API Reference

Overview
This guide documents how to use the Spectronic Devices CameraServer software to access camera devices connected to the computer. Applications have a choice of two ways to use the software:

- Directly calling functions on the CameraServer DLL
- Using the COM interface to the CameraServer DLL

The COM interface is the easiest interface to use when developing with late binding languages such as Visual Basic. In addition, by using COM, the Camera need not be installed on the same computer as the application software.

Accessing cameras through the DLL directly is primarily intended for developers using languages such as C/C++. This mechanism is less cumbersome for C applications than using COM but does mean that the application must be installed on the same computer as the camera.

How to use this Guide
Each section in this guide documents a single function. Since the CameraServer software can be used directly as a DLL linked to your program or accessed via a COM object, usages for both types are included.

The term Camera is used throughout this document when referring to the device being used. This device can be a Line Scan Camera or a Spectrometer.

Programming Considerations when using the COM version
VB users simply need to create an instance of the CameraServer object and can then start calling methods on it directly:

```
Dim cs as CameraServer
Set cs = CreateObject("SpectronicDevices.CameraServer")
```

Developers using C/C++ will need to:

- Include the header file csmidl.h
- #define _WIN32_DCOM
- Link against the libraries ole32.lib, oleaut32.lib and csmidl_iid.obj

Regardless of programming language used, all COM APIs will return the standard COM status code S_OK (0) if the function call succeeds.

Programming Considerations when Linking directly to the CS DLL
This is method is primarily intended for use by developers using C/C++ who do not wish to access Cameras not installed on the same computer as the application. Developers will need to:

- Include the header file CameraFunctions.h

- Link against the library CameraFunctions.lib

Most functions return a Boolean value to indicate whether the operation was completed successfully or not. Unless explicitly stated otherwise assumes these functions will return TRUE if the operation succeeds.

API Reference
Regardless of which programming model is chosen, the API breaks down into two distinct groups of functions:

- Standard API. Provides a simple mechanism for taking readings from a camera. This API is sufficient for most users of the software.
- Advanced API. Affords extra control over the camera. For use by developers who are intimately familiar with how the Spectronic Devices Cameras function.

Standard API

DoMeasurement	Instructs the camera to make a measurement and returns the data it collected.
GetConnectedDevices	Returns a list of cameras connected to the computer.
GetCameraAttributes	Returns various camera parameters.
GetDeviceObject	Returns a reference to a camera.
PrepareMeasurement	Configures the camera to take measurements.

Advanced API

EnableTimerTrigger	Sets the trigger source for the camera.
GetCameraData	Retrieves data from the camera for the last measurement.
GetVersionInfo	Returns the software version.
SetIntegrationPeriod	Sets the camera exposure.
SetReadouts	Sets the number of samples to make when making a measurement.
TriggerDataCapture	Triggers the camera.

Example Code
Included with the software are several examples of how to connect up to a camera and collect data from it. The examples include:

- CCOMClient.cpp A C++ example that demonstrates how to connect to and collect data from a Camera using COM.
- CDLLClient A C++ example that demonstrates how to connect to and collect data from a Camera by directly connecting to the Camera Server DLL.
- CSClient.xls An Excel Spreadsheet based example that uses VBA to connect to and collect data from the Camera.
- USBCamera is a program written using Delphi. The source code is available in the examples folder and an executable is in the “bin” folder. This examples shows how to initiate and use the cameraServer to communicate with a camera or spectrometer. Although it is called USBCamera it will communicate with all devices whether they have a USB, RS232 or Parallel Port EPP interface.

All the above examples use the standard version of the API to control the Camera and it is envisaged that this API will more than meet most users requirements.

DoMeasurement

This function is a convenience function to perform all the steps necessary to make a measurement after the camera has been first configured.

Direct DLL Usage

```
BOOL DoMeasurement(      PortDriverBase*   pCamera
                         BOOL              bAverage,
                         long*             piSamples,
                         long*             piPixels,
                         const long*       ppiData[])
```

piSamples and *piPixels* contain the number of samples and pixels per sample.

ppiData contains all the pixel data for each sample taken. For the organisation of the data array see the documentation for *GetCameraData.*

COM Usage

```
HRESULT ICSInterface::DoMeasurement(BSTR        bszCamera,
                                    int         bAverage,
                                    VARIANT*    vData)
```

vData contains a two dimensional safe array of long integers. For the organisation of the data array see the documentation for *GetCameraData.*

EnableTimerTrigger
Sets the trigger source for the camera. Valid trigger sources are:

1. Internal Trigger (uses the internal timer, period is set using SetIntegrationPeriod)
2. Software Trigger
3. External Trigger. (if this option is available with the camera).

Direct DLL Usage
```
BOOL EnableTimerTrigger(PortDriverBase*   pCamera,
                        Long              iTriggerType)
```

COM Usage
```
HRESULT ICSInterface::EnableTimerTrigger(    BSTR bszCamera,
                                             long iTriggerType)
```

GetCameraData
Fetches the data collected by the camera for the last measurement it took.

If the camera has been configured to take multiple samples, passing a value of TRUE for *bAverage* will result in a single sample being returned that is an average of all the samples taken.

Direct DLL Usage

```
BOOL GetCameraData(     PortDriverBase*   pCamera
                        BOOL              bAverage,
                        long*             piSamples,
                        long*             piPixels,
                        const long*       ppiData[])
```

piSamples and *piPixels* contain the number of samples and pixels per sample.

ppiData contains all the pixel data for each sample taken. Individual pixel elements for a given sample can be directly accessed within the array using the formula:

*pixel = ppiData[(sample number * pixels per sample) + required pixel]*

For example, if the camera supports 2048 pixels and has been requested to take 3 samples per measurement and average is set to false then:

ppiData[0] = pixel 1 of sample 1
ppiData[2047] = pixel 2048 of sample 1
ppiData[2048] = pixel 1 of sample 2
ppidata[6143] = pixel 2048 of sample 3

COM Usage

```
HRESULT ICSInterface:: GetCameraData(     BSTR        bszCamera,
                                          BOOL        bAverage,
                                          VARIANT*    vData)
```

vData contains a two dimensional safe array of long integers. The array is organised as n samples each containing m pixels.

GetConnectedDevices
Instruct the CameraServer software to scan the computer looking for Cameras

Direct DLL Usage
```
const char** GetConnectedDevices(long *piDevices)
```

Returns an array of strings listing the cameras that have been found on the computer. The number of cameras in the list is returned by *piDevices*.

COM Usage
```
HRESULT ICSInterface::GetConnectedDevices(VARIANT* vDevices)
```

Returns a Safe Array containing the list of cameras connected to the computer. The caller is responsible for destroying the list.

GetCameraAttributes

Obtains various camera parameters:

- Pixels – Number of pixels supported by the camera
- Resolution – Pixel word size in bits
- Pixel Clock period – microseconds.

Direct DLL Usage

```
BOOL GetCameraAttributes( PortDriverBase*  pCamera,
                          long*            piPixels,
                          long*            piResolution,
                          long*            piClkSpeed)
```

COM Usage

```
HRESULT ICSInterface::GetCameraAttributes(BSTR  bszCamera,
                                          long* piResolution,
                                          long* piPixels
                                          long* piClkSpeed)
```

GetDeviceObject

Returns a reference to the named camera object. All operations on the camera are performed in the context of a camera object.

Direct DLL Usage

```
PortDriverBase* GetDeviceObject(char* szCamera)


PortDriverBase* pCamera =  GetDeviceObject("COM1");
PrepareMeasurement(pCamera,1,15);
```

COM Usage

Not available

GetVersionInfo
Query the camera software for version information.

Direct DLL Usage
```
unsigned long GetVersionInfo()
```

Returns the DLL version number.

COM Usage
```
HRESULT ICSInterface::GetVersionInfo(       long* piCOMSvrVer,
                                            long* piDLLVer)
```

Obtains both the COM server version and the version of the DLL that controls the camera.

PrepareMeasurement
This function performs all the tasks necessary to prepare the camera for taking measurements. This function is a convenience function that encompasses various other low level APIs which include:

- Setting the integration time
- Setting the number of samples to take every time a measurement is made
- Setting the trigger source for the camera

This function (or the individual APIs it encompasses) need to be called once before the first measurement is taken. After that, this function need not be called again unless you wish to make changes to the camera configuration.

Note: See *EnableTimerTrigger* For valid values for *iTriggerType*.

Direct DLL Usage
```
BOOL  PrepareMeasurement(PortDriverBase*  pPort,
                         long             iNumSamples,
                         long             iIntegrationPeriod,
                         long             iTriggerType)
```

COM Usage
```
HRESULT ICSInterface::PrepareMeasurement(
                         BSTR  bszCamera,
                         long  iNumSamples,
                         long  iIntegrationPeriod,
                         long  iTriggerType)
```

SetIntegrationPeriod

Sets the integration period (exposure time in ms.) for the camera.

Direct DLL Usage

```
BOOL  SetIntegrationPeriod(   PortDriverBase*   pCamera,
                              long              iIntegrationPeriod)
```

COM Usage

```
HRESULT ICSInterface:: SetIntegrationPeriod( BSTR bszCamera,
                                             long iIntegrationPeriod)
```

SetReadouts

Sets the number of samples the camera should make when performing a measurement. By default, the camera will take one sample.

Direct DLL Usage

```
BOOL  SetReadouts(PortDriverBase* pCamera,long iNumSamples)
```

COM Usage

```
HRESULT ICSInterface::SetReadouts(  BSTR  bszCamera,
                                    Long  iNumSamples)
```

TriggerDataCapture

Trigger the camera to start taking measurements. If the internal trigger mode is selected (see EnableTimerTrigger). the measurement is synchronised to this timer. If external trigger is used the user must supply a trigger signal otherwise the camera will not be triggered and this function will only return after the timeout period.

Direct DLL Usage

```
BOOL TriggerDataCapture(PortDriverBase* pCamera)
```

COM Usage

```
HRESULT ICSInterface::TriggerDataCapture(BSTR bszDevice)
```

COM Ports and Baud Rates

In order for the CameraServer serial port driver to be able to detect a camera on a COM port, it must know the baud rate at which the camera is running. Auto detection of the baud rate is not used. The serial driver attempts to read this information from the registry – looking under:

HKEY_LOCAL_MACHINE\Software\Spectronic Devices\CameraServer\<Name>

Where <Name> is the name of the COM port (e.g. COM1). There is a value under this key called ‘Baud Rate’ which specifies the baud rate which the camera is operating at (e.g. 19200).

If the baud rate of a serial camera is changed using the ‘A’ command (see SpectronicDevicesRS232Interface.doc), then this registry entry needs to be changed to reflect the new baud rate. If this is not done, then the CameraServer will fail to detect the camera. Also returning the baud rate to the value used the last time the CameraServer was used will also ensure proper communication with the camera.

The Camera Server code sets the baud rate of the COM port to be the one specified in the registry. The COM port setting is not maintained when a PC is rebooted. Therefore, the baud rate is set to the value in the Registry by the Camera Server each time the COM port is opened to ensure communication with the camera.