Fianium

SuperChrome SDK

1 Introduction

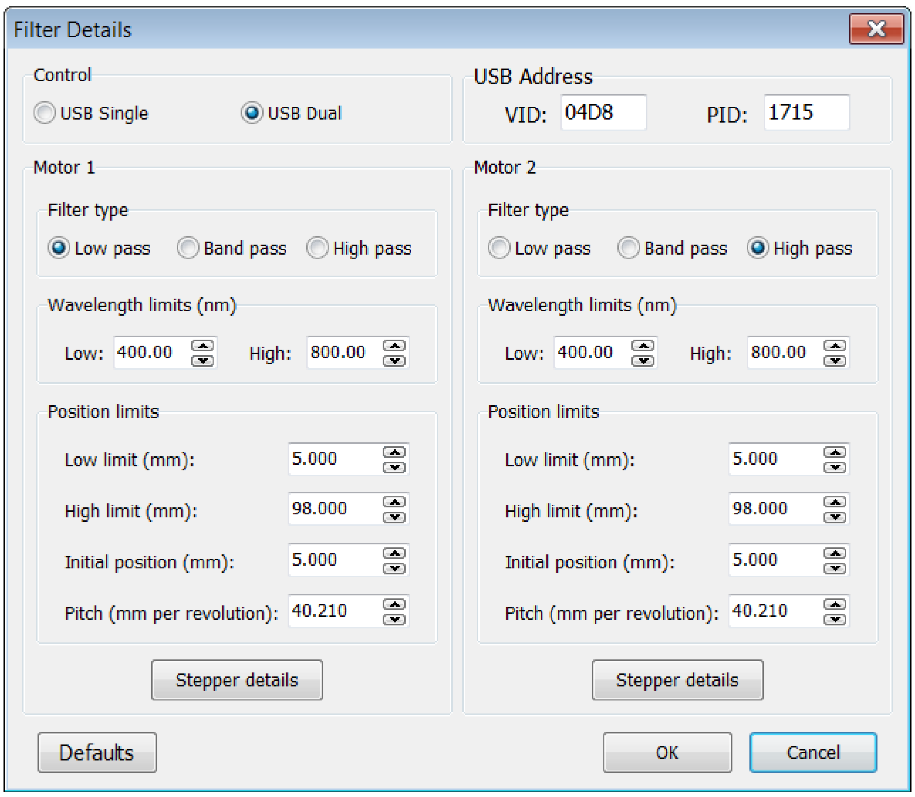
The Fianium SuperChrome wavelength selection unit is a device used to restrict the throughput from a Fianium supercontinuum laser to a specified wavelength and wavelength range (bandwidth). It does this by using a set of linear interference filters (also known as wedge filters) to set a passband of wavelengths. The motors used to drive these filters are computer-controlled and this control is the primary purpose of the SuperChrome SDK.

This manual describes a set of exportable functions from a dynamic link library (DLL) whose purpose is to permit control of a SuperChrome wavelength selection unit from an external programme. A description of the form used to configure the SuperChrome components is provided, as is a description of the form used to calibrate the unit for wavelength. Both of these forms can be called from the DLL. Note, however that a SuperChrome unit is both factory-configured and factory-calibrated prior to being shipped.

2 Description of Motor Characteristics

2.1 Configuration

When the ConfigureModule function is called the form shown in Figure 1 is opened. The purpose of this form is to permit the user to configure the filter device for the particular combination of filters contained in it.



Filter Details

Control: ( ) USB Single [✓ USB Dual]

USB Address: VID: 04D8 PID: 1715

Motor 1

Filter type: [✓ Low pass] ( ) Band pass ( ) High pass

Wavelength limits (nm): Low: 400.00 High: 800.00

Position limits:
Low limit (mm): 5.000
High limit (mm): 98.000
Initial position (mm): 5.000
Pitch (mm per revolution): 40.210

Motor 2

Filter type: ( ) Low pass ( ) Band pass [✓ High pass]

Wavelength limits (nm): Low: 400.00 High: 800.00

Position limits:
Low limit (mm): 5.000
High limit (mm): 98.000
Initial position (mm): 5.000
Pitch (mm per revolution): 40.210

Figure 1: Filter details configuration dialog box.

The sections in the form are:

Control: This group box is used to select the control source for the filter unit. The options are:

USB Single is used for the selection of a single filter under the control of a USB link to a computer. With this option selected the items in the Motor 1 grouping become available.

USB Dual is used for the selection of a dual filter under the control of a USB link to a computer. With this option selected both the Motor 1 and Motor 2 groupings will both become available.

USB Address: This grouping permits entry of the USB identifiers for USB Single and USB Dual filters. For these devices, the identities are normally:

VID: 04D8 (hexadecimal) = 1240 (decimal).

PID: 1715 (hexadecimal) = 5909 (decimal).

Filter type: This grouping permits entry of a type of filter driven by the motor. The choices are:

Low pass: This selection is used to indicate that the filter is a low-pass wavelength-dependent one.

Band pass: This selection is used to indicate that the filter is a band-pass wavelength-dependent one.

High pass: This selection is used to indicate that the filter is a high-pass wavelength-dependent one.

Wavelength limits: This grouping is visible for low pass, band pass, high pass and generic filter types. It is used to enter the upper and lower wavelength limits for the filter.

Position limits: The settings in this grouping are intended for entry of the physical limits of the motor movement and for initial positioning and gearing. The units are therefore those of length.

Low limit: This value is the physical low limit position that the motor can have. During initialisation the motor will move to its home position, detected by an optical or magnetic sensor, and then move out to this position. Thus the value should be set high enough to avoid influencing the home position sensor. This value is also used to move the filter to its 'out of beam path' position and again it should be verified that the value entered here achieves this purpose.

High limit: This value represents the highest position that the motor can take and it should be less than the physical end of travel monitored by an end-stop or position sensor.

Initial position: This value is the one that the motor should move to after it has been initialised.

Pitch: This value is the pitch of the worm gear or belt used to drive the motor.

The buttons in the form are:

Defaults: This button is used to enter a set of defaults for the selected control type. Values are also entered for motor(s).

OK: This button is used to save the entered values and then to close the form.

Cancel: This button is used to close the form without saving the values.

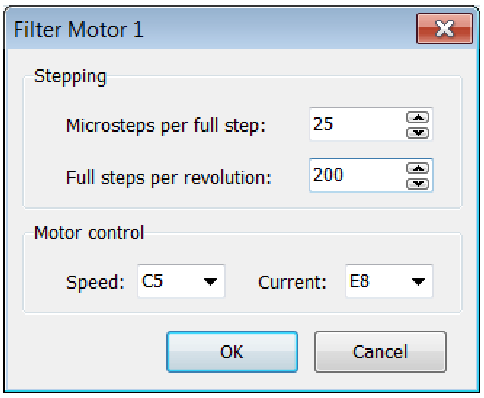
Stepper details: This button opens a dialog box (Figure 2) to permit entry of the stepper motor details. These are normally dependent on the motor's hardware and so are not freely settable. The values are:

Microsteps per full step: This is the number of microsteps in one full step of the motor.

Full steps per revolution: This is the number of full steps in one complete revolution of the stepper motor spindle.

Speed: This value sets the maximum speed that the motor can move and is used for tuning the motor movement.

Current: The value sets the motor drive current and is also used for tuning the motor movement


Filter Motor 1

Stepping

Microsteps per full step: 25

Full steps per revolution: 200

Motor control

Speed: C5

Current: E8

Figure 2: Filter motor configuration dialog box.

2.2 Calibration

Each filter is calibrated using a table of calibration values. The table consists of between two and ten sets of values and linear interpolation (with extrapolation if necessary) is used for wavelengths not in the table.

For each position in the table there are three stored values and these are:

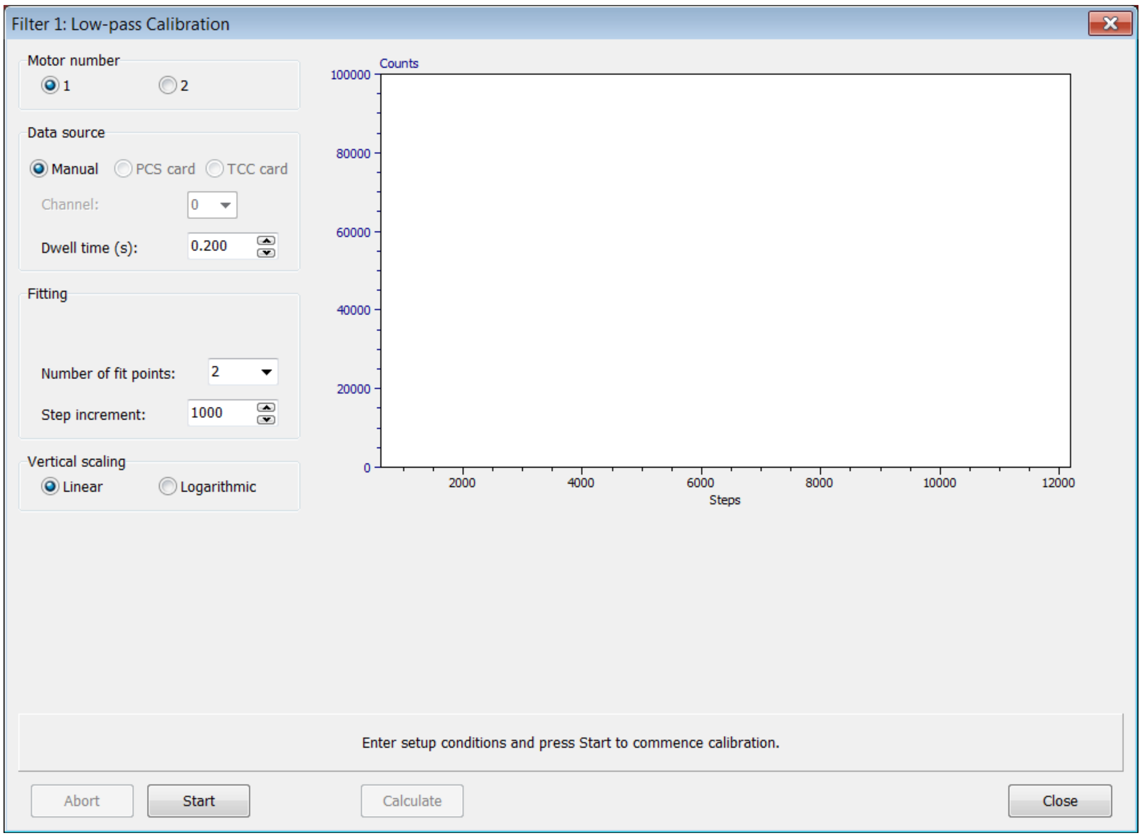
Wavelength: The wavelength where the values were measured.

StepsAtMax: The position, in steps, where the signal was at its maximum. For low and high pass filters this is the maximum value entered by the user during the calibration procedure; for band pass filters this is the mid-point of the lower and upper half-maximum values.

WaveOffset50: For low-pass and high-pass filters this is the wavelength difference between the maximum and the half-maximum points; for band-pass filters this is the wavelength difference between the maximum value and the half-maximum value.

In order to determine these values a wavelength calibration procedure is followed in which the user is asked to enter values from an experimentally-determined response curve, obtained by scanning the filter past a single-wavelength light source (normally a laser) from the lowest possible step position to the highest step position.

The calibration procedure is processed by calling the RunCalibration function and when this function is called, the form shown in Figure 3 is displayed. The purpose of this form is to perform a calibration procedure in order to determine the variable calibration parameters. The calibration procedure needs at least two different wavelengths in order to work. These can be obtained, for example, by using two different diode lasers or by changing the wavelength of a monochromator. A detector connected to an Edinburgh Instruments (EI) measuring card (PCS card or TCC card) is also currently required to perform the calibration interactively.


Filter 1: Low-pass Calibration

Motor number: [✓ 1] 2

Data source: [✓ Manual] PCS card TCC card

Channel: 0

Dwell time (s): 0.200

Fitting

Number of fit points: 2

Step increment: 1000

Vertical scaling: [✓ Linear] Logarithmic

Enter setup conditions and press Start to commence calibration.

Figure 3: Filter calibration dialog box (for a low-pass filter).

Motor number: The motor number whose calibration values are to be edited is selected here.

Data source: The measuring card that the detector is connected to is entered here. This can be either a PCS 900 card or a TCC 900 card. The channel is also entered as is the time (dwell time) that the card should count photons at each step position. . It is possible to enter the values manually using the Manual option.

Fitting: This grouping is used to enter the nature of the fit. The **Number of fit points** to be used in the fitting algorithm is entered here. The **Step increment** box is used to set the number of steps to be incremented for each data point.

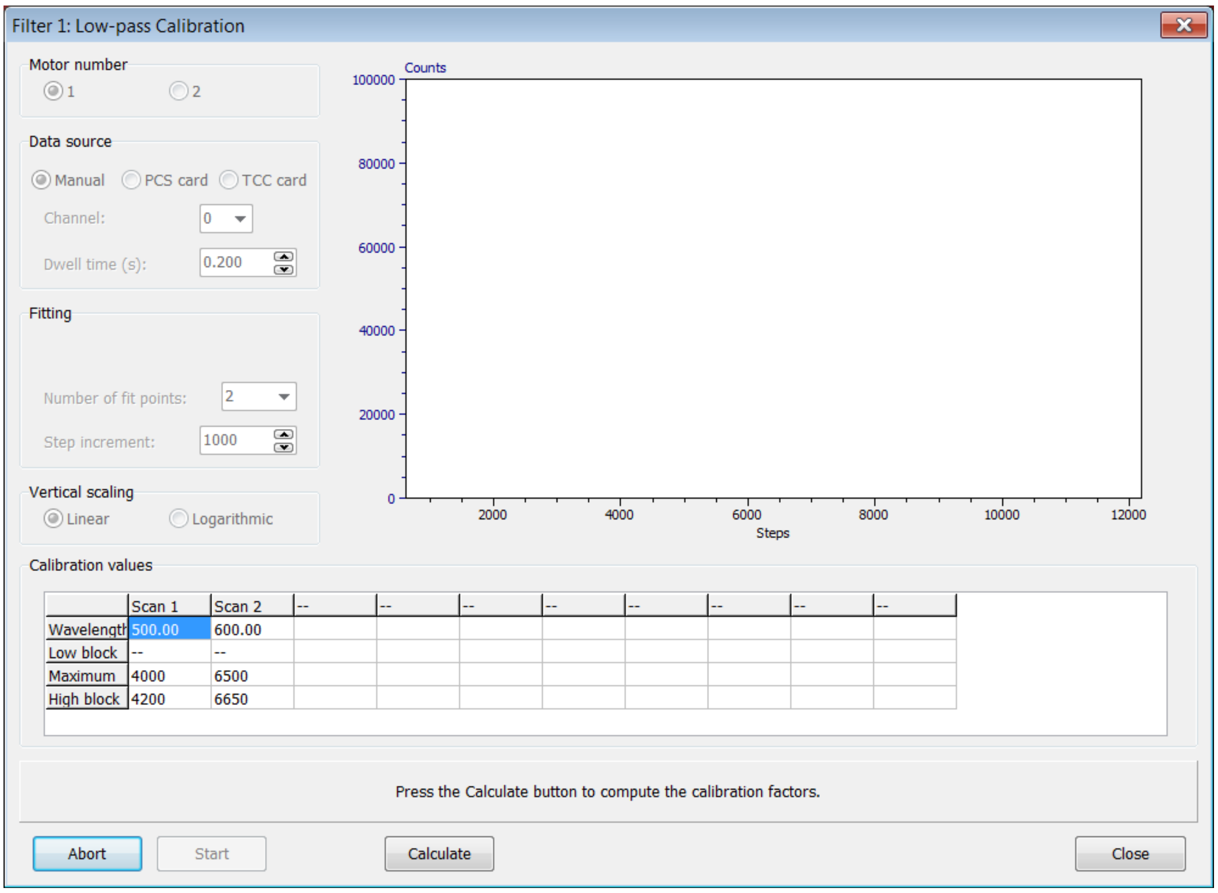
Vertical scaling: This changes the vertical scaling between linear and logarithmic (if appropriate).

The available buttons are:

Start: Pressing this button will commence the calibration procedure and the user should follow the instructions provided on the message bar.

Cancel: This button is used to close the form without updating the calibration values.

When the measuring process has completed the layout of the form is modified to that shown in Figure 4.



Filter 1: Low-pass Calibration

Motor number
[✓ 1] 2

Data source
[✓ Manual] PCS card TCC card
Channel: 0
Dwell time (s): 0.200

Fitting
Number of fit points: 2
Step increment: 1000

Vertical scaling
[✓ Linear] Logarithmic

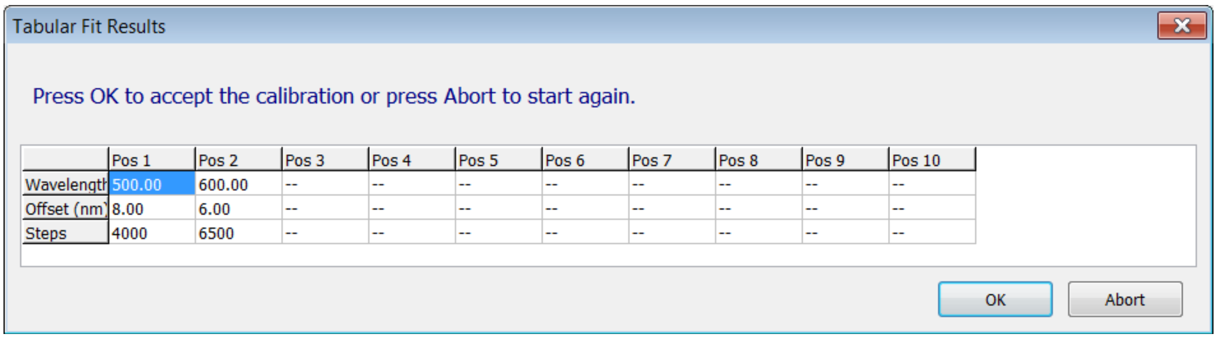
Calibration values

	Scan 1	Scan 2	--	--	--	--	--	--	--	--
Wavelengt	500.00	600.00								
Low block	--	--								
Maximum	4000	6500								
High block	4200	6650								

Press the Calculate button to compute the calibration factors.

Figure 4: Filter calibration calculation dialog box (for a low-pass filter).

This version of the calibration form shows the position values that have been obtained from the calibration procedure. If they are considered reasonable then the **Calculate** button should be pressed in order to start the calculation of the calibration values. Once the calculation is complete the form shown in Figure 5 is displayed.


Tabular Fit Results

Press OK to accept the calibration or press Abort to start again.

	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7	Pos 8	Pos 9	Pos 10
Wavelength	500.00	600.00	--	--	--	--	--	--	--	--
Offset (nm)	8.00	6.00	--	--	--	--	--	--	--	--
Steps	4000	6500	--	--	--	--	--	--	--	--

Figure 5: Filter calibration fit dialog box (for a low-pass filter).

The purpose of this form is to give the user the opportunity to accept (by pressing the **OK** button) or reject (by pressing the **Abort** button) the proposed calibration values.

3 SDK Functions

function ConfigureModule: Integer; stdcall;

Description: The purpose of this function is to configure the motor drives of the SuperChrome module. It is separate from the TSelector object as the configuration can be changed irrespective of whether there is hardware present.

Parameters: None.

function EditCalibration: Integer; stdcall;

Description: This function is used to edit the current calibration values.

Parameters: None.

function GetCurrentBw(BW: pDouble): Integer; stdcall;

Description: This function is used to return the current bandwidth for a dual filter.

Parameters: BW will contain the filter bandwidth.

function GetCurrentDistance(FilterNum: Integer; Dist: pDouble): Integer; stdcall;

Description: This function is used to return the current distance from the home position (in mm) for the specified filter.

Parameters: FilterNum is the number of the filter (1 or 2) being queried.

Dist will contain the distance value.

function GetCurrentSteps(FilterNum: Integer; Steps: pInteger): Integer; stdcall;

Description: This function is used to return the current distance from the home position (in motor steps) for the specified filter.

Parameters: FilterNum is the number of the filter (1 or 2) being queried.

Steps will contain the position in steps.

function GetCurrentWave(FilterNum: Integer; Wave: pDouble): Integer; stdcall;

Description: This function is used to return the current wavelength of the specified filter.

Parameters: FilterNum is the number of the filter (1 or 2) being queried.

Wave will contain the position in nanometres according to the current calibration values.

function GetCurrentWaveDual(DualWave: pDouble): Integer; stdcall;

Description: This function is used to return the current wavelength for a dual filter.

Parameters: DualWave will contain the combined wavelength in nanometres according to the current calibration values.

function GetSerialNum(SerNum: pAnsiChar): Integer; stdcall;

Description: This function is used to return the serial number of a filter module.

Parameters: SerNum will contain the serial number of the module. This will normally be less than 20 characters long, so the calling programme will have to allocate sufficient memory for the parameter in advance of calling the function.

function GetDualWaveMax(WaveMax: pDouble): Integer; stdcall;

Description: This function is used to return the common wavelength maximum for a dual filter.

Parameters: WaveMax will contain the maximum wavelength obtainable when both filters are being used in combination.

function GetDualWaveMin(WaveMin: pDouble): Integer; stdcall;

Description: This function is used to return the common wavelength minimum for a dual filter.

Parameters: WaveMin will contain the minimum wavelength obtainable when both filters are being used in combination.

function GetFilterActive(FilterNum: Integer; Active: pInteger): Integer; stdcall;

Description: This function is used to return whether the specified filter is active (i.e. is in the beam path. Zero is inactive; otherwise active.

Parameters: FilterNum is the number of the filter (1 or 2) being queried.

Active will contain the active status of the filter.

function GetFilterCalibrated(FilterNum: Integer; Cal: pInteger): Integer; stdcall;

Description:	This function is used to return whether the specified filter has been calibrated. Zero is uncalibrated; otherwise calibrated.
Parameters:	FilterNum is the number of the filter (1 or 2) being queried.
	Cal will contain the calibration status of the filter.

function GetFilterDual(IsDual: pInteger): Integer; stdcall;

Description:	This function is used to return whether the specified filter is a dual filter (i.e. has two separate filters driven by different motors. Zero is false; otherwise true.
Parameters:	IsDual will contain the dual filter status of the filter.

function GetWaveMax(FilterNum: Integer; WaveMax: pDouble): Integer; stdcall;

Description:	This function is used to return the wavelength maximum for the specified filter.
Parameters:	FilterNum is the number of the filter (1 or 2) being queried.
	WaveMax will contain the maximum wavelength for the filter as set in the configuration procedure.

function GetWaveMin(FilterNum: Integer; WaveMin: pDouble): Integer; stdcall;

Description:	This function is used to return the wavelength minimum for the specified filter.
Parameters:	FilterNum is the number of the filter (1 or 2) being queried.
	WaveMin will contain the minimum wavelength for the filter as set in the configuration procedure.

function InitialiseDll(AppHandle: HWND): Integer; stdcall;

Description:	This function is used to pass the application handle to the DLL.
Parameters:	AppHandle is the windows of the external programme or window calling rge DLL.

function Initialise: Integer; stdcall;

Description:	This function attempts to initialise the filter device by first creating the appropriate filter object and then trying to initialise it.
Parameters:	None.

function MoveDistance(FilterNum: Integer; Dist: Double): Integer; stdcall;

Description: This function is used to move the specified filter to the passed position in mm.

Parameters: FilterNum is the number of the filter (1 or 2) to be moved.

Dist is the position that the filter is to be moved to.

function MoveOutOfPath(FilterNum: Integer): Integer; stdcall;

Description: This function is used to move the specified filter out of the beam path.

Parameters: FilterNum is the number of the filter (1 or 2) to be moved.

function MoveSteps(FilterNum, Steps: Integer): Integer; stdcall;

Description: This function is used to move the specified filter to the passed position in steps.

Parameters: FilterNum is the number of the filter (1 or 2) to be moved.

Steps is the position that the filter is to be moved to.

function MoveSyncWave(Lambda1, Lambda2: Double): Integer; stdcall;

Description: This function is used to perform a synchronous wavelength move for a dual filter.

Parameters: Lambda1 is the wavelength for filter 1.

Lambda2 is the wavelength for filter 2.

function MoveSyncWaveAndBw(Lambda, BW: Double): Integer; stdcall;

Description: This function is used to move a dual filter to a specified wavelength and bandwidth combination.

Parameters: Lambda is the wavelength for the filter combination.

BW is the bandwidth for the filter combination.

function MoveWavelength(FilterNum: Integer; Lambda: Double): Integer; stdcall;

Description: This function is used to move the specified filter to the passed position in nm.

Parameters: FilterNum is the number of the filter (1 or 2) to be moved.

Lambda is the wavelength that the filter is to be moved to.

procedure ReleaseDll; stdcall;

Description: This procedure is used to free the local objects prior to the DLL being closed.

Parameters: None.

function RunCalibration: Integer; stdcall;

Description: This function is used to run a calibration process.

Parameters: None

4 Data Types

The data types and calling convention used in the exportable functions are:

Delphi type	**C Type**	**Description**
Integer	int, long int	32-bit signed integer
pInteger	*int	Pointer to a 32-bit signed integer
Double	double	Double precision floating-point value (8 bytes)
pDouble	*double	Pointer to a double precision floating point value
pAnsiChar	*char	Pointer to a string-type variable
HWND	HWND	Handle to a window
stdcall	_stdcall, WINAPI	convention usually used to call Win32 API functions

5 Return and Error Codes

Code	**Description**
0	Successful operation
{ DLL related }	
-4101	Filter DLL could not be found.
-4102	Filter DLL could not be created.
-4103	Filter DLL is not initialised.
-4104	Unhandled exception raised by filter DLL.
-4106	No configuration file has been generated.
-4108	Filter object is not initialised.
{ log file related }	
-4110	Exception occurred when writing to filter log file.
-4112	Filter log file could not be opened.
-4113	Filter log file could not be closed.
{ device and positioning errors }	
-4120	Device invalid for requested operation.
-4121	Filter type is invalid.
-4122	Filter position is invalid.
-4123	Filter wavelength is invalid.
-4124	Filter step value is invalid.
-4127	Fitting algorithm failed.
-4128	Filter motor is not calibrated.
{ USB error codes }	
-4130	Exception when accessing filter via USB interface.
-4131	Unable to create filter via USB interface.
-4132	Unable to open filter via USB interface.
-4133	Unable to access filter via USB interface.
-4135	Filter read failed via USB interface.
-4136	Filter write failed via USB interface.
-4138	The maximum number of polls was exceeded
-4139	Invalid message received from filter via USB interface.
{ Controller error codes }	
-4150	The controller position is invalid.
-4152	Measurement card not detected.
-4155	Abort button pressed.
-4158	Data conversion error (exception raised).

6 Sample Code

The segments of code shown here are written in Delphi; they can be easily translated into other common languages such as C.

The SuperChrome SDK is installed along with the SuperChrome software and by default is placed in the ‘C:\Program Files\Fianium\SuperChrome’ folder. Software using the DLL should look for it in this folder. Either static or dynamic linking can be made to the DLL.

An example of initialising the DLL and the SuperChrome module, including moving to a specified wavelength and bandwidth is:

```
function TSuperChromeMainForm.InitialiseFilterUnit: Integer;
var
  AppFolder: AnsiString;
  FilePath: AnsiString;
begin
 { initialise DLL interface }
  AppFolder := ExtractFileDir(Application.ExeName);
  FilePath := AppFolder + '\' + 'SuperChromeSDK.dll';
  if FileExists(FilePath) then
  begin
    fSuperChrome := TSuperChromeDLL.Create(Application.Handle, FilePath);
    Result := IfThen(fSuperChrome.Initialised, NoError, errFilterDllNotInit);
  end
  else
    Result := errFilterNoDll;
 { create filter device }
  if Result = NoError then
  begin
    Application.MessageBox(pAnsiChar(cInitMsg), 'SuperChrome Initialisation',
      MB_ICONWARNING + MB_OK);
    Result := fSuperChrome.Initialise;
  end;
  { move to initial position }
  if Result = NoError then
    Result := fSuperChrome.MoveSyncWaveAndBW(500.0, 25.0);
  fInitialised := (Result = NoError);
end;
```

In the above code segment the ExtractFileDir function is a wrapper for a Windows API call to retrieve the folder that the programme is running from.

A Delphi implementation of the DLL function declarations for dynamic linking is:

```
type
  { SuperChrome DLL type declarations }
  TscConfigureModule = function: Integer; stdcall;
  TscEditCalibration = function: Integer; stdcall;
  TscGetCurrentBw = function(BW: pDouble): Integer; stdcall;
  TscGetCurrentDistance = function(FilterNum: Integer;
   Dist: pDouble): Integer; stdcall;
  TscGetCurrentSteps = function(FilterNum: Integer;
   Steps: pInteger): Integer; stdcall;
  TscGetCurrentWave = function(FilterNum: Integer;
   Wave: pDouble): Integer; stdcall;
  TscGetCurrentWaveDual = function(DualWave: pDouble): Integer; stdcall;
  TscGetDualWaveMax = function(WaveMax: pDouble): Integer; stdcall;
  TscGetDualWaveMin = function(WaveMin: pDouble): Integer; stdcall;
  TscGetFilterActive = function(FilterNum: Integer;
   Active: pInteger): Integer; stdcall;
  TscGetFilterCalibrated = function(FilterNum: Integer;
   Cal: pInteger): Integer; stdcall;
  TscGetFilterDual = function(IsDual: pInteger): Integer; stdcall;
  TscGetWaveMax = function(FilterNum: Integer;
   WaveMax: pDouble): Integer; stdcall;
  TscGetWaveMin = function(FilterNum: Integer;
   WaveMin: pDouble): Integer; stdcall;
  TscInitialiseDll = function(AppHandle: HWND): Integer; stdcall;
  TscInitialise = function: Integer; stdcall;
  TscMoveDistance = function(FilterNum: Integer; Dist: Double): Integer; stdcall;
  TscMoveOutOfPath = function(FilterNum: Integer): Integer; stdcall;
  TscMoveSteps = function(FilterNum, Steps: Integer): Integer; stdcall;
  TscMoveSyncWave = function(Lambda1, Lambda2: Double): Integer; stdcall;
  TscMoveSyncWaveAndBw = function(Lambda, BW: Double): Integer; stdcall;
  TscMoveWavelength = function(FilterNum: Integer; Lambda: Double): Integer;
   stdcall;
  TscReleaseDll = procedure; stdcall;
  TscRunCalibration = function: Integer; stdcall;

  TSuperChromeDLL = class(TObject)
  protected
   fConfigureModule: TscConfigureModule;
   fEditCalibration: TscEditCalibration;
   fGetCurrentBw: TscGetCurrentBw;
   fGetCurrentDistance: TscGetCurrentDistance;
   fGetCurrentSteps: TscGetCurrentSteps;
   fGetCurrentWave: TscGetCurrentWave;
   fGetCurrentWaveDual: TscGetCurrentWaveDual;
   fGetDualWaveMax: TscGetDualWaveMax;
   fGetDualWaveMin: TscGetDualWaveMin;
   fGetFilterActive: TscGetFilterActive;
   fGetFilterCalibrated: TscGetFilterCalibrated;
   fGetFilterDual: TscGetFilterDual;
   fGetWaveMax: TscGetWaveMax;
   fGetWaveMin: TscGetWaveMin;
   fInitialise: TscInitialise;
   fInitialised: Boolean;
   fInitialiseDll: TscInitialiseDll;
   fMoveDistance: TscMoveDistance;
   fMoveOutOfPath: TscMoveOutOfPath;
   fMoveSteps: TscMoveSteps;
   fMoveSyncWave: TscMoveSyncWave;
   fMoveSyncWaveAndBw: TscMoveSyncWaveAndBw;
   fMoveWavelength: TscMoveWavelength;
   fReleaseDll: TscReleaseDll;
   fRunCalibration: TscRunCalibration;
   procedure GetDLLProcs; override;
  public
   constructor Create(AppHandle: THandle; DllFileName: AnsiString);
   destructor Destroy; override;
   property ConfigureModule: TscConfigureModule read fConfigureModule;
   property EditCalibration: TscEditCalibration read fEditCalibration;
   property GetCurrentBw: TscGetCurrentBw read fGetCurrentBw;
   property GetCurrentDistance: TscGetCurrentDistance read fGetCurrentDistance;
   property GetCurrentSteps: TscGetCurrentSteps read fGetCurrentSteps;
   property GetCurrentWave: TscGetCurrentWave read fGetCurrentWave;
   property GetCurrentWaveDual: TscGetCurrentWaveDual read
     fGetCurrentWaveDual;
   property GetDualWaveMax: TscGetDualWaveMax read fGetDualWaveMax;
   property GetDualWaveMin: TscGetDualWaveMin read fGetDualWaveMin;
   property GetFilterActive: TscGetFilterActive read fGetFilterActive;
   property GetFilterCalibrated: TscGetFilterCalibrated read fGetFilterCalibrated;
   property GetFilterDual: TscGetFilterDual read fGetFilterDual;
   property GetWaveMax: TscGetWaveMax read fGetWaveMax;
   property GetWaveMin: TscGetWaveMin read fGetWaveMin;
   property Initialise: TscInitialise read fInitialise;
   property Initialised: Boolean read fInitialised;
   property InitialiseDll: TscInitialiseDll read fInitialiseDll;
   property MoveDistance: TscMoveDistance read fMoveDistance;
   property MoveOutOfPath: TscMoveOutOfPath read fMoveOutOfPath;
   property MoveSteps: TscMoveSteps read fMoveSteps;
   property MoveSyncWave: TscMoveSyncWave read fMoveSyncWave;
   property MoveSyncWaveAndBw: TscMoveSyncWaveAndBw read
     fMoveSyncWaveAndBw;
   property MoveWavelength: TscMoveWavelength read fMoveWavelength;
   property ReleaseDll: TscReleaseDll read fReleaseDll;
   property RunCalibration: TscRunCalibration read fRunCalibration;
  end;
```

The Create constructor is:

```
constructor TSuperChromeDLL.Create(AppHandle: THandle; DllFileName: AnsiString);
begin
  try
    DLLName := aDLLName;
    if DLLName <> '' then
    begin
      DLLHandle := LoadLibrary(pAnsiChar(DLLName));
      if DLLHandle = 0 then
        raise(EDLLException.Create('Could not find DLL: ''' + DLLName + ''' '));
    end
    else
      DLLHandle := GetModuleHandle(nil); //Get handle to calling program
     GetDLLProcs;
     fInitialised := (fInitialiseDll(AppHandle) = NoError);
  except
    on E: EDLLGetProcException do
   begin
     MessageDlg(E.Message, mtError, [mbOK], 0);
     fInitialised := False;
   end;
  else
    fInitialised := False;
  end;
end;
```

The destructor is given by:

```
destructor TSuperChromeDLL.Destroy;
begin
  if fInitialised then
  begin
    fReleaseDll;
    fInitialised := False;
  end;
   if DLLHandle <> 0 then
  begin
    if not FreeLibrary(DLLHandle) then
      raise(EDLLException.Create('Error Unloading DLL: ''' + DLLName + '''. '));
  end;
end;
```

In this constructor dynamic linking to the DLL is made by using the Windows API functions LoadLibrary, and GetModuleHandle. The destructor uses FreeLibrary. The GetDllProcs procedure uses the GetProcAddress API function and is:

```
procedure TSuperChromeDLL.GetDLLProcs;
begin
  fConfigureModule := GetProcAddress(DLLHandle, pAnsiChar('ConfigureModule'));
  fEditCalibration := GetProcAddress(DLLHandle, pAnsiChar('EditCalibration'));
  fGetCurrentBw := GetProcAddress(DLLHandle, pAnsiChar('GetCurrentBw'));
  fGetCurrentDistance := GetProcAddress(DLLHandle,
    pAnsiChar('GetCurrentDistance'));
  fGetCurrentSteps := GetProcAddress(DLLHandle, pAnsiChar('GetCurrentSteps'));
  fGetCurrentWave := GetProcAddress(DLLHandle, pAnsiChar('GetCurrentWave'));
  fGetCurrentWaveDual := GetProcAddress(DLLHandle,
    pAnsiChar('GetCurrentWaveDual'));
  fGetDualWaveMax := GetProcAddress(DLLHandle, pAnsiChar('GetDualWaveMax'));
  fGetDualWaveMin := GetProcAddress(DLLHandle, pAnsiChar('GetDualWaveMin'));
  fGetFilterActive := GetProcAddress(DLLHandle, pAnsiChar('GetFilterActive'));
  fGetFilterCalibrated := GetProcAddress(DLLHandle,
    pAnsiChar('GetFilterCalibrated'));
  fGetFilterDual := GetProcAddress(DLLHandle, pAnsiChar('GetFilterDual'));
  fGetWaveMax := GetProcAddress(DLLHandle, pAnsiChar('GetWaveMax'));
  fGetWaveMin := GetProcAddress(DLLHandle, pAnsiChar('GetWaveMin'));
  fInitialise := GetProcAddress(DLLHandle, pAnsiChar('Initialise'));
  fInitialiseDll := GetProcAddress(DLLHandle, pAnsiChar('InitialiseDll'));
  fMoveDistance := GetProcAddress(DLLHandle, pAnsiChar('MoveDistance'));
  fMoveOutOfPath := GetProcAddress(DLLHandle, pAnsiChar('MoveOutOfPath'));
  fMoveSteps := GetProcAddress(DLLHandle, pAnsiChar('MoveSteps'));
  fMoveSyncWave := GetProcAddress(DLLHandle, pAnsiChar('MoveSyncWave'));
  fMoveSyncWaveAndBw := GetProcAddress(DLLHandle,
    pAnsiChar('MoveSyncWaveAndBw'));
  fMoveWavelength := GetProcAddress(DLLHandle, pAnsiChar('MoveWavelength'));
  fReleaseDll := GetProcAddress(DLLHandle, pAnsiChar('ReleaseDll'));
  fRunCalibration := GetProcAddress(DLLHandle, pAnsiChar('RunCalibration'));
end;
```