Mini-Circuits Portable Test Equipment Programming Manual

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 in the Introduction to this manual. 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 frequency counter 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
```

Python

```
MyPTE1 = win32com.client.Dispatch('MCL_FreqCounter.USB_FreqCounter')
        ' Initialize new frequency counter object, assign to MyPTE1
MyPTE1 = win32com.client.Dispatch('MCL_FreqCounter.USB_FreqCounter')
        ' Initialize new frequency counter object, 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 GetSampleTime(Float ST)
l) Short GetFirmwareInfo (Short FirmwareID, String FirmwareRev, Short FirmwareNo)
m) Short GetFirmware ()

5.1.3 - Detailed Description of DLL Functions

5.1.3 (a) - Connect

Declaration

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

Description

Opens a connection to the USB frequency counter. If multiple frequency counters 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 counter is no longer needed. The counter 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)
```
Python
```
status = MyPTE1.Connect(SN)
```

See Also

Disconnect

5.1.3 (b) - Disconnect

Declaration

```
Void Disconnect()
```

Description

Closes 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
```
Python
```
MyPTE1.Disconnect
```

See Also

Connect

5.1.3 (c) - Get List of Connected Serial Numbers

Declaration

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

Description

Returns a list of serial numbers for all available (currently connected) frequency counters.

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
}
```

Python

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

See Also

Connect
Read Serial Number

5.1.3 (d) - Get Status

Declaration

```
Short Get_Status()
```

Description

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
```
Python
```
status = MyPTE1.GetStatus
```

See Also

Connect

5.1.3 (e) - Read Model Name

Declaration

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

Description

Returns the full 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
	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
        MsgBox ("The connected frequency counter is " & ModelName)
End If
```

Visual C++

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
        MessageBox::Show("The connected frequency counter is " + ModelName);
}
```

Visual C#

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
        MessageBox.Show("The connected frequency counter is " + ModelName);
}
```

Matlab

```
[status, ModelName]= MyPTE1.Read_ModelName(ModelName)
If status > 0 then
{
        msgbox('The connected frequency counter is ', ModelName)
}
```

Python

```
status = MyPTE1.Read_ModelName(ModelName)
If status > 0:
        print('The connected frequency counter is', ModelName)
```

See Also

Read Serial Number

5.1.3 (f) - Read Serial Number

Declaration

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

Description

Returns 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
	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_SN(SN) > 0 Then
        MsgBox ("The connected frequency counter is " & SN)
End If
```

Visual C++

```
if (MyPTE1->Read_SN(SN) > 0)
{
        MessageBox::Show("The connected frequency counter is " + SN);
}
```

Visual C#

```
if (MyPTE1.Read_SN(ref(SN)) > 0)
{
        MessageBox.Show("The connected frequency counter is " + SN);
}
```

Matlab

```
[status, SN]= MyPTE1.Read_SN(SN)
If status > 0 then
{
        msgbox('The connected frequency counter is ', SN)
}
```

Python

```
status = MyPTE1.Read_SN(SN)
If status > 0:
        print('The connected frequency counter is', SN)
```

See Also

Read Model Name

5.1.3 (g) - Read Frequency

Declaration

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

Description

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
	1	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)
```
Python
```
status = MyPTE1.ReadFreq(Freq)
```

See Also

Set Sample Time

5.1.3 (h) - Set Range

Declaration

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

Description

Sets the measurement 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
	1	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)
```
Python
```
status = MyPTE1.SetRange(RequestedRange)
```

See Also

Get Range

5.1.3 (i) - Get Range

Declaration

```
String GetRange()
```

Description

Returns the measurement range of the frequency counter.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
String	Range: Auto	The counter will automatically select the appropriate measurement range
	Range: 1	The counter is expecting a frequency from 1 to 40MHz
	Range: 2	The counter is expecting a frequency from 40 to 190MHz
	Range: 3	The counter is expecting a frequency from 190 to 1400MHz
	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
```
Python
```
Range = MyPTE1.GetRange
```

See Also

Set Range

5.1.3 (j) - Set Sample Time

Declaration

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

Description

Sets the sample time to be used for frequency measurements. The allowed range is 0.1 to 3 seconds, in 0.1 second steps. The default sample time is 1 second.

Parameters

Data Type	Variable	Description
Float	SampleTime_sec	Required. Numeric value indicating the required sample time.

Return Values

Data Type	Value	Description
Short	0	Command failed
	1	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)
```
Python
```
Status = MyPTE1.SetSampleTime(SampleTime)
```

See Also

Read Frequency
Get Sample Time

5.1.3 (k) - Get Sample Time

Declaration

```
Short GetSampleTime(Float ST)
```

Description

Returns the sample time for frequency measurements, in seconds.

Parameters

Data Type	Variable	Description
Float	ST	Required. User defined variable which will be updated with the sample time in seconds.

Return Values

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

Examples

Visual Basic
```
Status = MyPTE1.GetSampleTime(ST)
```
Visual C++
```
Status = MyPTE1->GetSampleTime(ST);
```
Visual C#
```
Status = MyPTE1.GetSampleTime(ST);
```
Matlab
```
[Status, ST = MyPTE1.GetSampleTime(ST)
```
Python
```
Status = MyPTE1.GetSampleTime(ST)
```

See Also

Read Frequency
Set Sample Time

5.1.3 (I) - Get Firmware

Declaration

```
Short GetFirmwareInfo(Int FirmwareID, String FirmwareRev,
                                                    _ Int FirmwareNo)
```

Description

Returns the internal firmware version of the frequency counter.

Parameters

Data Type	Variable	Description
Short	FirmwareID	Required. User defined variable for factory use only.
String	FirmwareRev	Required. User defined variable which will be updated with the current firmware version, for example “B3”.
Short	FirmwareNo	Required. User defined variable for factory use only.

Return Values

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

Examples

Visual Basic

```
If MyPTE1.GetFirmwareInfo(fID, fRev, fNo) > 0 Then
        MsgBox ("Firmware version is " & fRev)
End If
```

Visual C++

```
if (MyPTE1->GetFirmwareInfo(fID, fRev, fNo) > 0 )
{
        MessageBox::Show("Firmware version is " + fRev);
}
```

Visual C#

```
if (MyPTE1.GetFirmwareInfo(ref(fID, fRev, fNo)) > 0 )
{
        MessageBox.Show("Firmware version is " + fRev);
}
```

Matlab

```
[status, fID, fRev, fNo]=MyPTE1.GetFirmwareInfo(fID, fRev, fNo)
If status > 0 then
{
        msgbox('Firmware version is ', fRev)
}
```

Python

```
If MyPTE1.GetFirmwareInfo(fID, fRev, fNo) > 0:
        print "Firmware version is", fRev
```

5.1.3 (m) - Get Firmware Version (Antiquated)

Declaration

```
Short Get_Firmware()
```

Description

This function is antiquated, GetFirmwareInfo should be used. Get_Firmware returns the internal firmware version number 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()
```
Python
```
status = MyPTE1.GetFirmware()
```

See Also

Get Firmware

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
f	Get Sample Time	33
g	Get Firmware	99

5.2.2 - Detailed Description of Commands

5.2.2 (a) - Get Device Model Name

Description

Returns the full Mini-Circuits part number of the connected frequency counter.

Transmit Array

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1 to (n-1)	Model Name	Series of bytes containing the ASCII code for each character in the model name
n	0	Zero value byte to indicate the end of the model name
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

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

Returns the serial number of the connected switch matrix.

Transmit Array

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1 to (n-1)	Serial Number	Series of bytes containing the ASCII code for each character in the serial number
n	0	Zero value byte to indicate the end of the serial number
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

Example

The following example indicates that the 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

5.2.2 (c) - Get Frequency and Range

Description

Returns the current frequency measurement and the frequency counter's range setting.

Transmit Array

Byte	Data	Description
0	2	Interrupt code for Get Frequency and Range
1 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	2	Interrupt code for Get Frequency and Range
1 - 16	Range_Chars	The measurement range (in the format "Range: 0" padded with space characters) returned as a series of ASCII character codes, 1 per byte.
17 - 32	Freq_Chars	The measured frequency (in the format "0000.0000 MHz" padded with space characters) returned as a series of ASCII character codes, 1 per byte.
33-63	Not significant	"Don't care" bytes, could be any value

Example

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

Byte	Data	Description
0	2	Interrupt code for Get Frequency and Range
1	32	Range: ASCII character code for " "
2	32	Range: ASCII character code for " "
3	32	Range: ASCII character code for " "
4	32	Range: ASCII character code for " "
5	82	Range: ASCII character code for "R"
6	97	Range: ASCII character code for "a"
7	110	Range: ASCII character code for "n"
8	103	Range: ASCII character code for "g"
9	101	Range: ASCII character code for "e"
10	58	Range: ASCII character code for ":"
11	32	Range: ASCII character code for " "
12	51	Range: ASCII character code for "3"
13	32	Range: ASCII character code for " "
14	32	Range: ASCII character code for " "
15	32	Range: ASCII character code for " "
16	32	Range: ASCII character code for " "
17	32	Frequency: ASCII character code for " "
18	51	Frequency: ASCII character code for "3"
19	48	Frequency: ASCII character code for "0"
20	48	Frequency: ASCII character code for "0"
21	46	Frequency: ASCII character code for "."
22	48	Frequency: ASCII character code for "0"
23	48	Frequency: ASCII character code for "0"
24	48	Frequency: ASCII character code for "0"
25	53	Frequency: ASCII character code for "5"
26	32	Frequency: ASCII character code for " "
27	77	Frequency: ASCII character code for "M"
28	72	Frequency: ASCII character code for "H"
29	122	Frequency: ASCII character code for "z"
30	32	Frequency: ASCII character code for " "
31	32	Frequency: ASCII character code for " "
32	32	Frequency: ASCII character code for " "
33-63	Not significant	"Don't care" bytes, could be any value

See Also

Set Range

5.2.2 (d) - Set Range

Description

Sets the measurement 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:

Transmit Array

Byte	Data	Description
0	4	Interrupt code for Set Range
1	Range	Integer value corresponding to the measurement range: 1 - Expected input is 1-40 MHz 2 - Expected input is 40-190 MHz 3 - Expected input is 190-1400 MHz 4 - Expected input is 1400-6000 MHz 255 - Counter will automatically set range
2 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	4	Interrupt code for Set Range
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array would set the frequency counter to "Auto Range" mode:

Byte	Data	Description
0	4	Interrupt code for Set Range
1	255	Set counter to "Auto Range"
2 - 63	Not significant	"Don't care" bytes, can be any value

See Also

Get Frequency and Range
Set Sample Time

5.2.2 (e) - Set Sample Time

Description

Sets the sample time for frequency measurements. The allowed range is 0.1 to 3 seconds, in 0.1 second steps. The default sample time is 1 second.

Transmit Array

Byte	Data	Description
0	3	Interrupt code for Set Sample Time
1	ST_Multiple	The sample time to set, multiplied by 10
2 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	3	Interrupt code for Set Sample Time
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array would be sent to set the sample time to 0.4 seconds:

Byte	Data	Description
0	33	Interrupt code for Set Sample Time
1	4	ST_Multiple = 0.4 x 10 = 4
2 - 63	Not significant	"Don't care" bytes, can be any value

See Also

Set Range

5.2.2 (f) - Get Sample Time

Description

Returns the sample time for frequency measurements, in seconds.

Transmit Array

Byte	Data	Description
0	33	Interrupt code for Get Sample Time
1 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	33	Interrupt code for Get Sample Time
1	ST_Multiple	The sample time multiplied by 10
2 - 63	Not significant	"Don't care" bytes, could be any value

Example

The below returned array indicates a sample time of 0.4 seconds has been set:

Byte	Data	Description
0	33	Interrupt code for Get Sample Time
1	4	Sample time = 4 / 10 = 0.4 seconds
2 - 63	Not significant	"Don't care" bytes, could be any value

See Also

Set Sample Time

5.2.2 (g) - Get Firmware

Description

Returns the internal firmware version of the frequency counter.

Transmit Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	Reserved	Internal code for factory use only
2	Reserved	Internal code for factory use only
3	Reserved	Internal code for factory use only
4	Reserved	Internal code for factory use only
5	Firmware Letter	ASCII code for the first character in the firmware revision identifier
6	Firmware Number	ASCII code for the second character in the firmware revision identifier
7-63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the frequency counter has firmware version C3:

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	55	Internal code for factory use only
2	52	Internal code for factory use only
3	83	Internal code for factory use only
4	87	Internal code for factory use only
5	67	ASCII code for the letter "C"
6	51	ASCII code for the number 3
7-63	Not significant	"Don't care" bytes, could be any value