const int paPrepareShutDown = 14

  casPerformActionEx ID: Stops all threads in the CAS library. Depending on how your app loads the library, it might be necessary to call this action from the main thread just before your app shuts down to avoid a dead-lock when the CAS library is unloaded. This action does not require that you pass a valid CASID when calling casPerformActionEx.

- const int paRecalcSpectrum = 16

  casPerformActionEx ID: Recalculates the spectrum from the raw data of the last measurement. This might be useful after you modified the calibration or something else that might influence the spectrum.

- const int paUpdateSpectralCalibration = 19

  casPerformActionEx ID: Recalculates the spectral calibration for all parameter sets. A typical reason to call this method, is when you modified the transmission directly via gcfTransmissionFunction. Whenever you change a measurement parameter that influences the spectral calibration (e.g. mpidNewDensityFilter), this action is called automatically.

- const int paUpdateCompleteCalibration = 20

  casPerformActionEx ID: Recalculates the X vector for the spectrum and resamples the transmission to the updated wavelength before finally recalculating the spectral calibration. This action should be called whenever the wavelength calibration was modified or when the transmission should be reloaded.

- const int paCheckAccessoriesTransmission = 21

  casPerformActionEx ID: Performs a check whether the current transmission file is suitable for the used spectrometer and optional optical probes equipped with an ID Key. The ID key is re-read during the method call. See chapter Using Serial Numbers for more info.

- const int paMeasureParamSets = 24

  casPerformActionEx ID: Performs measurements for several parameter sets. AParam1 = FirstParamSet, AParam2 = LastParamSet. This action only returns after all measurements have been done or if an error occured. See Working with Parameter Sets.

- const int paMeasureParamSetsDC = 25

casPerformActionEx ID: Performs DC measurements for several parameter sets. AParam1 = FirstParamSet, A↩ Param2 = LastParamSet. This action only returns after all DC measurements have been done or if an error occured. See Dark Current and Working with Parameter Sets.

- const int mpidIntegrationTime = 01

  float: the integration time in ms which will be used for the next measurement for the currently active parameter set. Valid values range from dpidIntTimeMin to dpidIntTimeMax. Note that after changing the integration time, a new dark current measurement might be necessary. See chapter Dark Current for an overview. Note that both mpidInt↩ TimeResolution and mpidIntTimeAlignPeriod will affect the actual integration time that is used when measuring. It is therefore important to verify and document mpidIntegrationTime after the measurement.

- const int mpidAverages = 02

  integer: the number of averages which will be used for the next measurement for the currently active parameter set. Valid values range from 1 to dpidAveragesMax. Note that after changing the integration time, a new dark current measurement might be necessary. See chapter Dark Current for an overview.

- const int mpidTriggerDelayTime = 03

  float: delay time in ms between trigger and start of the spectrum measurement. Valid values range from dpidTrigger↩ DelayTimeMin to dpidTriggerDelayTimeMax. Only applies to Triggered Measurements. dpidTriggerCapabilities must include tcoTriggerDelay for this to work.

- const int mpidTriggerTimeout = 04

  integer: timeout in ms which must not be exceeded when waiting for the external trigger or ErrorTriggerTimeout will be returned. Only applies to Triggered Measurements.

- const int mpidCheckStart = 05

  integer: first pixel which is included for checking the maximum value of the ADC during an acquisition, see mpidMax↩ ADCValue and mpidMaxADCPixel; leave at 0 to always use the first visible pixel of the calibration; see dpidDeadPixels; use casPixelToNm and casNmToPixel for conversion between pixels an wavelength

- const int mpidCheckStop = 06

  integer: last pixel which is included for checking the maximum value of the ADC during an acquisition, see mpid↩ MaxADCValue and mpidMaxADCPixel; leave at 0 to always use the last visible pixel of the calibration; see dpid↩ VisiblePixels; use casPixelToNm and casNmToPixel for conversion between pixels an wavelength

- const int mpidColormetricStart = 07

  lower bound in nm of the spectral range which is used for the colormetric calculation. If it is smaller than the wavelength of the first visible pixel, then this wavelength is used (i.e. the colormetric range can never exceed the range of the spectrum). Default value is 380nm. If you want to use the full range, set mpidColormetricStart and mpid↩ ColormetricStop both to 0. See also Getting the Spectrum and Results.

- const int mpidColormetricStop = 08

  upper bound in nm of the spectral range which is used for the colormetric calculation. If it is bigger than the wavelength of the last visible pixel, then this wavelength is used (i.e. the colormetric range can never exceed the range of the spectrum). Default value is 780nm. If you want to use the full range, set mpidColormetricStart and mpidColormetric↩ Stop both to 0. See also Getting the Spectrum and Results.

- const int mpidACQTime = 10

  integer: returns the time in ms the last acquisition took. This includes integration time for all averages as well as the time to read out the FIFO. The accuracy of this timing depends on the interface type, with PCI being the most accurate.

- const int mpidMaxADCValue = 11

  integer: returns the maximum value of the ADC during the last successful acquisition. Check this mpid for AD↩ C-overflow by comparing it with the dpidADCRange device parameter. Do not calculate a relative signal level using mpidMaxADCValue yourself, but query mpidRelSaturation instead! mpidMaxADCPixel returns the pixel at which this maximum occurred. The pixel range which is taken into account can be customized by mpidCheckStart and mpid↩ CheckStop. See also the section about validating a measurement in Performing a Measurement.

- const int mpidMaxADCPixel = 12

  integer: returns the pixel which had the maximum ADC value mpidMaxADCValue during the last successful acquisition. The pixel range which is taken into account can be customized by mpidCheckStart and mpidCheckStop. See also the section about validating a measurement in Performing a Measurement.

- const int mpidTriggerSource = 14

  integer: the current trigger source. Either trgSoftware for software triggering or trgFlipFlop when a hardware trigger is used. See chapter Triggered Measurements for an overview.

- const int mpidAmpOffset = 15

integer: returns the calculated amplifier offset of the last dark current measurement which has been performed with this device

- const int mpidSkipLevel = 16

  intensity below which spectral intensities are not taken into account during colormetric calculations (the so called SkipLevel). This maybe useful to suppress noise from affecting the colormetric calculation. Note that skip-leveling needs to be enabled with mpidSkipLevelEnabled.

- const int mpidSkipLevelEnabled = 17

  integer: determines whether skip-leveling is enabled (values <> 0). Set the actual intensity level using mpidSkipLevel

- const int mpidScanStartTime = 18

  integer: returns the value the internal timer had, when the last measurement was started. For Triggered Measurements the scan start is just before the spectrometer started accepting triggers, but not when the trigger actually arrived. See also mpidTimeSinceScanStart which might be useful for determining the timing at high-accuracy.

- const int mpidAutoRangeMaxIntTime = 19

  float: maximum integration time in milliseconds which must not be exceeded during an AutoRange measurements.

- const int mpidAutoRangeLevel = 20

  deprecated; use mpidAutoRangeMinLevel below

- const int mpidAutoRangeMinLevel = 20

  minimum relative maxADCValue between 0 and 100 percent which must be reached during an AutoRange measurements. See mpidAutoRangeMaxLevel for an upper limit.

- const int mpidDensityFilter = 21

  integer: returns previously set density filter or -1 if it never has been set. Use mpidCurrentDensityFilter to check the current physical filter position. When set, the filter wheel is immediately positioned. Valid values range from 0 to 7, otherwise ErrorParamOutOfRange will be returned. See chapter Density Filter for an overview and Performing a Measurement for prerequisites. mpidDensityFilter is not part of the parameter sets, but mpidNewDensityFilter is. Use casGetFilterName to translate a filter index into a filter name.

- const int mpidCurrentDensityFilter = 22

  integer: returns the current physical position of the density filter wheel, if the spectrometer supports reading the filter position (see coGetFilter option). If the device has no filter (i.e. coFilter option missing), -1 will be returned, otherwise the filter index between 0 and 7.

- const int mpidNewDensityFilter = 23

  integer: the filter wheel position between 0 and 7 which should be used for the next measurement of the active parameter set. See chapter Density Filter for an overview and Performing a Measurement for prerequisites. Reading mpidNewDensityFilter does not check the physical filter wheel position; use mpidCurrentDensityFilter for that.

- const int mpidLastDCAge = 24

  integer: returns the number of milliseconds that have passed, since the last Dark Current was measured for the active parameter set or -1 if no DC has been measured.

- const int mpidRelSaturation = 25

  returns the relative saturation (between 0 and 100%) of the previous successful measurement for the active parameter set. mpidRelSaturation can be checked against the dpidRelSaturationMin and dpidRelSaturationMax range, to ensure that the measurement has a sufficient signal level. However, to check for oversaturated measurements use mpidMaxADCValue and check it against dpidADCRange. Because with averaging, mpidRelSaturation might be $<$ 100% even though there were saturated spectra!

- const int mpidPulseWidth = 27

  If this parameter is non-zero, it will cause the spectrum to be corrected by the ratio of mpidPulseWidth divided by mpidIntegrationTime. This might be useful to correct spectra where the DUT is on for only a fraction of the integration time.

- const int mpidRemeasureDCInterval = 28

  integer: the time in ms, after which an automatic dark current measurement should be invalidated, so dpidDCRemeasureReasons becomes non-zero. This setting only has an effect, if the coAutoRemeasureDC option has been activated. The interval is checked against mpidLastDCAge. See chapter Dark Current for an overview.

- const int mpidFlashDelayTime = 29

  float: the delay time of the flash output signal in milliseconds. Applies only when flash is activated, i.e. mpidFlashType <> ftNone. If the delay time is negative, the flash signal and the delay happen before starting the measurement, otherwise the delay and the flash signal occur after the measurement started.

- const int mpidTOPAperture = 30

integer: the TOP aperture which should be taken into account when the calibration is applied to the spectrum. The TOP aperture parameter is a 0-based index ranging from 0 to 6 corresponding to the TOP apertures 1 to 7. This parameter doesn't actually adjust the aperture of the TOP, but controls which aperture calibration factor will be applied.

- const int mpidTOPDistance = 31

  the distance in mm from the DUT to the reference plane of the TOP. This parameter doesn't actually adjust the distance of the TOP, but controls which calibration factor will be applied.
- const int mpidTOPSpotSize = 32

  returns the size of the measurement spot for the current mpidTOPAperture and mpidTOPDistance.
- const int mpidTriggerOptions = 33

  integer: bit-set describing current trigger options. The set consists of the various to<XXX> bits, starting with to↩ AcceptOnlyWhenReady.
- const int mpidForceFilter = 34

  integer: flag which controls whether the filter wheel is moved for every measurement, even if it had been set to the same position previously. Internally the flag only affects dpidNeedDensityFilterChange, it will always return True if the ForceFilter flag is set.
- const int mpidFlashType = 35

  integer: the type of the flash signal which should be emitted during or before a measurement. This signal is a short pulse during the integration time which is typically used to trigger flash lamps. To check whether the spectrometer supports a flash signal in general, check the tcoFlashOutput flag in dpidTriggerCapabilities. Use mpidFlashOptions to control further options related to the flash. Possible values for mpidFlashType are all ft<XXX> constants, starting at ftNone.
- const int mpidFlashOptions = 36

  integer: bit-set describing the flash options of the device. Only applies, if flash is enabled, i.e. mpidFlashType is different from ftNone. The bits start with foEveryAverage.
- const int mpidACQStateLine = 37

  integer: number of the digital out port which should be used for the toShowACQState option in mpidTriggerOptions). The port specified by this parameter is set to the level given by mpidACQStateLinePolarity after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time. For possible port values refer to casSetDigitalOut and the hardware manual of the spectrometer.
- const int mpidACQStateLinePolarity = 38

  integer: level of the ACQ state line. Only applies if the toShowACQState option is enabled in mpidTriggerOptions. The value is 0 for low level, all other values indicate high level. The port specified by mpidACQStateLine is set to this level after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time.
- const int mpidBusyStateLine = 39

  integer: number of the digital out port which should be used for the toShowBusyState option in mpidTriggerOptions. The port specified by this parameter is set to the level given by mpidBusyStateLinePolarity after the spectrometer has been started and is ready to accept a trigger. The level is restored after the trigger has been received. For possible port values refer to casSetDigitalOut and the hardware manual of the spectrometer.
- const int mpidBusyStateLinePolarity = 40

  integer: level of the Busy state line. Only applies if the toShowBusyState option is enabled in mpidTriggerOptions. The value is 0 for low level, all other values indicate high level. The port specified by mpidACQStateLine is set to this level after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time.
- const int mpidAutoFlowTime = 41

  integer: returns the minimum flow time (in ms) the current source should switch on the DUT for, when it is triggered by either the busy or ACQ state line (mpidTriggerOptions). This time includes the mpidIntegrationTime for all mpid↩ Averages, a mpidTriggerDelayTime as well as a flash delay, if it is negative (see mpidFlashDelayTime). The read-out time of the CCD is not included. See chapter Synchronization, namely the section Trigger CAS sequence
- const int mpidCRIMode = 42

  integer: controls the way the CRI is calculated (see casCalculateCRI). This is a global setting which affects all devices! Possible values are criDIN6169, which is the default, and criCIE13_3_95.
- const int mpidObserver = 43

  integer: determines, which observer is used for casColorMetric. This is a global setting which affects all devices! Possible values are 2° cieObserver1931, which is the default, and 10° cieObserver1964.
- const int mpidTOPFieldOfView = 44

returns the field of view for the current mpidTOPAperture and mpidTOPDistance.

- const int mpidCurrentCCDTemperature = 46

  reading this mpid performs a temperature measurement and returns the CCD temperature in degrees Celsius. If the device does not support temperature measurements the return value will be NAN. To verify that the device supports temperature measurements, check the coGetTemperature flag in casGetOptions.

- const int mpidLastCCDTemperature = 47

  returns the previously measured CCD temperature in degrees Celsius. If the last temperature measurement is too old, a new temperature measurement will be performed. The interval which triggers a new temperature measurement is device dependent. If the device does not support temperature measurements the return value will be NAN. To verify that the device supports temperature measurements, check the coGetTemperature flag in casGetOptions.

- const int mpidDCCCDTemperature = 48

  returns the CCD temperature in degrees Celsius which was previously measured during the dark current measurement. If the dark current hasn't been measured or if the device does not support temperature measurements, the return value will be NAN. To verify that the device supports temperature measurements, check the coGetTemperature flag in casGetOptions.

- const int mpidAutoRangeMaxLevel = 49

  maximum relative maxADCValue between 0 and 100 percent which must not be exceeded during an AutoRange measurements. See mpidAutoRangeMinLevel for the lower limit.

- const int mpidMultiTrackAcqTime = 50

  integer: returns the acquisition time in milliseconds of the complete MultiTrack measurement series. The accuracy of the timing varies depending on the spectrometer interface type.

- const int mpidTimeSinceScanStart = 51

  integer: returns the time in milliseconds since the spectrometer was started, i.e. mpidScanStartTime. This mpid can replace calls to casStopTime, which has been deprecated.

- const int mpidCMTTrackStart = 52

  float: returns the time in milliseconds between the start of the MultiTrack measurement series and the start of the current track (which was previously set with casMultiTrackCopyData). For the first track this is typically 0, the second it would be mpidIntegrationTime, and so on.

- const int mpidColormetricWidthLevel = 54

  integer: level in % which should be used for the next calculation of casGetWidthEx. Default is 50% to calculate FWHM. Independent for each parameter set, see Working with Parameter Sets

- const int mpidIntTimeResolution = 55

  float in microseconds, resolution for mpidIntegrationTime. Supported values can be queried using dpidIntTimePossibleResolutions after the device has been initialized. Might return ErrorParamOutOfRange when set to an unsupported value. Attention: changing the mpidIntTimeResolution will affect dpidIntTimeMax, so mpidIntegrationTime needs to be checked for being inside the allowed range.

- const int mpidIntTimeAlignPeriod = 56

  float in ms: if non-zero positive, every measurement (also AutoRange, but not DC array measurements) uses an integration time which is a mulitple of this period. Check mpidIntegrationTime after the measurement for the actual integration time that was measured with.

- const int mpidColormetricType = 57

  integer: determines the type of colormetric calculations that should be performed. Default is cmtDefaultColormetric. Affects a wide range of calculation methods, e.g. casColorMetric, casGetCCT, casCalculateCRI and casCalculateLambdaDom. Note: the wavelength range covered by the photopic luminosity function V(Λ) and color matching functions x(Λ), y(Λ), z(Λ) depend on mpidColormetricType. For cmtDefaultColormetric the functions range from 380nm to 780nm and for cmtSWPColormetric they have a broader range of 360nm to 830nm. When using cmtSWPColormetric it is especially important to adjust mpidColormetricStart and mpidColormetricStop, because their historic default values of 380nm and 780nm would unexpectedly constrain the calculation of the affected results. Set both mpidColormetricStart and mpidColormetricStop to zero in order to use the full range of the spectrum for calculation.

- const int mpidAveragingOnDevice = 58

  integer: determines whether averaging happens on the device (values $<>$ 0) or in software, i.e. the CAS SDK. Only supported by some devices; check the coCanAverageOnDevice capability. If the device does not support averaging on device, getting and setting this mpid will return ErrorFeatureNotSupported. Averaging on device might be helpful for a large number of averages with very short integration times, as only the averaged spectrum is transfered. It should however be taken into account that with this option enabled, the ADC non-linearity correction is only performed on the averaged spectrum and not on each measured spectrum before they are averaged.

- const int mpidMeasured = 59

  integer (read-only): returns True (value <> 0) if the current parameter set was successfully measured. This flag is reset to False when a new measurement or DC measurement is started. When measuring single parameter sets, evaluating this flag is typically not necessary, as error checking the calls to casMeasure etc. is sufficient. When measuring multiple parameter sets with paMeasureParamSets or paMeasureParamSetsDC, all affected parameter sets get their measured flag reset immediately. This allows determing which spectra were correctly measured, even if an error occured later on in the call.

- const int mpidCMTSensorTemp = 61

  float (read-only): returns the sensor temperature in degree Celsius for the MultiTrack measurement series of the current track (which was previously set with casMultiTrackCopyData). Only devices returning coCanMultiTrack←SensorTemp support this, others will just return 0.

- const int mpidFlashDuration = 63

  float: the duration of the flash signal in millseconds. Minimum supported value is dpidFlashDurationMin, but only if dpidTriggerCapabilities contains tcoFlashHardwareDuration

- const int mpidColormetricPeakDiffWidth = 67

  float: the width in nm that is used for differentiation when calculating the peak wavelength casGetPeak. If negative, a width of 7nm or at least 6 pixels are used - like in previous versions. Default value is -1. Applies to the current CAS ID and the currently active ParamSet only!

- const int cmtDefaultColormetric = 0

  mpidColormetricType value constant: default CAS SDK colormetric like in CAS SDK version 4.9 and before

- const int cmtSWPColormetric = 100

  mpidColormetricType value constant: SWP colormetric like in SWP 3.6 and beyond. Note that SWP colormetric calculation might be taking more time than cmtDefaultColormetric due to the increased accuracy of the results.

- const int toAcceptOnlyWhenReady = 1

  mpidTriggerOptions bit: the hardware trigger is only accepted when the spectrometer is ready for acquisition, i.←e. the dpidLine1FlipFlop is reset before the spectrometer starts waiting for a trigger. This option flag can only be modified if tcoTriggerOnlyWhenReady is included in dpidTriggerCapabilities. Otherwise it is either enforced or cleared, depending on the spectrometer type.

- const int toForEachAutoRangeTrial = 2

  mpidTriggerOptions bit: for hardware triggered AutoRange measurement this flag controls, whether each acquisition during AutoRange requires a hardware trigger or only the first one. Since the number of acquisitions during Auto←Range can't be determined in advance, the spectrometer should be triggered whenever either the busy state or the ACQ state (see below) indicate that it is waiting for a trigger. Only supported if tcoAutoRangeTriggering flag is set in dpidTriggerCapabilities! Has no effect if mpidTriggerSource is not trgFlipFlop.

- const int toShowBusyState = 4

  mpidTriggerOptions bit: enables the busy signal. As soon as the spectrometer is ready to accept a trigger, the signal on the digital out port identified by mpidBusyStateLine changes to the level specified by mpidBusyStateLinePolarity. The signal is reset as soon as the trigger was received. Only supported if tcoShowBusyState flag is set in dpid←TriggerCapabilities!

- const int toShowACQState = 8

  mpidTriggerOptions bit: enables the acquisition signal. The digital out port identified by mpidACQStateLine changes to the level given by mpidACQStateLinePolarity as soon as the spectrometer is waiting for a trigger and is reset only after the acquisition has been finished, i.e. after integration and read-out time. Only supported if tcoShowACQState flag is set in dpidTriggerCapabilities!

- const int ftNone = 0

  mpidFlashType value constant: No flash signal is emitted. Default value and "supported" by all spectrometers.

- const int ftHardware = 1

  mpidFlashType value constant: The flash signal is controlled by the spectrometer. Refer to the hardware manual for details. Depending on the flag tcoFlashHardwareDelay and related flags, this might enable the mpidFlashDelayTime. Only supported if the tcoFlashHardware flag is set in dpidTriggerCapabilities.

- const int ftSoftware = 2

  mpidFlashType value constant: The flash signal is controlled by the CAS-DLL. Depending on the flag tcoFlash←SoftwareDelay and related flags, this might enable the mpidFlashDelayTime. Software flash in general is only supported if the tcoFlashSoftware flag is set in dpidTriggerCapabilities.

- const int foEveryAverage = 1

mpidFlashOptions bit: if set, the flash signal is emitted for each acquisition when mpidAverages is bigger than 1. Only applies to hardware flash and only supported by some devices (check tcoFlashHardwareForEachAverage in dpidTriggerCapabilities).

- const int trgSoftware = 0

  mpidTriggerSource value constant: no external trigger, casMeasure immediately starts the acquisition

- const int trgFlipFlop = 3

  mpidTriggerSource value constant: external trigger at PIN 1 (DB9) is used to start the acquisition. See chapter Triggered Measurements for a detailed overview on this topic.

- const int criDIN6169 = 0

  mpidCRIMode value constant: CRI calculation according to DIN 6169

- const int criCIE13_3_95 = 1

  mpidCRIMode value constant: CRI calculation according to CIE 13.3-95

- const int cieObserver1931 = 0

  mpidObserver value constant: 2° observer according to CIE 1931

- const int cieObserver1964 = 1

  mpidObserver value constant: 10° observer according to CIE 1964

- const int casShutterInvalid = -1

  return value of casGetShutter if current shutter state is invalid (device failure) or device has no shutter (check using casGetOptions)

- const int casShutterOpen = 0

  return value of casGetShutter if current shutter state is open, i.e. normal spectra will be measured

- const int casShutterClose = 1

  return value of casGetShutter if current shutter state is closed, i.e. dark current will be measured

- const int gcfDensityFunction = 0

  casGetCalibrationFactors AWhat: spectral calibration of the density filters

- const int gcfSensitivityFunction = 1

  casGetCalibrationFactors AWhat: spectral calibration

- const int gcfTransmissionFunction = 2

  casGetCalibrationFactors AWhat: transmission of additional optical hardware

- const int gcfDensityFactor = 3

  casGetCalibrationFactors AWhat: density filter factors

- const int gcfTOPApertureFactor = 4

  casGetCalibrationFactors AWhat: absolute calibration for TOP

- const int gcfTOPDistanceFactor = 5

  casGetCalibrationFactors AWhat: distance calibration factors for TOP

- const int gcfTDDistanceMin = -3

  casGetCalibrationFactors AIndex with AWhat=gcfTOPDistanceFactor to retrieve the minimum calibrated TOP distance

- const int gcfTDDistanceMax = -2

  casGetCalibrationFactors AIndex with AWhat=gcfTOPDistanceFactor to retrieve the maximum calibrated TOP distance

- const int gcfTDCount = -1

  casGetCalibrationFactors AIndex with AWhat=gcfTOPDistanceFactor to retrieve the number of TOP distance factors

- const int gcfTDExtraDistance = 1

  casGetCalibrationFactors AExtra with AWhat=gcfTOPDistanceFactor to retrieve a given TOP distance at AIndex

- const int gcfTDExtraFactor = 2

  casGetCalibrationFactors AExtra with AWhat=gcfTOPDistanceFactor to retrieve a given TOP distance factor at A↩ Index

- const int gcfWLCalibrationChannel = 6

  casGetCalibrationFactors AWhat: wavelength calibration channels, get/set with 0-based AIndex to get/set channels, i.e. Pixels with calibration points

- const int gcfWLExtraFirstCoefficient = -6 *casGetCalibrationFactors AExtra with AWhat=gcfWLCalibrationChannel to retrieve the first coefficient C0 of a polynom wavelength calibration; check for non-zero to find out whether polynom calibration is used.*

- const int gcfWLExtraLastCoefficient = -2

  casGetCalibrationFactors AExtra with AWhat=gcfWLCalibrationChannel to retrieve the last coefficient C4 of a polynom wavelength calibration

- const int gcfWLCalibPointCount = -1

  casGetCalibrationFactors AExtra with AWhat=gcfWLCalibrationChannel to retrieve the number of wavelength calibration points

- const int gcfWLExtraCalibrationDelete = 1

  casSetCalibrationFactors AExtra with AWhat=gcfWLCalibrationChannel: set to delete WL calibration point at AIndex

- const int gcfWLExtraCalibrationDeleteAll = 2

  casSetCalibrationFactors AExtra with AWhat=gcfWLCalibrationChannel: set to delete ALL calibration points

- const int gcfWLCalibrationAlias = 7

  casGetCalibrationFactors AWhat: wavelength calibration wavelengths, get/set with 0-based AIndex to get/set wavelengths for corresponding channels/pixels

- const int gcfWLCalibrationSave = 8

  casGetCalibrationFactors AWhat: set to any value, to update configuration file with current wavelength calibration

- const int gcfDarkArrayValues = 9

  casGetCalibrationFactors AWhat: get/set integration times and actual dark current spectra of dark current array

- const int gcfDarkArrayDepth = -1

  casGetCalibrationFactors AExtra with AWhat=gcfDarkArrayValues: get/set depth/length of dark current array

- const int gcfDarkArrayIntTime = -2

  casGetCalibrationFactors AExtra with AWhat=gcfDarkArrayValues: get/set integration time of dark current array at AIndex

- const int gcfTOPParameter = 11

  casGetCalibrationFactors AWhat: get various TOP parameter

- const int gcfTOPApertureSize = 0

  casGetCalibrationFactors AExtra with AWhat=gcfTOPParameter: get aperture size of TOP aperture AIndex

- const int gcfTOPSpotSizeDenominator = 1

  casGetCalibrationFactors AExtra with AWhat=gcfTOPParameter: get spot size distance denominator to calculate spot size from distance

- const int gcfTOPSpotSizeOffset = 2

  casGetCalibrationFactors AExtra with AWhat=gcfTOPParameter: get spot size distance offset to calculate spot size from distance

- const int gcfLinearityFunction = 12

  casGetCalibrationFactors AWhat: linearity calibration of the ADC

- const int gcfLinearityCounts = 0

  casGetCalibrationFactors AExtra with AWhat=casGetCalibrationFactors: get counts of linearity calibration array at AIndex

- const int gcfLinearityFactor = 1

  casGetCalibrationFactors AExtra with AWhat=casGetCalibrationFactors: get factor of linearity calibration array at AIndex

- const int gcfRawData = 14

  casGetCalibrationFactors AWhat: retrieve raw data of previously acquired spectrum, AIndex = channel/pixel

- const int cLambdaWidth = 0

  AWhat constant for casGetWidthEx: returns the peak width in nm, identical to casGetWidth

- const int cLambdaLow = 1

  AWhat constant for casGetWidthEx: start wavelength of the width.

- const int cLambdaMiddle = 2

  AWhat constant for casGetWidthEx: center wavelength of the width, i.e. Center - Start = Stop - Center.

- const int cLambdaHigh = 3

  AWhat constant for casGetWidthEx: stop wavelength of the width.

- const int cLambdaOuterWidth = 4

AWhat constant for casGetWidthEx: identical to cLambdaWidth, but using old algorithm.

- const int cLambdaOuterLow = 5

  AWhat constant for casGetWidthEx: identical to cLambdaLow, but using old algorithm.
- const int cLambdaOuterMiddle = 6

  AWhat constant for casGetWidthEx: identical to cLambdaMiddle, but using old algorithm.
- const int cLambdaOuterHigh = 7

  AWhat constant for casGetWidthEx: identical to cLambdaHigh, but using old algorithm.
- const int ecvVisualEffect = 2

  AWhat constant for casGetExtendedColorValues: Visual Effect in %. Tristimulus Y divided by radiometric VIS integral (380..780nm)
- const int ecvUVA = 3

  AWhat constant for casGetExtendedColorValues: UVA radiometric integral 315..400 nm.
- const int ecvUVB = 4

  AWhat constant for casGetExtendedColorValues: UVB radiometric integral 280..315 nm.
- const int ecvUVC = 5

  AWhat constant for casGetExtendedColorValues: UVC radiometric integral 200..280 nm.
- const int ecvVIS = 6

  AWhat constant for casGetExtendedColorValues: VIS radiometric integral 380..780 nm.
- const int ecvCRICCT = 7

  AWhat constant for casGetExtendedColorValues: correlated color temperature in K used for CRI calculation. Not identical to casGetCCT!
- const int ecvCDI = 8

  AWhat constant for casGetExtendedColorValues: color discrimination index.
- const int ecvDistance = 9

  AWhat constant for casGetExtendedColorValues: distance between the color locus (u, v) and the Planck curve.
- const int ecvCalibMin = 10

  AWhat constant for casGetExtendedColorValues: minimum wavelength of the currently used calibration.
- const int ecvCalibMax = 11

  AWhat constant for casGetExtendedColorValues: maximum wavelength of the currently used calibration.
- const int ecvScotopicInt = 12

  AWhat constant for casGetExtendedColorValues: scotopic integral, unit identical to photometric integral retrieved by casGetPhotInt
- const int ecvCRIFirst = 100

  AWhat constant for casGetExtendedColorValues: retrieves first (common) CRI, increase up to ecvCRIList to retrieve others.
- const int ecvCRILast = 116

  AWhat constant for casGetExtendedColorValues: retrieves last CRI, see ecvCRIFirst.
- const int ecvCRITriKXFirst = 120

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.
- const int ecvCRITriKXLast = 136

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.
- const int ecvCRITriKYFirst = 140

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.
- const int ecvCRITriKYLast = 156

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.
- const int ecvCRITriKZFirst = 160

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.
- const int ecvCRITriKZLast = 176

AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRXordUFirst = 180

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRXordULast = 196

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRYordVFirst = 200

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRYordVLast = 216

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRZordWFirst = 220

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int ecvCRITriRZordWLast = 236

  AWhat constant for casGetExtendedColorValues: intermediate result of CRI calculation. See overview table at cas←GetExtendedColorValues.

- const int extNoError = 0

  AWhat constant for casSetStatusLED

- const int extExternalError = 1

  AWhat constant for casSetStatusLED

- const int extFilterBlink = 2

  AWhat constant for casSetStatusLED

- const int extShutterBlink = 4

  AWhat constant for casSetStatusLED

8.1.1 Detailed Description

The CAS DLL interface class

This static class provides access to all methods the CAS DLL exports as well as defining all the constants that are used for calling the methods or interpreting the results. Even if you're accessing the CAS DLL directly and not via this interface class, you can still use this documentation as a reference for the available methods and constants.

8.1.2 Member Function Documentation

8.1.2.1 casAssignDeviceEx()

```
static int InstrumentSystems.CAS4.CAS4DLL.casAssignDeviceEx (
            int ASourceDevice,
            int ADestDevice,
            int AOption )  [static]
```

Assigns properties and or parameters from one device handle to another

Parameters

ASourceDevice	Source device
ADestDevice	Destination device
AOption	Controls which aspects get assigned from ASourceDevice to ADestDevice

Returns

This method returns 0 if the assignment was successful. Otherwise it returns an Error Code

Assigning devices might be useful if you want to store different measurement setups for a given device. By keeping copies of all parameters with a second device handle it is possible to store measurement setups which may even differ by calibration or dark current, something which cannot be achieved using parameter sets.

8.1.2.2 casCalculateCorrectedData()

```
static void InstrumentSystems.CAS4.CAS4DLL.casCalculateCorrectedData (
            int ADevice )  [static]
```

Applies the spectral correction to the previously acquired raw spectrum.

Parameters

ADevice	The device / CASID

This method causes the spectral correction to be applied to the previously acquired raw spectrum but without a transmission correction defined by dpidTransmissionFileName! This is done automatically after a measurement (but taking the transmission correction into account if applicable), so normally there is no need to call this method explicitly.

If you want to apply the transmission correction as well, use casConvoluteTransmission instead.

Note

This method call is useful if for one spectrum, you want the same spectrum with and without the transmission correction applied.

8.1.2.3 casCalculateCRI()

```
static int InstrumentSystems.CAS4.CAS4DLL.casCalculateCRI (
            int ADevice )  [static]
```

Calculates the color rendering indices (CRI) of a previously measured spectrum.

Parameters

ADevice	The device / CASID

Returns

The return value is 0, if the CRI calculation was successful or a negative error code. Most likely is ErrorCRI, which can indicate that there's not enough signal to calculate CRI or that the CCT is outside a supported range.

Since different standards for calculating CRI are supported, use the global mpidCRIMode to select the standard before calling casCalculateCRI.

Note

casGetCCT must have been called before calling casCalculateCRI, because the CRI calculation uses a special CCT (ecvCRICCT).

To retrieve the actual color rendering indices, use casGetCRI.

[Delphi]

```
...
casColorMetric(CASID);
CheckError(casGetError(CASID));
CCT:= casGetCCT(CASID); //also calculates CRICCT and stores it internally
CheckError(casGetError(CASID));
CheckError(casCalculateCRI(CASID));
CRIRa:= casGetCRI(CASID, 0); //Index = 0 retrieves common CRI
CheckError(casGetError(CASID));
...
```

8.1.2.4 casCalculateLambdaDom()

```
static int InstrumentSystems.CAS4.CAS4DLL.casCalculateLambdaDom (
            int ADevice,
            double IllX,
            double IllY,
            ref double LambdaDom,
            ref double Purity )  [static]
```

Calculates dominant wavelength aka LambdaDom and purity for the specified illuminant reference and the color coordinates that have been calculated by casColorMetric. This method replaces the now deprecated cmXYTo↵ DominantWavelength and does take the new mpidColormetricType into account.

Parameters

ADevice	The device / CASID
IllX	The x color coordinate of the illuminant reference
IllY	The y color coordinate of the illuminant reference
LambdaDom	The variable which will receive the calculated dominant wavelength
Purity	The variable which will receive the calculated purity

Returns

The return value is 0, if the calculation was successful, or a negative error code (see chapter Error Handling).

The device must have a valid spectrum and casColorMetric must have been called successfully. IllX and IllY are the color coordinates of the illuminant reference. One of the typical references is Illuminant E where x and y are 0.3333. Purity is 0 if x = IllX and y = IllY. Purity will never exceed 1.

8.1.2.5 casCalculateTOPParameter()

```
static int InstrumentSystems.CAS4.CAS4DLL.casCalculateTOPParameter (
            int ADevice,
            int AAperture,
```

```
double ADistance,
ref double ASpotSize,
ref double AFieldOfView )  [static]
```

Calculates spot size and field of view of the TOP.

Parameters

ADevice	The device / CASID
AAperture	The 0-based index of the TOP aperture, e.g. mpidTOPAperture
ADistance	The TOP distance in mm, e.g. mpidTOPDistance
ASpotSize	Variable which will receive the calculated spot size in mm.
AFieldOfView	Variable which will receive the calculated field of view in °.

Returns

The return value is 0 if the method was successful (even if coTOPHasFieldOfViewConfig is not present!). Otherwise the return value is an error code.

This method calculates the spot size and field of view of the TOP for the given aperture and distance and returns them in the ASpotSize and AFieldOfView parameter.

```
//this demo displays the FieldOfView in ° and SpotSize in mm for the current TOP aperture
...
var
  lSpotSize, lFieldOfView: Double;
...
if casGetOptions(CASID) and coTopHasFieldOfViewConfig <> 0 then
begin
  CheckError(casCalculateTOPParameter(CASID, CurrentTOPAperture, CurrentTOPDistance, lSpotSize,
       lFieldOfView));
  lblFieldOfView.Caption:= FormatFloat('0.## °', lFieldOfView);
  lblSpotSize.Caption:= FormatFloat('0.## mm', lSpotSize);
end else
begin
  lblFieldOfView.Caption:= 'No Field Of View Info';
  lblSpotSize.Caption:= 'N/A';
end;
```

8.1.2.6 casChangeDevice()

```
static int InstrumentSystems.CAS4.CAS4DLL.casChangeDevice (
              int ADevice,
              int AInterfaceType,
              int AInterfaceOption )  [static]
```

Change the interface type and/or option of a device / CASID

Parameters

ADevice	The device / CASID
AInterfaceType	New interface type
AInterfaceOption	New interface option

Returns

The CASID of the changed device or a negative error like ErrorInvalidDeviceType or errCasDeviceNotFound

Using this method it is possible to change the interface type and option of a device created with casCreateDeviceEx. Refer to the chapter Interfaces Types and Options for an overview about interface types and options.

8.1.2.7 casClearCalibration()

```
static void InstrumentSystems.CAS4.CAS4DLL.casClearCalibration (
            int ADevice,
            int AWhat )  [static]
```

Clears the specified calibration part of the given device.

Parameters

ADevice	The device / CASID
AWhat	One of the gcf<XXX> constants, specifying which calibration part to delete. See table below for possible values.

The following table lists the possible values for AWhat

AWhat	Description
gcfDensityFunction	Spectral calibration of all density filters
gcfSensitivityFunction	Spectral calibration of the spectrometer
gcfTransmissionFunction	Transmission correction of additional optical accessories
gcfTOPApertureFactor	Absolute calibration of the TOP apertures
gcfTOPDistanceFactor	TOP distance factors
gcfWLCalibrationChannel	Clears the wavelength calibration
gcfLinearityFunction	Linearity calibration of the ADC

There is no need to call casUpdateCalibrations after casClearCalibration, as this is done implicitly.

8.1.2.8 casClearDarkCurrent()

```
static int InstrumentSystems.CAS4.CAS4DLL.casClearDarkCurrent (
            int ADevice )  [static]
```

Clears a previously measured dark current of the given device

Parameters

ADevice	The device / CASID

Returns

The return value is 0 if no error occurred, otherwise it is one of the error codes.

Normally there is no need to call this method since a new dark current measurement (see casMeasureDarkCurrent) automatically replaces the previous one. This method only affects the dark current of the current parameter set (see Working with Parameter Sets). Refer to chapter Dark Current for an overview over related methods and dpid's.

8.1.2.9 casColorMetric()

```
static int InstrumentSystems.CAS4.CAS4DLL.casColorMetric (
             int ADevice )   [static]
```

Calculates colormetric results for the previously measured spectrum.

Parameters

ADevice	The device / CASID

Returns

The return value is 0, if the method was successful, or a negative error code (see chapter Error Handling).

The spectral range taken into account for the calculation is defined by mpidColormetricStart and mpidColormetric←Stop. The calculation is also influenced by mpidObserver. Refer to the chapter Getting the Spectrum and Results for an overview of the methods (and the order they have to be called) to retrieve the results of this calculation.

8.1.2.10 casConvoluteTransmission()

```
static void InstrumentSystems.CAS4.CAS4DLL.casConvoluteTransmission (
             int ADevice )   [static]
```

Applies the spectral and transmission correction to the previously acquired raw spectrum.

Parameters

ADevice	The device / CASID

This method causes the spectral and transmission correction to be applied to the previously acquired raw spectrum! This is done automatically after a measurement if the coUseTransmission option is enabled, so normally there is no need to call this method explicitly. The file containing the transmission correction must have been set using dpidTransmissionFileName before the device was initialized with casInitialize!

If you don't want to apply the transmission correction as well, use casCalculateCorrectedData instead.

Note

This method call is useful if for one spectrum, you want the same spectrum with and without the transmission correction applied.

8.1.2.11 casCreateDevice()

```
static int InstrumentSystems.CAS4.CAS4DLL.casCreateDevice ( )   [static]
```

Deprecated method. Creates a device context aka CASID within the CAS DLL

Returns

If the function was successful, the return value is a device handle (>=0). A negative value indicates an error

Use casCreateDeviceEx instead

8.1.2.12 casCreateDeviceEx()

```
static int InstrumentSystems.CAS4.CAS4DLL.casCreateDeviceEx (
            int AInterfaceType,
            int AInterfaceOption )  [static]
```

Creates a device context aka CASID within the CAS DLL

Parameters

AInterfaceType	Type of the interface, i.e. the way the spectrometer is connected to the PC
AInterfaceOption	The interface option, identifying either the spectrometer itself or the interface card

Returns

If the function was successful, the return value is a device handle (>=0). A negative value indicates an error, e.g. -3 = ErrorInvalidDeviceType

This method must be called to work with a spectrometer. It returns a device handle which is later used with subsequent function calls to identify a specific device (typically called the ADevice parameter, also referred to as CASID).

For every device handle created, you should call casDoneDevice once it is no longer needed.

For interface type and option, there are some predefined interface constants (like InterfaceUSB), but it is generally recommended to enumerate interface types and options and let the user choose from a set of two lists. Refer to chapter Interfaces Types and Options for a detailed overview.

Note

casCreateDeviceEx does not perform hardware initialization. Use casInitialize for that.

8.1.2.13 casDeleteParamSet()

```
static int InstrumentSystems.CAS4.CAS4DLL.casDeleteParamSet (
            int ADevice,
            int AParamSet )  [static]
```

Deletes a specific parameter set for the given device.

Parameters

ADevice	The device / CASID
AParamSet	The 0-based index of the parameter set which should be deleted

Returns

The return value is 0 if no error occurred, otherwise it is one of the error codes, e.g. ErrorSelectParamSet

AParamSet may range from 0 to dpidParamSets - 1. If the dpidCurrentParamSet is deleted, the first parameter set becomes active. There must be at least one parameter set, so casDeleteParamSet cannot delete the last remaining one. Refer to chapter Working with Parameter Sets for an overview.

8.1.2.14 casDoneDevice()

```
static int InstrumentSystems.CAS4.CAS4DLL.casDoneDevice (
            int ADevice )  [static]
```

Release resources used by the device

Parameters

ADevice	The device / CASID

Returns

The method returns 0 if finalization of the device was successful, otherwise an error code

Call this method for every device which was created with casCreateDeviceEx. Note that after a successful call to this method, the CASID ADevice is no longer valid!

8.1.2.15 casFIFOHasData()

```
static int InstrumentSystems.CAS4.CAS4DLL.casFIFOHasData (
            int ADevice )  [static]
```

Use this method to check if the spectrometer has data available which can be read

Parameters

ADevice	The device / CASID

Returns

The return value is 0 if the spectrometer is still performing the acquisition, and 1 if FIFO data is ready to be read. Negative return values indicate an error which can be translated into an error message using casGet↩ErrorMessage

Calling casFIFOHasData is only necessary if a measurement was started with casStart. It allows to check if the integration time of the CCD is over.

Note

Not all spectrometer types do support this - check dpidTriggerCapabilities, namely the tcoQueryHasData bit.

8.1.2.16 casGetCalibrationFactors()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetCalibrationFactors (
            int ADevice,
            int AWhat,
            int AIndex,
            int AExtra )  [static]
```

Returns various details about the configuration/calibration of the given device.

Parameters

ADevice	The device / CASID
AWhat	What calibration part to retrieve: one of the gcf<XXX> constants, e.g. gcfDensityFunction.
AIndex	For calibration parts which consist of lists or spectra, this is the index or pixel for which to retrieve the calibration information. See the documentation for the gcf<XXX> constant used as AWhat.
AExtra	Additional constant to indicate what calibration info to retrieve. Often another gcf<XXX> constant, e.g. gcfDarkArrayDepth.

Returns

The return value is the calibration information identified by the parameter which were passed.

The following table lists the possible values for AIndex and AExtra for the different AWhat constants.

AWhat	AIndex	AExtra	Description
gcfDensityFunction	0..Pixel-1	Density Filter (0..7)	Spectral calibration of the density filters
gcfSensitivityFunction	0..Pixel-1	Unused	Spectral calibration of the CAS
gcfTransmissionFunction	0..Pixel-1	Unused	Transmission of additional optical hardware
gcfDensityFactor	0..7	Unused	Absolute calibration for the density filters
gcfTOPApertureFactor	0..6	Unused	Absolute calibration factor for TOP, aperture given by Index. Check against 0 to find out, which TOP apertures are calibrated.
gcfTOPDistanceFactor	gcfTDDistanceMin	Unused	The minimum calibrated TOP distance
	gcfTDDistanceMax	Unused	The maximum calibrated TOP distance
	gcfTDCount	Unused	Number of distance and factor pairs
	0..n	gcfTDExtraDistance	TOP distance in mm
		gcfTDExtraFactor	TOP calibration factor for this distance
gcfWLCalibrationChannel	gcfWLCalibPointCount	Unused	Number of calibration points of the wavelength calibration
	0..CalibPointCount-1	Unused	Wavelength calibration: pixel of the CalibPoint specified by Index

gcfWLCalibrationAlias	0..CalibPointCount-1	Unused	Wavelength calibration↩ : wavelength of the CalibPoint specified by Index
gcfDarkArrayValues	unused	gcfDarkArrayDepth	Depth of the dark array, i.e. the number of dark currents in the array
	0..DarkArrayDepth-1	gcfDarkArrayIntTime	The integration time of the dark current measurement identified by Index.
	0..Pixel-1	0..DarkArrayDepth-1	The intensity of the dark current at the pixel given by Index and the dark current measurement identified by Extra.
gcfTOPParameter	0..6	gcfTOPApertureSize	Size of the TOP aperture in mm
	Unused	gcfTOPSpotSize↩ Denominator	Denominator for calculation of the spot size of the TOP, rather use cas↩ CalculateTOPParameter
	Unused	gcfTOPSpotSizeOffset	Offset for calculation of the spot size of the T↩ OP, rather use cas↩ CalculateTOPParameter
gcfLinearityFunction	0..n	gcfLinearityCounts	ADC counts for linearity correction
		gcfLinearityFactor	Correction Factor for A↩ DC range
gcfRawData	0..Pixel - 1	Unused	Spectral raw data, normalized to 1 ms and 1 average

Some of these gcf<XXX> constants are also used by casClearCalibration.

Warning

Since this method returns the actual calibration information, error handling has to be done using casGet↩ Error calls. Typical error codes are ErrorInvalidParameter and ErrorParamOutOfRange. When enumerating some calibration information lists, there might be no other way than increasing the AIndex parameter until ErrorParamOutOfRange occurs.

8.1.2.17 casGetCCT()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetCCT (
            int ADevice )  [static]
```

Calculates the correlated color temperature CCT of a previously measured spectrum.

Parameters

ADevice	The device / CASID

Returns

The return value is the CCT in K. Use casGetError to check for errors.

Before calling casGetCCT, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

As with all colormetric calculations, the range of the spectrum which is used for calculating the CCT is defined by mpidColormetricStart and mpidColormetricStop. The calculation is also affected by mpidObserver.

Use casGetError to check for errors.

8.1.2.18 casGetCentroid()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetCentroid (
            int ADevice )  [static]
```

Retrieves the previously calculated centroid wavelength.

Parameters

ADevice	The device / CASID

Returns

The return value is the centroid wavelength in nm. Use casGetError to check for errors.

Before calling casGetCentroid, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

8.1.2.19 casGetColorCoordinates()

```
static void InstrumentSystems.CAS4.CAS4DLL.casGetColorCoordinates (
            int ADevice,
            ref double x,
            ref double y,
            ref double z,
            ref double u,
            ref double v1976,
            ref double v1960 )  [static]
```

Retrieves CIE color coordinates of a previously measured spectrum.

Parameters

ADevice	The device / CASID
x	Receives the x color coordinate according to CIE 1931
y	Receives the y color coordinate according to CIE 1931
z	Receives the z color coordinate according to CIE 1931
u	Receives the u / u' color coordinate according to CIE 1960 / CIE 1976
v1976	Receives the v' color coordinate according to CIE 1976
v1960	Receives the v color coordinate according to CIE 1960

Before calling casGetColorCoordinates, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

As with all colormetric calculations, the range of the spectrum which is used for calculating the color coordinates is defined by mpidColormetricStart and mpidColormetricStop. The calculation is also affected by mpidObserver.

Use casGetError to check for errors.

8.1.2.20 casGetCRI()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetCRI (
            int ADevice,
            int Index )  [static]
```

Retrieves one of the previsouly calculated the color rendering indices.

Parameters

ADevice	The device / CASID
Index	Index ranging from 0 to 16, identifying which index should be returned. 0 returns the common index, which is the average of the indices 1..16.

Returns

The return value is the CRI. Use casGetError to check for errors.

The following preconditions must have been met, before calling casGetCRI:

- the device must have a valid spectrum
- set CRI calculation mode globally using mpidCRIMode
- colormetric calculation using casColorMetric)
- calculation of the CCT using (casGetCCT)
- calculation of the CRI using (casCalculateCRI)

8.1.2.21 casGetDarkCurrent()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetDarkCurrent (
            int ADevice,
            int AIndex )  [static]
```

Returns the intensity of the previously measured/calculated dark current for the given pixel.

Parameters

ADevice	The device / CASID
AIndex	The 0-based index of the pixel in the complete pixel array for which the dark current should be returned

Returns

The return value is the dark current for the given pixel in negative calibration units. Use casGetError to check for ErrorInvalidParameter and other errors.

AIndex can range from 0 to dpidPixels - 1, but the valid dark current ranges from dpidDeadPixels to dpidDeadPixels + dpidVisiblePixels - 1.

Use casGetXArray to retrieve the corresponding wavelength and casGetData to retrieve the intensity of a measured spectrum.

8.1.2.22 casGetData()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetData (
            int ADevice,
            int AIndex )  [static]
```

Returns the intensity of the previously acquired spectrum for the given pixel.

Parameters

ADevice	The device / CASID
AIndex	The 0-based index of the pixel in the complete pixel array for which the intensity should be returned

Returns

The return value is the intensity for the given pixel in the calibration unit dpidCalibrationUnit. Use casGetError to check for ErrorInvalidParameter and other errors.

AIndex can range from 0 to dpidPixels - 1, but the valid calibrated spectrum ranges from dpidDeadPixels to dpidDeadPixels + dpidVisiblePixels - 1.

Use casGetXArray to retrieve the corresponding wavelength. A previously measured dark current can be retrieved using casGetDarkCurrent.

[Visual Basic]

```
DeadPixels = casGetDeviceParameter(CasID, dpidDeadPixels)
VisiblePixels = casGetDeviceParameter(CasID, dpidVisiblePixels)

'get Lambda and Intensity, skip dead pixels!
For i = 0 To VisiblePixels - 1
  Spectrum(0, i) = casGetXArray(CasID, DeadPixels + i)
  Spectrum(1, i) = casGetData(CasID, DeadPixels + i)
Next i
```

8.1.2.23 casGetDeviceParameter()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetDeviceParameter (
            int ADevice,
            int AWhat )  [static]
```

Retrieves a float representing a device parameter

Parameters

ADevice	The device / CASID
AWhat	dpid constant identifying which device parameter to retrieve

Returns

Either an error code or a float whose meaning depends on the AWhat parameter. Refer to the documentation of the dpid constant used for AWhat.

8.1.2.24 casGetDeviceParameterString()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDeviceParameterString (
            int ADevice,
            int AWhat,
            StringBuilder ADest,
            int AMaxLen )  [static]
```

Retrieves a string representing a device parameter

Parameters

ADevice	The device / CASID
AWhat	dpid constant identifying which device parameter to retrieve
ADest	StringBuilder/PAnsiChar buffer to retrieve the string
AMaxLen	number of characters ADest can hold - including a terminating zero

Returns

0 if successful or an error code

8.1.2.25 casGetDeviceTypeName()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetDeviceTypeName (
            int AInterfaceType,
            StringBuilder ADest,
            int AMaxLen )  [static]
```

Retrieves the name of the given interface type

Parameters

AInterfaceType	The interface type for which to retrieve the name
ADest	Destination string for the interface type name
AMaxLen	The maximum number of characters ADest can hold

Returns

The return value should not be used!

Use this method when iterating over all interface names. For AInterfaceType use constants from 0 to casGetDeviceTypes - 1.

Note

If an empty string is returned for an interface name, this interface type should not be used nor presented to the user!

For more details on interface types and options, refer to chapter Interfaces Types and Options

8.1.2.26 casGetDeviceTypeOption()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDeviceTypeOption (
            int AInterfaceType,
            int AIndex )  [static]
```

Returns the value of the interface option for the given interface type and option index

Parameters

AInterfaceType	The interface type for which to retrieve the option
AIndex	0-based index of the option

Returns

Returns an interface option. Do not interpret as an error code!

Use this method when iterating over all interface options. AIndex can range from 0 to casGetDeviceTypeOptions - 1. The returned interface option can be used when calling casCreateDeviceEx or casChangeDevice.

For more details on interface types and options, refer to chapter Interfaces Types and Options

8.1.2.27 casGetDeviceTypeOptionName()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetDeviceTypeOptionName (
            int AInterfaceType,
            int AInterfaceOptionIndex,
            StringBuilder ADest,
            int AMaxLen )  [static]
```

Retrieves the name of the given interface option

Parameters

AInterfaceType	The interface type for which to retrieve the interface option name
AInterfaceOptionIndex	The 0-based index of the interface option for which to retrieve the name
ADest	Destination string for the interface option name
AMaxLen	The maximum number of characters ADest can hold

Returns

The return value should not be used!

Use this method when iterating over all interface options. For AInterfaceOptionIndex use constants from 0 to cas←GetDeviceTypeOptions - 1.

For more details on interface types and options, refer to chapter Interfaces Types and Options

8.1.2.28 casGetDeviceTypeOptions()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDeviceTypeOptions (
            int AInterfaceType )  [static]
```

Retrieves the number of options a given interface type currently supports

Parameters

AInterfaceType	The interface type for which to retrieve the number of options

Returns

Returns a negative error code like ErrorInvalidParameter or a number indicating how many interface options the given interface type supports

Use this method when iterating over interface options using casGetDeviceTypeOption

For more details on interface types and options, refer to chapter Interfaces Types and Options

8.1.2.29 casGetDeviceTypes()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDeviceTypes ( )  [static]
```

Retrieves the number of interface types the CAS DLL supports

Returns

Returns a positive number indicating how many interface types are supported

Use this method when iterating over all interface names using casGetDeviceTypeName

For more details on interface types and options, refer to chapter Interfaces Types and Options

8.1.2.30 casGetDigitalIn()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDigitalIn (
            int ADevice,
            int APort )  [static]
```

Returns the state of the specified digital input port.

Parameters

ADevice	The device / CASID
APort	Identifies which DigitalIn port state to retrieve. The possible values differ with the spectrometer type and interface - refer to the hardware manual for more information.

Returns

The return value is 0 for low and 1 for high state. If APort is invalid for this device, ErrorParamOutOfRange will be returned. Other error codes are possible

8.1.2.31 casGetDigitalOut()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetDigitalOut (
            int ADevice,
            int APort )  [static]
```

Returns the state the specified digital output port was set to by a former call to casSetDigitalOut.

Parameters

ADevice	The device / CASID
APort	Identifies which DigitalOut port state to retrieve. The possible values differ with the spectrometer type and interface - refer to the hardware manual for more information.

Returns

The return value is 0 for low and 1 for high state. If APort is invalid for this device, ErrorParamOutOfRange will be returned. Other error codes are possible

APort=1 is pin 8 and APort=2 for pin 7 of the DB9 connector of the PCI/ISA card or the trigger connector on the rear panel of the CAS140CT with USB interface.

8.1.2.32 casGetDLLFileName()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetDLLFileName (
            StringBuilder Dest,
            int AMaxLen )  [static]
```

Retrieves the complete path and file name of CAS library

Parameters

Dest	A buffer which can hold at least the number of characters specified in AMaxLen
AMaxLen	Number of characters including a trailing zero that Dest can hold

Returns

The return value is the pointer passed in Dest and can be ignored

There are A and W overloads for ANSI and Unicode versions of the method. The returned string is null-terminated.

8.1.2.33 casGetDLLVersionNumber()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetDLLVersionNumber (
            StringBuilder Dest,
            int AMaxLen )  [static]
```

Retrieves the version of CAS library

Parameters

Dest	A buffer which can hold at least the number of characters specified in AMaxLen
AMaxLen	Number of characters including a trailing zero that Dest can hold

Returns

The return value is the pointer passed in Dest and can be ignored

There are A and W overloads for ANSI and Unicode versions of the method. The returned string is null-terminated.

Note

some platforms might not support version info and will return "N/A".

8.1.2.34 casGetError()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetError (
            int ADevice )  [static]
```

Return error code for a given device/CASID

Parameters

ADevice	The device / CASID

Some methods provided by the CAS DLL do not return error codes. Call casGetError afterwards to check for errors which might have occurred during these method calls. A negative value indicates an error. ErrorNoError indicates that the previous action was successful. See chapter Error Handling for more details.

Note

Every subsequent call into the CAS DLL will clear the previous error for the given device! casGetErrorMessage can be used to translate the returned error into an error message.

8.1.2.35 casGetErrorMessage()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetErrorMessage (
            int AError,
            StringBuilder ADest,
            int AMaxLen )  [static]
```

Translates a given error code into a readable error message

Parameters

AError	The error code which should be translated
ADest	Destination for the error message
AMaxLen	The maximum number of characters ADest can hold

Returns

The return value should not be used

Use this method to translate an error constant (AError) into a user-readable error message. The error constant can either be the return value of a method of CAS DLL or can be explicitly retrieved using casGetError. The message is copied into the buffer ADest is pointing to. This buffer must be able to hold at least the number of characters specified by AMaxLen plus a trailing zero.

8.1.2.36 casGetExtendedColorValues()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetExtendedColorValues (
            int ADevice,
            int AWhat )  [static]
```

Retrieves results and conditions for previously performed colormetric calculations.

Parameters

ADevice	The device / CASID
AWhat	Determines which colormetric value to return. One of the ecv constants starting with ecvVisualEffect

Returns

The return value depends on the ecv constant passed as What parameter. Use casGetError to check for errors.

Before calling casGetExtendedColorValues the values have to be calculated by calling casColorMetric, casGetCCT and casCalculateCRI. Refer to these methods to find out which other properties influence their results, like colormetric range or mpidCRIMode.

The ecv constants starting with ecvCRIFirst allow access to intermediate- and end-results of the CRI calculation. The following table lists their meaning when mpidCRIMode is criDIN6169 and mpidColormetricType is cmtDefaultColormetric. For cmtSWPColormetric these results are not available and will return 0.

AWhat range	Description
ecvCRIFirst..ecvCRILast	Returns the CRI value, identical to casGetCRI, i.e. AWhat = ecvCRIFirst returns the common CRI, ecvCRIFirst + 1 .. ecvCRILast return CRIs 1 to 16.
ecvCRITriKXFirst..ecvCRITriKXLast	Returns the Tristimulus X of the spectrum without a test color (ecvCRITriKXFirst) or with test color 1..16 (ecvCRITriKXFirst + 1 .. ecvCRITriKXLast)
ecvCRITriKYFirst..ecvCRITriKYLast	Returns the Tristimulus Y of the spectrum without a test color (ecvCRITriKYFirst) or with test color 1..16 (ecvCRITriKYFirst + 1 .. ecvCRITriKYLast)
ecvCRITriKZFirst..ecvCRITriKZLast	Returns the Tristimulus Z of the spectrum without a test color (ecvCRITriKZFirst) or with test color 1..16 (ecvCRITriKZFirst + 1 .. ecvCRITriKZLast)
ecvCRITriRXordUFirst..ecvCRITriRXordULast	Returns the Tristimulus X of the calculated reference spectrum without a test color (ecvCRITriRXordUFirst) or with test color 1..16 (ecvCRITriRXordUFirst + 1 .. ecvCRITriRXordUFirstLast)
ecvCRITriRYordVFirst..ecvCRITriRYordVLast	Returns the Tristimulus Y of the calculated reference spectrum without a test color (ecvCRITriRYordVFirst) or with test color 1..16 (ecvCRITriRYordVFirst + 1 .. ecvCRITriRYordVLast)
ecvCRITriRZordWFirst..ecvCRITriRZordWLast	Returns the Tristimulus Z of the calculated reference spectrum without a test color (ecvCRITriRZordWFirst) or with test color 1..16 (ecvCRITriRZordWFirst + 1 .. ecvCRITriRZordWLast)

The following table lists their meaning when mpidCRIMode is criCIE13_3_95, regardless of mpidColormetricType.

AWhat range	Description
ecvCRIFirst..ecvCRILast	Returns the CRI value, identical to casGetCRI, i.e. AWhat = ecvCRIFirst returns the common CRI, ecvCRIFirst + 1 .. ecvCRILast return CRIs 1 to 16.
ecvCRITriKXFirst..ecvCRITriKXLast	Returns the Tristimulus X of the spectrum without a test color (ecvCRITriKXFirst) or with test color 1..16 (ecvCRITriKXFirst + 1 .. ecvCRITriKXLast)
ecvCRITriKYFirst..ecvCRITriKYLast	Returns the Tristimulus Y of the spectrum without a test color (ecvCRITriKYFirst) or with test color 1..16 (ecvCRITriKYFirst + 1 .. ecvCRITriKYLast)
ecvCRITriKZFirst..ecvCRITriKZLast	Returns the Tristimulus Z of the spectrum without a test color (ecvCRITriKZFirst) or with test color 1..16 (ecvCRITriKZFirst + 1 .. ecvCRITriKZLast)
ecvCRITriRXordUFirst + 1..ecvCRITriRXordULast	Returns dU for CRI 1..16 (ecvCRITriRXordUFirst + 1 .. ecvCRITriRXordUFirstLast)
ecvCRITriRYordVFirst + 1..ecvCRITriRYordVLast	Returns dV for CRI 1..16 (ecvCRITriRYordVFirst + 1 .. ecvCRITriRYordVLast)
ecvCRITriRZordWFirst + 1..ecvCRITriRZordWLast	Returns dW for CRI 1..16 (ecvCRITriRZordWFirst + 1 .. ecvCRITriRZordWLast)

8.1.2.37 casGetExternalADCValue()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetExternalADCValue (
            int ADevice,
            int AIndex )  [static]
```

Obsolete. Use mpidCurrentCCDTemperature. Used to retrieve the CCD temperature.

Parameters

ADevice	The device / CASID
AIndex	The ADC channel to retrieve

Returns

The return value is the CCD temperature

This method should no longer be used. It might not work with some CAS models! Refer to chapter CCD Temperature Monitoring for a complete overview of the topic.

8.1.2.38 casGetFIFOData()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetFIFOData (
            int ADevice )  [static]
```

This method reads the acquired spectrum from the FIFO and stores it internally

Parameters

ADevice	The device / CASID

Returns

Negative return values indicate an error which can be translated into an error message using casGetError↩ Message. All other return values indicate that the spectrum was read successfully

The spectrum is sensitivity corrected and subsequent averages are performed if mpidAverages is $>$ 1.

Calling casGetFIFOData should only be done after a measurement was started with casStart.

To access the acquired spectrum use casGetData

8.1.2.39 casGetFilterName()

```
static IntPtr InstrumentSystems.CAS4.CAS4DLL.casGetFilterName (
            int ADevice,
            int AFilter,
            StringBuilder Dest,
            int AMaxLen )  [static]
```

Translates a filter index into a user-readable density filter name.

Parameters

ADevice	The device / CASID
AFilter	The 0-based filter index ranging from 0 to 7
Dest	Destination buffer for the filter name
AMaxLen	The maximum number of characters Dest can hold

Returns

The return value is identical to Dest or nil if there was an error

The density filter names are defined in the configuration file of the device, so the device must have been initialized (see casInitialize). A default name "Filter x" will be returned when the configuration doesn't define a name for a given filter.

8.1.2.40 casGetMeasurementParameter()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetMeasurementParameter (
            int ADevice,
            int AWhat )  [static]
```

Returns a numeric measurement parameter for a given spectrometer

Parameters

ADevice	The device / CASID
AWhat	Integer constant, defining which parameter to return. One of the mpid<XXX> constants starting at mpidIntegrationTime

Returns

The return value is the measurement parameter itself and value depends on AWhat. Call casGetError immediately afterwards to do proper See Error Handling

After performing a measurement, it is often recommended to query the various measurement parameter to build a list of measurement conditions for documentation.

8.1.2.41 casGetOptions()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetOptions (
            int ADevice )  [static]
```

Returns the features and options the device currently supports or performs.

Parameters

ADevice	The device / CASID

Returns

An integer that has the bits starting with coShutter set - each bit standing for an option or feature.

For historic reasons the options is a mixture of capabilities - which are defined by the device and cannot be changed - and actual options that can be enabled or disabled.

Use the method casSetOptionsOnOff to turn specific options on or off.

The following example checks if the device supports measuring the CCD temperature:

[Delphi]

```
if (casGetOptions(CASID) and coGetTemperature) <> 0 then
  ShowMessage('Device supports measuring CCD temperature') else
  ShowMessage('Device does NOT support measuring CCD temperature');
```

8.1.2.42 casGetPeak()

```
static void InstrumentSystems.CAS4.CAS4DLL.casGetPeak (
              int ADevice,
              out double x,
              out double y )  [static]
```

Retrieves the previously calculated peak wavelength and intensity.

Parameters

ADevice	The device / CASID
x	Receives the wavelength in nm of the interpolated peak
y	Receives the intensity in calibration units of the interpolated peak

Before calling casGetPeak, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

Use casGetError to check for errors.

8.1.2.43 casGetPhotInt()

```
static void InstrumentSystems.CAS4.CAS4DLL.casGetPhotInt (
              int ADevice,
              out double APhotInt,
              StringBuilder AUnit,
              int AUnitMaxLen )  [static]
```

Retrieves the previously calculated photometric integral and it's data unit.

Parameters

ADevice	The device / CASID
APhotInt	Double that will contain the photometric integral after a successful method call.
AUnit	Destination buffer for string representing the photometric data unit
AUnitMaxLen	The maximum number of characters AUnit can hold

Before calling casGetPhotInt, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

The value and data unit will always be returned using the basic unit, e.g. APhotInt = 1.23E-5 and AUnit = "lx" and not in "mlx" or "klx".

The integral is calculated from 380nm to 780nm for cmtDefaultColormetric or from 360nm to 830nm for cmtSW↩PColormetric unless further constrained using mpidColormetricStart and mpidColormetricStop. For integration, the spectrum is multiplied with V(Lambda) and the photometric equivalent Km. It is independent of mpidObserver.

Use casGetError to check for errors.

casGetRadInt retrieves the radiometric integral.

8.1.2.44 casGetRadInt()

```
static void InstrumentSystems.CAS4.CAS4DLL.casGetRadInt (
            int ADevice,
            out double ARadInt,
            StringBuilder AUnit,
            int AUnitMaxLen )  [static]
```

Retrieves the previously calculated radiometric integral and it's data unit.

Parameters

ADevice	The device / CASID
ARadInt	Double that will contain the radiometric integral after a successful method call.
AUnit	Destination buffer for string representing the radiometric data unit
AUnitMaxLen	The maximum number of characters AUnit can hold

Before calling casGetRadInt, a call to casColorMetric is necessary, to calculate all results from the current spectrum.

The value and data unit will always be returned using the basic unit, e.g. ARadInt = 1.23E-5 and AUnit = "W" and not in "mW" or "kW".

The integral is calculated from 380nm to 780nm unless further constrained using mpidColormetricStart and mpid↩ColormetricStop.

Use casGetError to check for errors.

casGetPhotInt retrieves the photometric integral.

8.1.2.45 casGetSerialNumberEx()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetSerialNumberEx (
            int ADevice,
            int AWhat,
            StringBuilder ADest,
            int AMaxLen )  [static]
```

Retrieves serial numbers of the specified device and/or additional information.

Parameters

ADevice	The device / CASID
AWhat	constant starting with "casSerial" identifying which serial number to retrieve
ADest	StringBuilder/PAnsiChar buffer to retrieve the string
AMaxLen	number of characters ADest can hold - including a terminating zero

Returns

0 if successful or an error code

As opposed to dpidSerialNo, this method can retrieve the serial number of the device and additional equipment separately. See chapter Using Serial Numbers for an overview. Possible values for AWhat start with casSerial↩Complete

8.1.2.46 casGetShutter()

```
static int InstrumentSystems.CAS4.CAS4DLL.casGetShutter (
            int ADevice )  [static]
```

Returns the current shutter position of the given device

Parameters

ADevice	The device / CASID

Returns

The return value is one of the casShutter constants starting with casShutterInvalid or one of the error codes, e.g. errCasDeviceNotFound

Note that the actual shutter position of the device is only checked if the device supports the coGetShutter. Otherwise the most recently set shutter state is returned, which may not reflect the actual shutter position.

8.1.2.47 casGetTriStimulus()

```
static void InstrumentSystems.CAS4.CAS4DLL.casGetTriStimulus (
            int ADevice,
            ref double X,
            ref double Y,
            ref double Z )  [static]
```

Retrieves the previously calculated X, Y and Z tristimulus values.

Parameters

ADevice	The device / CASID
X	Variable to receive the tristimulus X
Y	Variable to receive the tristimulus Y
Z	Variable to receive the tristimulus Z

Before calling casGetTriStimulus, casColorMetric must have been called for an already measured spectrum. Calculation is influenced by mpidObserver. The tristimulus is calculated between 380 and 780 nm, but the range can be further restricted by mpidColormetricStart and mpidColormetricStop.

8.1.2.48 casGetWidth()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetWidth (
            int ADevice )  [static]
```

Retrieves the peak width in nm.

Parameters

ADevice	The device / CASID

Returns

The return value is the peak width in nm. Use casGetError to check for errors.

casGetWidth calculates the width of the peak and returns just the width in nm. It is recommended to use cas←GetWidthEx instead, as it also returns other aspects of the peak width. The level of the peak width is 50% by default so the FWHM is calculated. It can be adjusted with mpidColormetricWidthLevel *before* calling casGetWidth or casGetWidthEx.

As with all colormetric calculations, the range of the spectrum which is used for determining the width is defined by mpidColormetricStart and mpidColormetricStop.

8.1.2.49 casGetWidthEx()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetWidthEx (
            int ADevice,
            int AWhat ) [static]
```

Calculates and retrieves various aspects about the peak width (e.g. full width half maximum aka FWHM aka 50% bandwidth).

Parameters

ADevice	The device / CASID
AWhat	One of the cLambda constants specifying what to retrieve; see table below

Returns

The return value is the detail about the width that was specified by AWhat. Use casGetError to check for errors.

By default the FWHM is calculated, i.e. the peak level used for the calculation is 50%. You can modify this level independently for each parameter set using mpidColormetricWidthLevel before calling casGetWidthEx.

The following table lists possible values for the AWhat parameter and a description which aspect of the peak width is returned. The meaning of the algorithm and calculates columns are explained below.

AWhat	Calculates?∗	Algorithm∗∗	Description
cLambdaWidth	Yes	Inner width	The peak width in nm
cLambdaLow	No	Inner width	The start wavelength of the peak width
cLambdaMiddle	No	Inner width	The center wavelength of the peak width, in the middle between start and stop wavelength
cLambdaHigh	No	Inner width	The stop wavelength of the peak width
cLambdaOuterWidth	Yes	Outer width	Returns the peak width
cLambdaOuterLow	Yes	Outer width	The start wavelength of the peak width
cLambdaOuterMiddle	Yes	Outer width	The center wavelength of the peak width, in the middle between start and stop wavelength
cLambdaOuterHigh	Yes	Outer width	The stop wavelength of the peak width

As with all colormetric calculations, the range of the spectrum which is used for determining the peak width is defined by mpidColormetricStart and mpidColormetricStop.

∗ Some AWhat constants only retrieve a previously calculated value and some perform the actual calculation. For example, before you can retrieve cLambdaLow, you must retrieve cLambdaWidth, as only the latter performs the actual calculation using the Inner width algorithm

∗∗ Normally the peak width is determined by "walking" the spectrum up and down from the intensity maximum to check when the intensity falls below the level defined by mpidColormetricWidthLevel. That is the so called "Inner width" algorithm which should normally be used. The "Outer width" algorithm is mainly supported because very old CAS DLL implementations (version 3 and before) used this. This algorithm actually starts walking from the ends of the colormetric range towards the center and stops when the width level of the intensity maximum is reached.

8.1.2.50 casGetXArray()

```
static double InstrumentSystems.CAS4.CAS4DLL.casGetXArray (
            int ADevice,
            int AIndex )  [static]
```

Returns the wavelength of the spectrum for the given pixel.

Parameters

ADevice	The device / CASID
AIndex	The 0-based index of the pixel in the complete pixel array for which the wavelength should be returned

Returns

The return value is the wavelength for the given pixel in nm. Use casGetError to check for ErrorInvalidParameter and other errors.

AIndex can range from 0 to dpidPixels - 1, but the valid calibrated spectrum ranges from dpidDeadPixels to dpidDeadPixels + dpidVisiblePixels - 1.

Use casGetData to retrieve the corresponding intensity at this wavelength.

Note

Contrary to casGetData, casGetXArray does not require that a spectrum has been measured. A successful initialization with casInitialize is sufficient.

[Visual Basic]

```
DeadPixels = casGetDeviceParameter(CasID, dpidDeadPixels)
VisiblePixels = casGetDeviceParameter(CasID, dpidVisiblePixels)

'get Lambda and Intensity, skip dead pixels!
For i = 0 To VisiblePixels - 1
  Spectrum(0, i) = casGetXArray(CasID, DeadPixels + i)
  Spectrum(1, i) = casGetData(CasID, DeadPixels + i)
Next i
```

8.1.2.51 casInitialize()

```
static int InstrumentSystems.CAS4.CAS4DLL.casInitialize (
            int ADevice,
            int Perform )  [static]
```

Initializes the hardware of the device after loading the configuration and calibration files

Parameters

ADevice	The device / CASID
Perform	One of the following constants: InitOnce, InitForced and InitNoHardware

Returns

Returns 0 if the device was successfully initialized or a negative error constant like errCasDeviceNotFound, errCasNoConfig, ErrorNoCalibration or ErrorAdrControl

.

8.1.2.52 casLoadTestData()

```
static void InstrumentSystems.CAS4.CAS4DLL.casLoadTestData (
            int ADevice,
            string AFileName )  [static]
```

Loads a spectrum from an .ISD file into the CASID

Parameters

ADevice	The device / CASID
AFileName	Full null-terminated path of the .ISD file which should be loaded

Use casLoadTestData to load a spectrum from the ISD-file specified by AFileName. The spectrum is stored internally for the given device almost as if it would have just been measured, so a subsequent call to casColorMetric etc. will perform calculations for this spectrum.

Warning

casLoadTestData will only work for a CASID of a real spectrometer, which is connected to the PC and has been initialized. Otherwise it will return ErrorFeatureNotSupported!

Note

Loaded spectra will not work with something like paRecalcSpectrum since this recalculates the spectrum from the previously acquired raw data - which is not affected by casLoadTestData.
The loaded spectrum will be resampled to the wavelength calibration of the spectrometer, so it is only safe to use casLoadTestData with a CASID that has been initialized with the same calibration files that were used to measure the saved spectrum.
Call casGetError after casLoadTestData for error handling.

There are A and W overloads for ANSI and Unicode versions of the method.

8.1.2.53 casMeasure()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMeasure (
            int ADevice )  [static]
```

Performs a measurement for the given device using the measurement parameter which have been previously set

Parameters

ADevice	The device / CASID

Returns

Negative return values indicate an error which can be translated into an error message using casGetError↩ Message. All other return values indicate that the measurement was successful

The spectrum is stored internally and can be accessed using casGetData.

Before starting a measurement, you should check whether the density filter needs to be updated and whether a dark current measurement is necessary. Calling casPerformActionEx with paPrepareMeasurement is also recommended.

See chapter performing a measurement for all prerequisites and more details. If the AutoRange option is enabled (see casGetOptions):

- the integration time is automatically determined (within the bounds of the mpidAutoRangeMinLevel, mpid↩ AutoRangeMaxLevel and mpidAutoRangeMaxIntTime). Refer to the chapter AutoRange measurements for more details.
- using a dark array may be helpful (see chapter Dark current), because casMeasure may automatically perform dark current measurements and even change the density filter if necessary and enabled (coAutorangeFilter)

8.1.2.54 casMeasureDarkCurrent()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMeasureDarkCurrent (
              int ADevice )   [static]
```

Use casMeasureDarkCurrent to perform a dark current measurement for the given device.

Parameters

ADevice	The device / CASID

Returns

Negative return values indicate an error which can be translated into an error message using casGetError↩ Message. All other return values indicate that the dark current was measured successfully

Note

This method does not close the shutter! Call casSetShutter to close the shutter before calling casMeasure↩ DarkCurrent and again to open it afterwards

Warning

Before measuring dark current, check whether the density filter needs to be moved!

Refer to chapter Dark Current for more details and sample code.

To access the measured dark current use casGetDarkCurrent

8.1.2.55 casMultiTrackCopyData()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackCopyData (
            int ADevice,
            int ATrack )  [static]
```

Releases a previously allocated MultiTrack buffer

Parameters

ADevice	The device / CASID
ATrack	The 0-based track index which should be copied/activated

Returns

The return value is 0 if the method was successful or a negative error code

casMultiTrackCopyData retrieves the spectrum specified by ATrack from the MultiTrack measurement buffer (A↩ Track is 0-based, so it may range from 0 to dpidMultiTrackCount - 1). The MultiTrack measurement may have been performed (paMultiTrackStart) or loaded (casMultiTrackLoadData). The spectrum is stored internally as if it would have just been measured, so a subsequent call to casColorMetric calculates the colormetric results for it etc. For an overview, refer to the chapter MultiTrack measurements.

8.1.2.56 casMultiTrackCount()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackCount (
            int ADevice )  [static]
```

Deprecated method.

Returns

Returns the number of MultiTracks already allocated. A negative value indicates an error

Use dpidMultiTrackCount instead

8.1.2.57 casMultiTrackDone()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackDone (
            int ADevice )  [static]
```

Releases a previously allocated MultiTrack buffer

Parameters

ADevice	The device / CASID

Returns

The return value is 0 if the method was successful or a negative error code

casMultiTrackDone releases the MultiTrack buffer after it has been allocated with casMultTrackInit or casMultiTrackLoadData. Obviously, after that, the raw spectra in the MultiTrack buffer are no longer available. For an overview, refer to the chapter MultiTrack measurements.

8.1.2.58 casMultiTrackInit()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackInit (
            int ADevice,
            int ATracks )  [static]
```

Initializes the MultiTrack buffer

Parameters

ADevice	The device / CASID
ATracks	The number of MultiTracks that should later be measured

Returns

The return value is 0 if the method was successful or a negative error code

casMultiTrackInit prepares a MultiTrack measurements for the given device. ATracks defines the number of tracks (i.e. raw spectra) which will be acquired during the MultiTrack measurement and should not exceed dpidMultiTrackMaxCount. Since the memory used for saving MultiTrack measurements needs to be allocated beforehand, it is not possible to change the number of tracks on the fly. You may use dpidMultiTrackCount to check how many tracks have been allocated.

Note

It is mandatory to call casMultiTrackDone to discard the allocated MultiTrack memory when the MultiTrack data is no longer needed. casMultiTrackInit and casMultiTrackLoadData implicitly call casMultiTrackDone before initializing/loading new MultiTrack data.

8.1.2.59 casMultiTrackLoadData()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackLoadData (
            int ADevice,
            string AFileName )  [static]
```

Loads a MultiTrack buffer from a .SWM file.

Parameters

ADevice	The device / CASID
AFileName	Full null-terminated path of the .ISD file which should be loaded

Returns

The return value is the number of tracks loaded if the method was successful or a negative error code

Loads MultiTrack data for the specified device from the file specified by AFileName and discards any MultiTrack data which may have previously been loaded or acquired (casMultiTrackDone is called implicitly). The file must have been saved with casMultiTrackSaveData. Use dpidMultiTrackCount to check how many tracks have been loaded. If you want to retrieve colormetric results or the spectra, call casMultiTrackCopyData, but make sure the same configuration and calibration files are used for the device!

Note

It is mandatory to call casMultiTrackDone to discard the allocated MultiTrack memory, when the MultiTrack data is no longer needed. For an overview, refer to the chapter MultiTrack measurements. There are A and W overloads for ANSI and Unicode versions of the method.

8.1.2.60 casMultiTrackSaveData()

```
static int InstrumentSystems.CAS4.CAS4DLL.casMultiTrackSaveData (
            int ADevice,
            string AFileName )  [static]
```

Saves a MultiTrack buffer to a .SWM file.

Parameters

ADevice	The device / CASID
AFileName	Full null-terminated path of the .SWM file which should be created

Returns

The return value is 0 if the method was successful or a negative error code

Saves the MultiTrack buffer to AFileName. Depending on the number of tracks, the file can get rather big and saving it may take a while. The resulting .SWM file can be loaded by SpecWin Pro or by calling casMultiTrackLoadData. An existing file at AFileName will be overwritten. For an overview, refer to the chapter MultiTrack measurements. There are A and W overloads for ANSI and Unicode versions of the method.

8.1.2.61 casNmToPixel()

```
static int InstrumentSystems.CAS4.CAS4DLL.casNmToPixel (
            int ADevice,
            double nm )  [static]
```

Converts a wavelength into the corresponding CCD pixel index.

Parameters

ADevice	The device / CASID
nm	The wavelength which should be converted into a pixel

Returns

The return value is the 0-based CCD pixel index or a negative error code.

The conversion is done using the wavelength calibration of the specified device. Therefore the device must have been initialized using casInitialize or the wavelength calibration updated with paUpdateCompleteCalibration. The return value is the first 0-based pixel whose wavelength is greater or equal than the nm parameter. Only visible pixel are taken into account. A negative return value indicates an error. casPixelToNm does the conversion the other way around. The returned pixel index can be used with methods like casGetXArray, casGetData or casGetDarkCurrent.

8.1.2.62 casPerformAction()

```
static int InstrumentSystems.CAS4.CAS4DLL.casPerformAction (
            int ADevice,
            int AID ) [static]
```

Deprecated. Use casPerformActionEx instead!

8.1.2.63 casPerformActionEx()

```
static int InstrumentSystems.CAS4.CAS4DLL.casPerformActionEx (
            int ADevice,
            int AActionID,
            int AParam1,
            int AParam2,
            IntPtr AParam3 ) [static]
```

Generic method which performs one of various actions with the specified device. This method replaces the now deprecated method casPerformAction.

Parameters

ADevice	The device / CASID. For some AActionID constants, a valid CASID is not required, so any value will be accepted.
AActionID	Integer defining the action to perform. One of the constants starting at paPrepareMeasurement
AParam1	First integer parameter for the action to perform. Meaning depends on the action to be performed
AParam2	Second integer parameter for the action to perform. Meaning depends on the action to be performed
AParam3	Third pointer parameter for the action to perform. Meaning depends on the action to be performed

Returns

0 for success or a negative error code.

8.1.2.64 casPixelToNm()

```
static double InstrumentSystems.CAS4.CAS4DLL.casPixelToNm (
            int ADevice,
            int APixel ) [static]
```

Converts a CCD pixel index into the corresponding wavelength.

Parameters

ADevice	The device / CASID
APixel	The 0-based CCD pixel for which to retrieve the corresponding wavelength

Returns

The return value is the wavelength. Use casGetError for error checking.

The conversion is done using the wavelength calibration of the specified device. Therefore the device must have been initialized using casInitialize or the wavelength calibration updated with paUpdateCompleteCalibration. casNmToPixel does the conversion the other way around.

8.1.2.65 casSaveCalibration()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSaveCalibration (
            int ADevice,
            string AFileName )  [static]
```

Saves the calibration of the given device to a file.

Parameters

ADevice	The device / CASID
AFileName	The full path where the calibration file should be stored. Typical file extension is .ISC

This calibration file does not contain all details of the calibration, e.g. the wavelength calibration - which is saved in the configuration file. Use casSetCalibrationFactors with gfcWLCalibrationSave to save the wavelength calibration in the current configuration file.

8.1.2.66 casSaveSpectrum()

```
static int InstrumentSystems.CAS4.CAS4DLL.casSaveSpectrum (
            int ADevice,
            string AFileName )  [static]
```

Saves a previously measured spectrum to an .ISD file.

Parameters

ADevice	The device / CASID
AFileName	Full null-terminated path of the .ISD file which should be created.

Returns

The return value is 0 if the method was successful or a negative error code

The ISD file format is an ANSI/UTF-8 text file which can be loaded by SpecWin Pro and other programs. An existing file at AFileName will be overwritten. There are A and W overloads for ANSI and Unicode versions of the method.

8.1.2.67 casSetCalibrationFactors()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetCalibrationFactors (
            int ADevice,
            int AWhat,
            int AIndex,
            int AExtra,
            double AValue )  [static]
```

Changes various details about the configuration/calibration of the given device effectively overriding information that normally comes from configuration/calibration files.

Parameters

ADevice	The device / CASID
AWhat	What calibration part to modify: one of the gcf<XXX> constants, e.g. gcfDensityFunction.
AIndex	For calibration parts which consist of lists or spectra, this is the index or pixel for which to modify the calibration information. See the documentation for the gcf<XXX> constant used as AWhat.
AExtra	Additional constant to indicate what calibration info to change. Often another gcf<XXX> constant, e.g. gcfDarkArrayDepth.
AValue	New value for the calibration information that is identified by the other parameter.

The following table lists the possible values for AIndex and AExtra for the different AWhat constants.

AWhat	AIndex	AExtra	Description
gcfDensityFunction	0..Pixel-1	Density Filter (0..7)	Spectral calibration of the density filters
gcfSensitivityFunction	0..Pixel-1	Unused	Spectral calibration of the CAS
gcfTransmissionFunction	0..Pixel-1	Unused	Transmission of additional optical hardware
gcfDensityFactor	0..7	Unused	Absolute calibration for the density filters
gcfTOPApertureFactor	0..6	Unused	Absolute calibration factor for TOP, aperture given by Index. Check against 0 to find out, which TOP apertures are calibrated.
gcfTOPDistanceFactor	gcfTDCount	Unused	Number of distance and factor pairs
	0..n	gcfTDExtraDistance	TOP distance in mm
		gcfTDExtraFactor	TOP calibration factor for this distance
gcfWLCalibrationChannel	Channel/Pixel to delete	gcfWLExtraCalibrationDelete	Deletes the calibration point for the channel/pixel value passed in Index. No error if no calibration point at this pixel.
	Unused	gcfWLExtraCalibrationDeleteAll	Deletes all calibration points. Identical to calling casClearCalibration with gcfWLCalibrationChannel

	Use gcfWLCalibrationAlias to define calibration points in one go.		
gcfWLCalibrationAlias	0..Pixels-1	Unused	Add/modify calibration point for pixel/channel given by Index and set it to the wavelength passed in AValue.
gcfWLCalibrationSave	Unused	Unused	Immediately saves the modified wavelength calibration in the current configuration file specified by dpidConfigFileName.
gcfDarkArrayValues	unused	gcfDarkArrayDepth	Depth of the dark array, i.e. the number of dark currents in the array
	0..DarkArrayDepth-1	gcfDarkArrayIntTime	Sets the integration time of the DarkArray at Index to the integration time passed in value. Note: the integration times in the dark array must be in ascending order and start with dpidIntTimeMin!
gcfLinearityFunction	0..n	ADCCounts	Adds/modifies the linearity correction array to the correction factor passed in AValue for the ADC counts passed in AIndex. The array must be sorted in ascending order by ADCCounts.

Warning

After changing the calibration paUpdateSpectralCalibration should be called before the next measurement is performed, so the new calibration factors are used! Some actions (like changing the active parameter set or the density filter) do this implicitly so it might not always be necessary. The wavelength calibration is updated automatically every time it is modified.

Use casSaveCalibration if you want to save the modified calibration as a .ISC file. The wavelength calibration can be saved by calling casSetCalibrationFactors with gcfWLCalibrationSave as mentioned above. All the other information that is stored in the configuration file cannot be saved by the CAS-DLL.

8.1.2.68 casSetDeviceParameter()

```
static int InstrumentSystems.CAS4.CAS4DLL.casSetDeviceParameter (
            int ADevice,
            int AWhat,
            double AValue )  [static]
```

Sets a numerical device parameter

Parameters

ADevice	The device / CASID
AWhat	dpid constant identifying which device parameter to change
AValue	the numerical value the device parameter should be set to

Returns

0 if successful or an error code

8.1.2.69 casSetDeviceParameterString()

```
static int InstrumentSystems.CAS4.CAS4DLL.casSetDeviceParameterString (
            int ADevice,
            int AWhat,
            string AValue )  [static]
```

Sets a string device parameter

Parameters

ADevice	The device / CASID
AWhat	dpid constant identifying which device parameter to change
AValue	the null terminated string value the device parameter should be set to

Returns

0 if successful or an error code

8.1.2.70 casSetDigitalOut()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetDigitalOut (
            int ADevice,
            int APort,
            int OnOff )  [static]
```

Sets the state the specified digital output port.

Parameters

ADevice	The device / CASID
APort	Identifies which DigitalOut port state to change. The possible values differ with the spectrometer type and interface - refer to the hardware manual for more information.
OnOff	The desired state: 0 for low, all other values for high state

See casGetDigitalOut for more information.

Warning

Proper error handling using casGetError is essential! If APort is not supported by the device, ErrorParamOutOfRange will be returned!

8.1.2.71 casSetMeasurementParameter()

```
static int InstrumentSystems.CAS4.CAS4DLL.casSetMeasurementParameter (
            int ADevice,
```

```
int AWhat,
double AValue )  [static]
```

Sets a numeric measurement parameter for a given spectrometer

Parameters

ADevice	The device / CASID
AWhat	Integer constant, defining which parameter to modify. One of the mpid<XXX> constants starting at mpidIntegrationTime
AValue	Double holding the new value the measurement parameter should have

Returns

The return value is 0 if successful. Negative values indicate an error code. See Error Handling

Typical error codes include ErrorParamOutOfRange and ErrorInvalidParameter.

8.1.2.72 casSetOptions()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetOptions (
            int ADevice,
            int AOptions )  [static]
```

This method sets and clears all device options for the device

Parameters

ADevice	The device / CASID
AOptions	Integer which has all bits set for the corresponding options which should be enabled and all bits cleared whose corresponding options should be disabled. Bits start with coShutter

Since some options are defined by the hardware and should not be changed manually, it is recommend to use the method casSetOptionsOnOff instead as it allows setting and clearing only the bits you actually want to modify.

8.1.2.73 casSetOptionsOnOff()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetOptionsOnOff (
            int ADevice,
            int AOptions,
            int AOnOff )  [static]
```

This method can set or clear several options for the device

Parameters

ADevice	The device / CASID
AOptions	Integer which has all bits set for the corresponding options which should modified. Bits start with coShutter
AOnOff	0 if the options should be disabled, all other values will enable them

Some options are defined by the device and should not be changed manually, see casGetOptions.

[Delphi]

```
//turn AutoRange and AutoRange filter on
casSetOptionsOnOff(CASID, coAutorangeMeasurement or coAutorangeFilter, 1);

//turn UseTransmission off
casSetOptionsOnOff(CASID, coUseTransmission, 0);
```

8.1.2.74 casSetShutter()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetShutter (
            int ADevice,
            int OnOff )  [static]
```

Sets the shutter position of the given device.

Parameters

ADevice	The device / CASID
OnOff	Desired shutter state: either casShutterOpen or casShutterClose

If the device does not have a shutter (coShutter option missing), this method will not raise an error!

Warning

Error handling for this method using casGetError is essential!

8.1.2.75 casSetStatusLED()

```
static void InstrumentSystems.CAS4.CAS4DLL.casSetStatusLED (
            int ADevice,
            int AWhat )  [static]
```

Method to control the status LED of the spectrometer

Parameters

ADevice	The device / CASID
AWhat	One of the ext constants starting with extNoError describing the state the status LED should change to

In general, the status LED is controlled by the firmware of the spectrometer and an error of filter or shutter operation is signalled via the status-LED to the operator. The casSetStatusLED command can be used to override this and to generate a user defined error (e.g. if the sensor temperature rises above -8℃). If the LED signals a malfunction of the shutter or filterwheel, a successful operation of the same component brings the LED back to the "normal" state. For example when you manually signal a shutter error using casSetStatusLED, a successful shutter operation causes the LED to stop blinking (the LED is green again). If you operate the filter wheel successfully this has no influence on the status LED. In general, using this method is not recommended nor necessary. Refer to the hardware manual of the spectrometer for more details about the status LED and it's possible states.

The following table list the states of the CAS 140CT

AWhat	LED State	Description
N/A	red	after power-on, not initialized
N/A	orange	spectrometer is busy, e.g. moving filter wheel
extNoError	green	initialized, no error
extFilterBlink	red, blinking	filter error
extShutterBlink	green, blinking	shutter error
extExternalError	orange, blinking	user defined error

Note

Use casGetError for error handling!

8.1.2.76 casStart()

```
static int InstrumentSystems.CAS4.CAS4DLL.casStart (
             int ADevice )  [static]
```

Starts a measurement for the given device and returns immediately

Parameters

ADevice	The device / CASID

Returns

The return value depends on the interface type. Use casGetError to retrieve error information

Unlike casMeasure, this method does not wait until the measurement has been performed and it also doesn't store the measured spectrum internally. Use the method casFIFOHasData to wait until the measurement has finished and casGetFIFOData to store the spectrum internally.

Note

It is not guaranteed that all interface types actually return immediately. Some may return only after the integration time is over.

Before starting a measurement, you should check whether the density filter needs to be updated and whether a dark current measurement is necessary. Calling casPerformActionEx with paPrepareMeasurement is also recommended.

8.1.2.77 casUpdateCalibrations()

```
static void InstrumentSystems.CAS4.CAS4DLL.casUpdateCalibrations (
             int ADevice )  [static]
```

Updates the calibration information for the given device. Deprecated! Use paUpdateSpectralCalibration.

Parameters

ADevice	The device / CASID

This method recalculates the spectral calibration factors after changing them via casSetCalibrationFactors. If you don't edit the spectral calibration manually, there is no need to call this method, since it is called automatically whenever necessary (for example when changing the density filter or the currently active parameter set).

8.1.2.78 cmXYToDominantWavelength()

```
static int InstrumentSystems.CAS4.CAS4DLL.cmXYToDominantWavelength (
            double x,
            double y,
            double IllX,
            double IllY,
            ref double LambdaDom,
            ref double Purity )  [static]
```

Calculates dominant wavelength aka LambdaDom and purity from a given color coordinate and illuminant reference. This method is deprecated because it does not take the new mpidColormetricType into account. Use casCalculateLambdaDom instead.

Parameters

x	The x color coordinate of the spectrum
y	The y color coordinate of the spectrum
IllX	The x color coordinate of the illuminant reference
IllY	The y color coordinate of the illuminant reference
LambdaDom	The variable which will receive the calculated dominant wavelength
Purity	The variable which will receive the calculated purity

Returns

The return value is 0 and should be ignored

IllX and IllY are the color coordinates of the illuminant reference. One of the typical references is Illuminant E where x and y are 0.3333. To use the color coordinates of the previously measured spectrum, retrieve them with casGetColorCoordinates and pass them as x and y. Purity is 0 if x = IllX and y = IllY. Purity will never exceed 1.

8.1.3 Member Data Documentation

8.1.3.1 mpidACQStateLine

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidACQStateLine = 37
```

integer: number of the digital out port which should be used for the toShowACQState option in mpidTriggerOptions). The port specified by this parameter is set to the level given by mpidACQStateLinePolarity after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time. For possible port values refer to casSetDigitalOut and the hardware manual of the spectrometer.

Note

When setting this parameter the level of the given line will be changed immediately if the device has been initialized and the toShowACQState trigger option is enabled.

Warning

Not all device types support changing this parameter which also causes varying default values depending on the device type. Therefore always verify this parameter, if you're relying on a specific line.

See chapter Triggered Measurements for an overview of all related options and parameters.

8.1.3.2 mpidACQStateLinePolarity

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidACQStateLinePolarity = 38
```

integer: level of the ACQ state line. Only applies if the toShowACQState option is enabled in mpidTriggerOptions. The value is 0 for low level, all other values indicate high level. The port specified by mpidACQStateLine is set to this level after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time.

Note

When setting this parameter the level of the digital port will be changed immediately, if the device has been initialized and the toShowACQState trigger option is enabled.

Warning

Not all device types support changing this parameter, which also causes varying default values depending on the device type. Therefore always verify this parameter, if you're relying on a specific polarity. We recommend using the low level for ACQ. You might also check the tcoBusyStatePolarity capability to see if the spectrometer supports a custom polarity.

See chapter Triggered Measurements for an overview of all related options and parameters.

8.1.3.3 mpidBusyStateLine

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidBusyStateLine = 39
```

integer: number of the digital out port which should be used for the toShowBusyState option in mpidTriggerOptions. The port specified by this parameter is set to the level given by mpidBusyStateLinePolarity after the spectrometer has been started and is ready to accept a trigger. The level is restored after the trigger has been received. For possible port values refer to casSetDigitalOut and the hardware manual of the spectrometer.

Note

When setting this parameter the level of the given line will be changed immediately if the device has been initialized and the toShowBusyState trigger option is enabled.

Warning

Not all device types support changing this parameter, which also causes varying default values depending on the device type. Therefore always verify this parameter, if you're relying on a specific line. You might also check the tcoBusyStatePolarity capability to see if the spectrometer supports a custom polarity.

See chapter Triggered Measurements for an overview of all related options and parameters.

8.1.3.4 mpidBusyStateLinePolarity

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidBusyStateLinePolarity = 40
```

integer: level of the Busy state line. Only applies if the toShowBusyState option is enabled in mpidTriggerOptions. The value is 0 for low level, all other values indicate high level. The port specified by mpidACQStateLine is set to this level after the spectrometer has been started and is ready to accept a trigger. The level is restored after the acquisition, i.e. after integration and read-out time.

Note

When setting this parameter the level of the digital port will be changed immediately, if the device has been initialized and the toShowACQState trigger option is enabled.

Warning

Not all device types support changing this parameter, which also causes varying default values depending on the device type. Therefore always verify this parameter, if you're relying on a specific polarity. We recommend using the low level for ACQ.

See chapter Triggered Measurements for an overview of all related options and parameters.

8.1.3.5 mpidCurrentCCDTemperature

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidCurrentCCDTemperature = 46
```

reading this mpid performs a temperature measurement and returns the CCD temperature in degrees Celsius. If the device does not support temperature measurements the return value will be NAN. To verify that the device supports temperature measurements, check the coGetTemperature flag in casGetOptions.

Note

This method call might cause an error if the CCD temperature is outside of the allowed range (ErrorCCDTemperatureFail). Therefore it is especially important to check casGetError after retrieving the current CCD temperature.

8.1.3.6 mpidFlashDelayTime

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidFlashDelayTime = 29
```

float: the delay time of the flash output signal in milliseconds. Applies only when flash is activated, i.e. mpidFlashType <> ftNone. If the delay time is negative, the flash signal and the delay happen before starting the measurement, otherwise the delay and the flash signal occur after the measurement started.

Note

Not all spectrometers support flash delay! Check the relevant bits in dpidTriggerCapabilities to ensure that the desired behaviour is actually supported by the spectrometer.

8.1.3.7 mpidForceFilter

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidForceFilter = 34
```

integer: flag which controls whether the filter wheel is moved for every measurement, even if it had been set to the same position previously. Internally the flag only affects dpidNeedDensityFilterChange, it will always return True if the ForceFilter flag is set.

Note

Some device types might enable ForceFilter during casInitialize. This is typically done for devices which can read the current filter state from the device (coGetFilter in casGetOptions). If only one application and device handle access the spectrometer, setting mpidForceFilter back to 0 can help minimize the measurement cycle time, but note that some devices might even enforce that ForceFilter is enabled. This is typically done for devices where monitoring the filter is a very fast operation ($<$1ms). Refer to the chapter Density Filter for an overview of all methods and parameter related to the density filter.

8.1.3.8 mpidLastCCDTemperature

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidLastCCDTemperature = 47
```

returns the previously measured CCD temperature in degrees Celsius. If the last temperature measurement is too old, a new temperature measurement will be performed. The interval which triggers a new temperature measurement is device dependent. If the device does not support temperature measurements the return value will be NAN. To verify that the device supports temperature measurements, check the coGetTemperature flag in casGetOptions.

Note

If a new temperature measurement had to be performed, this method call might cause an error if the CCD temperature is outside of the allowed range (ErrorCCDTemperatureFail), so it is especially important to check casGetError.

8.1.3.9 mpidRelSaturation

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidRelSaturation = 25
```

returns the relative saturation (between 0 and 100%) of the previous successful measurement for the active parameter set. mpidRelSaturation can be checked against the dpidRelSaturationMin and dpidRelSaturationMax range, to ensure that the measurement has a sufficient signal level. However, to check for oversaturated measurements use mpidMaxADCValue and check it against dpidADCRange. Because with averaging, mpidRelSaturation might be $<$ 100% even though there were saturated spectra!

Note

Contrary to mpidMaxADCValue, the relative saturation is also affected by the Dark Current. It returns the percentage of the signal of the actual ADC range available for light, i.e. dpidADCRange minus the dark current at the mpidMaxADCPixel

8.1.3.10 mpidTOPAperture

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidTOPAperture = 30
```

integer: the TOP aperture which should be taken into account when the calibration is applied to the spectrum. The TOP aperture parameter is a 0-based index ranging from 0 to 6 corresponding to the TOP apertures 1 to 7. This parameter doesn't actually adjust the aperture of the TOP, but controls which aperture calibration factor will be applied.

Note

If no TOP is used, leave mpidTOPAperture at 0. To find out whether a specific TOP aperture is calibrated, use casGetCalibrationFactors with gcfTOPApertureFactor to check whether the factor for the aperture is not 0.

8.1.3.11 mpidTOPDistance

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidTOPDistance = 31
```

the distance in mm from the DUT to the reference plane of the TOP. This parameter doesn't actually adjust the distance of the TOP, but controls which calibration factor will be applied.

Note

To check which distance range is covered by the calibration, use casGetCalibrationFactors with gcfTOPDistanceFactor. If the calibration does not contain TOP distance factors, this parameter has no effect.

8.1.3.12 mpidTOPFieldOfView

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidTOPFieldOfView = 44
```

returns the field of view for the current mpidTOPAperture and mpidTOPDistance.

Note

Not all configurations support the calculation of field of view - in these cases mpidTOPFieldOfView will always return 0. It is recommended to only use this measurement condition when the coTOPHasFieldOfViewConfig option is returned by casGetOptions.

8.1.3.13 mpidTriggerOptions

```
const int InstrumentSystems.CAS4.CAS4DLL.mpidTriggerOptions = 33
```

integer: bit-set describing current trigger options. The set consists of the various to<XXX> bits, starting with toAcceptOnlyWhenReady.

Note

Some options apply to hardware and software triggers! See chapter Triggered Measurements for an overview. Additionally a device type might enforce specific trigger options, for example toShowACQState is always enabled for some CAS USB devices with hardware triggered measurements (mpidTriggerSource = trgFlipFlop).

The documentation for this class was generated from the following file:

- InstrumentSystems.CAS4.CAS4DLL.cs

Index