Chapter 5 - Frequency Counters

5.1 - Operating in a Windows Environment

5.1.1 - Referencing the DLL Library

The DLL file is installed in the host PC's system folders using the steps outlined above. In order to use the DLL functionality, some programming environments will require the user to set a reference to the relevant DLL file, usually through a built in GUI in the programming environment.

Once this is done, the user just needs to instantiate a new instance of the USB_FreqCounter object in order to use the frequency counter functions. The details of this vary greatly between programming environments and languages but Mini-Circuits can provide detailed support on request. A new frequency counter object would need to be initialized for every USB frequency counter that the user wishes to control. In the following examples, MyPTE1 and MyPTE2 will be used as names of 2 declared sensor objects.

Examples

Visual Basic

```
Public MyPTE1 As New mcl_FreqCounter.USB_ FreqCounter
        ' Initialize new frequency counter object, assign to MyPTE1
Public MyPTE2 As New mcl_FreqCounter.USB_ FreqCounter
        ' Initialize new frequency counter object, assign to MyPTE2
```

Visual C++

```
USB_FreqCounter ^MyPTE1 = gcnew USB_FreqCounter();
        // Initialize new frequency counter instance, assign to MyPTE1
USB_FreqCounter ^MyPTE2 = gcnew USB_FreqCounter();
        // Initialize new frequency counter instance, assign to MyPTE2
```

Visual C#

```
MCL_FreqCounter.USB_FreqCounter MyPTE1 = new
                                        _ MCL_FreqCounter.USB_FreqCounter();
        // Initialize new frequency counter instance, assign to MyPTE1
MCL_FreqCounter.USB_FreqCounter MyPTE2 = new
                                        _ MCL_FreqCounter.USB_FreqCounter();
        // Initialize new frequency counter instance, assign to MyPTE2
```

Matlab

```
MyPTE1 =actxserver('MCL_FreqCounter.USB_FreqCounter')
        % Initialize new frequency counter instance, assign to MyPTE1
MyPTE2 =actxserver('MCL_FreqCounter.USB_FreqCounter')
        % Initialize new frequency counter instance, assign to MyPTE2
```

5.1.2 - Summary of DLL Functions

The following functions are defined in both of the DLL files. Please see the following sections for a full description of their structure and implementation.

a) Short Connect (Optional String SN)
b) Void Disconnect ()
c) Short Get_Available_SN_List (String* SN_List)
d) Short GetStatus ()
e) Short Read_ModelName (String ModelName)
f) Short Read_SN (String SN)
g) Short ReadFreq (Double RetFreq)
h) Short SetRange (Short RequestedRange)
i) String GetRange ()
j) Short SetSampleTime (Float SampleTime_sec)
k) Short GetFirmware ()

5.1.3 - Detailed Description of DLL Functions

5.1.3 (a) - Connect to Frequency Counter

Declaration

```
Short Connect(Optional String SN)
```

Description

This function is called to initialize the connection to a USB frequency counter. If multiple sensors are connected to the same computer, then the serial number should be included, otherwise this can be omitted. The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the sensor is no longer needed. The sensor should be disconnected on completion of the program using the Disconnect function.

Parameters

Data Type	Variable	Description
String	SN	Optional. A string containing the serial number of the USB control box. Can be omitted if only one control box is connected but must be included otherwise.

Return Values

Data Type	Value	Description
Short	0	No connection was possible
	1	Connection successfully established
	2	Device already connected
	3	Requested serial number is not available

Examples

Visual Basic
```
status = MyPTE1.Connect(SN)
```
Visual C++
```
status = MyPTE1->Connect(SN);
```
Visual C#
```
status = MyPTE1.Connect(SN);
```
Matlab
```
status = MyPTE1.Connect(SN)
```

See Also

Disconnect from Frequency Counter

5.1.3 (b) - Disconnect from Frequency Counter

Declaration

```
Void Disconnect()
```

Description

This function is called to close the connection to the frequency counter. It is strongly recommended that this function is used prior to ending the program. Failure to do so may result in a connection problem with the device. Should this occur, shut down the program and unplug the frequency counter from the computer, then reconnect the frequency counter before attempting to start again.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

Examples

Visual Basic
```
MyPTE1.Disconnect()
```
Visual C++
```
MyPTE1->Disconnect();
```
Visual C#
```
MyPTE1.Disconnect();
```
Matlab
```
MyPTE1.Disconnect
```

See Also

Connect to Frequency Counter

5.1.3 (c) - Get List of Connected Serial Numbers

Declaration

```
Short Get_Available_SN_List(String SN_List)
```

Description

This function takes a user defined variable and updates it with a list of serial numbers for all available (currently connected) control boxes.

Parameters

Data Type	Variable	Description
String	SN_List	Required. User defined variable which will be updated with a list of all connected serial numbers, separated by a single space character, for example “11110001 11110002 11110003”.

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Get_Available_SN_List(SN_List) > 0 Then
        array_SN() = Split(SN_List, " ")
                ' Split the list into an array of serial numbers
        For i As Integer = 0 To array_SN.Length - 1
                ' Loop through the array and use each serial number
        Next
End If
```

Visual C++

```
if (MyPTE1 ->Get_Available_SN_List(SN_List) > 0)
{
        // split the List into array of SN's
}
```

Visual C#

```
if (MyPTE1.Get_Available_SN_List(ref(SN_List)) > 0)
{
        // split the List into array of SN's
}
```

Matlab

```
[status, SN_List]= MyPTE1.Get_Available_SN_List(SN_List)
If status > 0 then
{
        % split the List into array of SN's
}
```

See Also

Connect to Frequency Counter
Read Serial Number of Frequency Counter

5.1.3 (d) - Get Status

Declaration

```
Short Get_Status()
```

Description

This function checks whether the USB connection to the frequency counter is still active.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	No connection
Short	1	USB connection to frequency counter is active

Examples

Visual Basic
```
status = MyPTE1.GetStatus
```
Visual C++
```
status = MyPTE1->GetStatus();
```
Visual C#
```
status = MyPTE1.GetStatus();
```
Matlab
```
status = MyPTE1.GetStatus
```

See Also

Connect to Frequency Counter

5.1.3 (e) - Read Model Name of Frequency Counter

Declaration

```
Short Read_ModelName(String ModelName)
```

Description

This function is called to determine the Mini-Circuits part number of the connected frequency counter.

Parameters

Data Type	Variable	Description
String	Model Name	Required. User defined variable which will be updated with the Mini-Circuits part number.

Return Values

Data Type	Value	Description
Short	0	Command failed
	Non zero	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
        MsgBox ("The connected generator is " & ModelName)
                ' Display a message stating the model name
End If
```

Visual C++

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
        MessageBox::Show("The connected generator is " + ModelName);
                // Display a message stating the model name
}
```

Visual C#

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
        MessageBox.Show("The connected generator is " + ModelName);
                // Display a message stating the model name
}
```

Matlab

```
[status, ModelName]= MyPTE1.Read_ModelName(ModelName)
If status > 0 then
{
        msgbox('The connected generator is ', ModelName)
                % Display a message stating the model name
}
```

See Also

Read Serial Number of Frequency Counter

5.1.3 (f) - Read Serial Number of Frequency Counter

Declaration

```
Short Read_SN(String SN)
```

Description

This function is called to determine the serial number of the connected frequency counter.

Parameters

Data Type	Variable	Description
String	SN	Required. User defined variable which will be updated with the device serial number.

Return Values

Data Type	Value	Description
Short	0	Command failed
	Non zero	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_SN(SN) > 0 Then
        MsgBox ("The connected generator is " & SN)
                'Display a message stating the serial number
End If
```

Visual C++

```
if (MyPTE1->Read_SN(SN) > 0)
{
        MessageBox::Show("The connected generator is " + SN);
                // Display a message stating the serial number
}
```

Visual C#

```
if (MyPTE1.Read_SN(ref(SN)) > 0)
{
        MessageBox.Show("The connected generator is " + SN);
                // Display a message stating the serial number
}
```

Matlab

```
[status, SN]= MyPTE1.Read_SN(SN)
If status > 0 then
{
        msgbox('The connected generator is ', SN)
                % Display a message stating the serial number
}
```

See Also

Read Model Name of Frequency Counter

5.1.3 (g) - Read Frequency of Frequency Counter

Declaration

```
Short ReadFreq(Double RetFreq)
```

Description

This function returns the frequency in MHz of the signal at the frequency counter's RF input.

Parameters

Data Type	Variable	Description
Double	RetFreq	Required. User defined variable which will be updated with the current frequency reading in Hz

Return Values

Data Type	Value	Description
Short	0	Command failed
	Non zero	Command completed successfully

Examples

Visual Basic
```
status = MyPTE1.ReadFreq(Freq)
```
Visual C++
```
status = MyPTE1->ReadFreq(Freq);
```
Visual C#
```
status = MyPTE1.ReadFreq(ref(Freq));
```
Matlab
```
status = MyPTE1.ReadFreq(Freq)
```

See Also

Set Sample Time

5.1.3 (h) - Set Range of Frequency Counter

Declaration

```
Short SetRange(Short RequestedRange)
```

Description

This function sets the range of the frequency counter. By default the frequency counter is in "Auto Range" mode and will automatically set the correct frequency range, this process will take typically 50ms. If the frequency range of the input signal is known then the user can eliminate this delay by specifying the appropriate range. The options are:

Range	Minimum Frequency (MHz)	Maximum Frequency (MHz)
255 (Auto)	1	6000
1	1	40
2	40	190
3	190	1400
4	1400	6000

Parameters

Data Type	Variable	Description
Short	Requested _Range	Required. An integer value to set the range from 1 to 4, or 255 for "Auto Range"

Return Values

Data Type	Value	Description
Short	0	Command failed
	Non zero	Command completed successfully

Examples

Visual Basic
```
status = MyPTE1.SetRange(RequestedRange)
```
Visual C++
```
status = MyPTE1->SetRange(RequestedRange);
```
Visual C#
```
status = MyPTE1.SetRange(ref(RequestedRange));
```
Matlab
```
status = MyPTE1.SetRange(RequestedRange)
```

See Also

Get Range of Frequency Counter

5.1.3 (i) - Get Range of Frequency Counter

Declaration

```
String GetRange()
```

Description

This function returns the range setting that the frequency counter is currently using for measurements.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
String	Auto	The counter will automatically determine the best range setting
	1	Range 1; the counter is expecting a frequency from 1 to 40MHz
	2	Range 2; the counter is expecting a frequency from 40 to 190MHz
	3	Range 3; the counter is expecting a frequency from 190 to 1400MHz
	4	Range 4; the counter is expecting a frequency from 1400 to 6000MHz

Examples

Visual Basic
```
Range = MyPTE1.GetRange
```
Visual C++
```
Range = MyPTE1->GetRange();
```
Visual C#
```
Range = MyPTE1.GetRange();
```
Matlab
```
Range = MyPTE1.GetRange
```

See Also

Set Range of Frequency Counter

5.1.3 (j) - Set Sample Time

Declaration

```
Short SetSampleTime(Float SampleTime_sec)
```

Description

This function sets the sample time for the frequency measurements, in 0.1 second steps up to 1 second, or 1 second steps up to 3 seconds. The default sample time is 1 second.

Parameters

Data Type	Variable	Description
Float	SampleTime_sec	Required. Numeric value indicating the required sample time. Allowed values are 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3

Return Values

Data Type	Value	Description
Short	0	Command failed
	Non zero	Command completed successfully

Examples

Visual Basic

```
Status =  MyPTE1.SetSampleTime(SampleTime)
```

Visual C++

```
status = MyPTE1->SetSampleTime(SampleTime);
```

Visual C#

```
status = MyPTE1.SetSampleTime(ref(SampleTime));
```

Matlab

```
Status = MyPTE1.SetSampleTime(SampleTime)
```

See Also

Read Frequency of Frequency Counter

5.1.3 (k) - Get Firmware Version

Declaration

```
Short Get_Firmware()
```

Description

This function returns the internal firmware version of the frequency counter.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	Firmware	An integer value for the internal firmware version

Examples

Visual Basic
```
status = MyPTE1.GetFirmware()
```
Visual C++
```
status = MyPTE1->GetFirmware();
```
Visual C#
```
status = MyPTE1.GetFirmware();
```
Matlab
```
status = MyPTE1.GetFirmware()
```

5.2 - Operating in a Linux Environment

To open a connection to Mini-Circuits RF Frequency Counter, the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- Frequency counter Product ID: 0x10

Communication with the frequency counter is carried out by way of USB Interrupt. The transmitted and received buffer sizes are 64 Bytes each:

- Transmit Array = [Byte 0][Byte1][Byte2]…[Byte 63]
- Returned Array = [Byte 0][Byte1][Byte2]…[Byte 63]

In most cases, the full 64 byte buffer size is not needed so any unused bytes become “don’t care” bytes; they can take on any value without affecting the operation of the frequency counter.

A worked example is included in Appendix C of this document. The example uses the libhid and libusb libraries to interface with the frequency counter as a USB HID (Human Interface Device).

5.2.1 - Summary of Commands

The commands that can be sent to the frequency counter are summarized in the table below and detailed on the following pages.

#	Description	Command Code (Byte 0)
a	Get Device Model Name	40
b	Get Device Serial Number	41
c	Get Frequency and Range	2
d	Set Range	4
e	Set Sample Time	3

5.2.2 - Detailed Description of Commands

5.2.2 (a) - Get Device Model Name

Description

This function determines the Mini-Circuits part number of the connected frequency counter.

Send code 40 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are "don't care" bytes and can be any value.

The model name is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately preceding the first zero value byte. All subsequent bytes up to BYTE63 are "don't care" bytes and could be any value.

Transmit Array

Byte	Byte 0
Description	Code
Value	40

Returned Array

Byte	Byte 0	Byte 1	Byte 2	…	Byte (N-1)	Byte N
Description	Code	First Char	Second Char	…	Last Char	End Marker
Value	40	ASCII	ASCII	…	ASCII	0

Example

The following array would be returned for Mini-Circuits' UFC-6000 frequency counter. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5
Value	40	85	70	67	45	54
ASCII Character	N/A	U	F	C	-	6

Byte	Byte 6	Byte 7	Byte 8	Byte 9
Description	Char 6	Char 7	Char 8	End Marker
Value	48	48	48	0
ASCII Character	0	0	0	N/A

See Also

Get Device Serial Number

5.2.2 (b) - Get Device Serial Number

Description

This function determines the serial number of the connected frequency counter.

Send code 41 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are "don't care" bytes and can be any value.

The serial number is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately preceding the first zero value byte. All subsequent bytes up to BYTE63 are "don't care" bytes and could be any value.

Transmit Array

Byte	Byte 0
Description	Code
Value	41

Returned Array

Byte	Byte 0	Byte 1	Byte 2	…	Byte (N-1)	Byte N
Description	Code	First Char	Second Char	…	Last Char	End Marker
Value	41	ASCII	ASCII	…	ASCII	0

Example

The following example indicates that the current frequency counter has serial number 1100040023. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5
Value	41	49	49	48	48	48
ASCII Character	N/A	1	1	0	0	0

Byte	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
Description	Char 6	Char 7	Char 8	Char 9	Char 10	End Marker
Value	52	48	48	50	51	0
ASCII Character	4	0	0	2	3	N/A

See Also

Get Device Model Name

5.2.3 - Get Frequency and Range

Description

This function returns the current frequency measurement and the frequency counter's range setting.

The transmit array contains 2 in BYTE0. BYTE1 to BYTE63 are "don't care" bytes and could be any value. The returned array is made up of the following bytes:

- BYTE0
  - Code 2
- BYTE1 to BYTE16
  - Series of ASCII character codes representing the range setting in the format "Range: 0", padded with space characters
- BYTE17 to BYTE32
  - Series of ASCII character codes representing the measured frequency in the format "0000.0000 MHz", padded with space characters
- BYTE33 to BYTE63
  - Can be any value ("don't care" bytes)

Transmit Array

Byte	Byte 0
Description	Code
Value	2

Returned Array

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Range 1	Range 2	Range 3	Range 4	Range 5	Range 6
Value	2	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII

Byte	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13
Description	Range 7	Range 8	Range 9	Range 10	Range 11	Range 12	Range 13
Value	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII

Byte	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20
Description	Range 14	Range 15	Range 16	Freq 1	Freq 2	Freq 3	Freq 4
Value	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII

Byte	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27
Description	Freq 5	Freq 6	Freq 7	Freq 8	Freq 9	Freq 10	Freq 11
Value	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII

Byte	Byte 28	Byte 29	Byte 30	Byte 31	Byte 32
Description	Freq 12	Freq 13	Freq 14	Freq 15	Freq 16
Value	ASCII	ASCII	ASCII	ASCII	ASCII

Example

The following array would be returned for a frequency measurement of 300.0005MHz with the range at setting 3:

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Value	2	32	32	32	32	82	97
ASCII Character	N/A					R	a

Byte	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13
Value	110	103	101	58	32	51	32
ASCII Character	n	g	e	:		3	

Byte	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20
Value	32	32	32	32	51	48	48
ASCII Character					3	0	0

Byte	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27
Value	46	48	48	48	53	32	77
ASCII Character	.	0	0	0	5		M

Byte	Byte 28	Byte 29	Byte 30	Byte 31	Byte 32
Value	72	122	32	32	32
ASCII Character	H	z			

See Also

Set Range

5.2.3 (a) - Set Range

Description

This function sets the range of the frequency counter. By default the frequency counter is in "Auto Range" mode and will automatically set the correct frequency range, this process will take typically 50ms. If the frequency range of the input signal is known then the user can eliminate this delay by specifying the appropriate range. The options are:

Range	Minimum Frequency (MHz)	Minimum Frequency (MHz)
255 (Auto)	1	6000
1	1	40
2	40	190
3	190	1400
4	1400	6000

The transmit array is made up of the following bytes:

- BYTE0
  - Code 4
- BYTE1
  - An integer value from 1 to 4 to set the range, or 255 for "Auto Range"
- BYTE2 to BYTE63
  - Can be any value ("don't care" bytes)

Transmit Array

Byte	Byte 0	Byte 1
Description	Code	Range
Value	4	1-4 or 255

Returned Array

Byte	Byte 0
Description	Code
Value	4

Example

Send the following transmit array to set the frequency counter to "Auto Range":

Byte	Byte 0	Byte 1
Description	Code	Range
Value	4	255

See Also

Get Frequency and Range
Set Sample Time

5.2.3 (b) - Set Sample Time

Description

This function sets the sample time for the frequency measurements, in 0.1 second steps up to 1 second, or 1 second steps up to 3 seconds. The default sample time is 1 second.

The transmit array is made up of the following bytes:

- BYTE0
  - Code 3
- BYTE1
  - An integer value representing the sample time in seconds multiplied by 10. This allows a precision of 1 decimal place to be set.
- BYTE2 to BYTE63
  - Can be any value ("don't care" bytes)

Transmit Array

Byte	Byte 0	Byte 1
Description	Code	Sample Time*10
Value	103	1 to 30

Returned Array

Byte	Byte 0
Description	Code
Value	103

Example

To set the sample time to 0.3 seconds, multiply by 10 to get an integer 3 in BYTE1. The full transmit array would be:

Byte	Byte 0	Byte 1
Description	Code	Sample Time*10
Value	103	3

See Also

Set Range