

Test Solutions - Programming Manual

Integrated Frequency Counter & Power Meter



FCPM Series Integrated Frequency & Power Meters

Mini-Circuits®

PO Box 350166, Brooklyn, NY 11235-0003
+1 718-934-4500 | testsolutions@minicircuits.com
www.minicircuits.com

Important Notice

This guide is owned by Mini-Circuits and is protected by copyright, trademark and other intellectual property laws.

The information in this guide is provided by Mini-Circuits as an accommodation to our customers and may be used only to promote and accompany the purchase of Mini-Circuits' Parts. This guide may not be reproduced, modified, distributed, published, stored in an electronic database, or transmitted and the information contained herein may not be exploited in any form or by any means, electronic, mechanical recording or otherwise, without prior written permission from Mini-Circuits.

This guide is subject to change, qualifications, variations, adjustments or modifications without notice and may contain errors, omissions, inaccuracies, mistakes or deficiencies. Mini-Circuits assumes no responsibility for, and will have no liability on account of, any of the foregoing. Accordingly, this guide should be used as a guideline only.

Trademarks

Microsoft, Windows, Visual Basic, Visual C# and Visual C++ are registered trademarks of Microsoft Corporation. LabVIEW and CVI are registered trademarks of National Instruments Corporation. Delphi is a registered trademark of Delphi Technologies, Inc. MATLAB is a registered trademark of The MathWorks, Inc. Agilent VEE is a registered trademark of Agilent Technologies, Inc. Linux is a registered trademark of Linus Torvalds. Mac is a registered trademark of Apple Inc. Python is a registered trademark of Python Software Foundation Corporation.

All other trademarks cited within this guide are the property of their respective owners. Neither Mini-Circuits nor the Mini-Circuits PTE (portable test equipment) series are affiliated with or endorsed or sponsored by the owners of the above referenced trademarks.

Mini-Circuits and the Mini-Circuits logo are registered trademarks of Scientific Components Corporation.

Mini-Circuits

13 Neptune Avenue

Brooklyn, NY 11235, USA

Phone: +1-718-934-4500

Email: sales@minicircuits.com

Web: www.minicircuits.com

1 - Overview	7
2 - Operating in a Windows Environment via USB	8
2.1 - The DLL (Dynamic Link Library) Concept	8
2.1 (a) - ActiveX COM Object	9
2.1 (b) - Microsoft.NET Class Library	11
2.2 - Referencing the DLL Library	12
2.3 - Summary of DLL Properties/Functions	13
1.1 (a) - DLL - General Functions.....	13
2.3 (a) - DLL - Power Measurement Properties & Functions	13
2.3 (b) - DLL - Frequency Measurement Functions	14
2.3 (c) - DLL - Ethernet Configuration Functions	14
2.4 - DLL - General Functions	15
2.4 (a) - Open USB Connection	15
2.4 (b) - Close USB Connection	16
2.4 (c) - Read Model Name	17
2.4 (d) - Read Serial Number	18
2.4 (e) - Get List of Connected Serial Numbers	19
2.4 (f) - Get Status.....	20
1.1 (b) - Check Connection.....	21
2.4 (g) - Get Internal Temperature	22
2.4 (h) - Get Firmware	23
2.4 (i) - Get Firmware Version (Antiquated)	24
2.4 (j) - Get USB Device Name	25
2.4 (k) - Get USB Device Handle.....	26
2.4 (l) - Open Any Sensor (Antiquated).....	27
2.4 (m) - Initialize Any Sensor (Antiquated).....	28
2.4 (n) - Close Sensor Connection (Antiquated)	29
2.5 - DLL - Power Measurement Properties & Functions	30
2.5 (a) - Set Compensation Frequency	30
2.5 (b) - Set Averaging Mode	31
2.5 (c) - Set Average Count	32
2.5 (d) - Set Power Format.....	33
2.5 (e) - Set Offset Value.....	34
2.5 (f) - Enable Offset.....	35
2.5 (g) - Set Measurement Mode	36
2.5 (h) - Set Power Range	37
2.5 (i) - Read Power	38
2.5 (j) - Read Immediate Power	39
2.5 (k) - Read Voltage	40
2.5 (l) - Get Offset Values.....	41
2.5 (m) - Set Offset Values	43
2.5 (n) - Set Compensation Frequency Mode.....	45
2.5 (o) - Get Compensation Frequency Mode	46
2.6 - DLL - Frequency Measurement Functions	47
2.6 (a) - Set Range.....	47
2.6 (b) - Get Range.....	48
2.6 (c) - Get Requested Range	49
2.6 (d) - Set Sample Time.....	50
2.6 (e) - Get Sample Time	51

2.6 (f) - Read Frequency.....	52
2.6 (g) - Get Reference Source.....	53
2.7 - DLL - Ethernet Configuration Functions	54
2.7 (a) - Get Ethernet Configuration.....	54
2.7 (b) - Get IP Address.....	56
2.7 (c) - Get MAC Address.....	58
2.7 (d) - Get Network Gateway.....	60
2.7 (e) - Get Subnet Mask.....	62
2.7 (f) - Get TCP/IP Port	64
2.7 (g) - Get DHCP Status	65
2.7 (h) - Get Password Status	66
2.7 (i) - Get Password.....	67
2.7 (j) - Save IP Address	68
2.7 (k) - Save Network Gateway	69
2.7 (l) - Save Subnet Mask	70
2.7 (m) - Save TCP/IP Port.....	71
2.7 (n) - Use DHCP.....	72
2.7 (o) - Use Password	73
2.7 (p) - Set Password	74
3 - Operating in a Linux Environment via USB.....	75
3.1 - Interrupts - General Functions	75
3.1 (a) - Get Device Model Name	76
3.1 (b) - Get Device Serial Number	77
3.1 (c) - Get Internal Temperature	78
3.1 (d) - Get Firmware	79
3.2 - Interrupts - Power Measurement Functions.....	80
3.2 (a) - Set Measurement Mode	80
3.2 (b) - Set Compensation Frequency Mode.....	81
3.2 (c) - Get Compensation Frequency Mode	82
3.2 (d) - Read Power	83
3.3 - Interrupts - Frequency Measurement Functions.....	85
3.3 (a) - Set Range.....	86
3.3 (b) - Get Range.....	87
3.3 (c) - Get Requested Range	88
3.3 (d) - Set Sample Time.....	89
3.3 (e) - Get Sample Time	90
3.3 (f) - Get Reference Source	91
3.3 (g) - Read Frequency.....	92
3.4 - Interrupts - Ethernet Configuration Functions	93
3.4 (a) - Set Static IP Address.....	94
3.4 (b) - Set Static Subnet Mask.....	95
3.4 (c) - Set Static Network Gateway.....	96
3.4 (d) - Set HTTP Port	97
3.4 (e) - Set Telnet Port.....	98
3.4 (f) - Use Password	99
3.4 (g) - Set Password	100
3.4 (h) - Use DHCP.....	101
3.4 (i) - Get Static IP Address	102
3.4 (j) - Get Static Subnet Mask	103
3.4 (k) - Get Static Network Gateway	104

3.4 (l) - Get HTTP Port	105
3.4 (m) - Get Telnet Port	106
3.4 (n) - Get Password Status	107
3.4 (o) - Get Password	108
3.4 (p) - Get DHCP Status	109
3.4 (q) - Get Dynamic Ethernet Configuration	110
3.4 (r) - Get MAC Address	112
3.4 (s) - Reset Ethernet Configuration	113
4 - Ethernet Control over IP Networks	114
4.1 (a) - Configuring Ethernet Settings via USB	114
4.2 - Ethernet Communication Methodology	115
4.2 (a) - Setting Sensor Properties Using HTTP and SCPI	115
4.2 (b) - Querying Power Sensor Properties Using HTTP and SCPI	116
4.2 (c) - Communication Using Telnet and SCPI	117
4.3 - Device Discovery Using UDP	118
5 - SCPI Commands for Ethernet Control	120
5.1 - Summary of SCPI Commands / Queries	120
5.1 (a) - SCPI - General Commands	120
5.1 (b) - SCPI - Power Measurement Commands	121
5.1 (c) - SCPI - Frequency Measurement Commands	121
5.2 - SCPI - General Commands	122
5.2 (a) - Get Model Name	122
5.2 (b) - Get Serial Number	123
5.2 (c) - Get Firmware	124
5.2 (d) - Get Temperature Units	125
5.2 (e) - Set Temperature Units	126
5.2 (f) - Get Internal Temperature	127
5.2 (g) - Get LCD Format	128
5.2 (h) - Set LCD Format	129
5.3 - SCPI - Power Measurement Commands	130
5.3 (a) - Get Measurement Mode	130
5.3 (b) - Set Measurement Mode	131
5.3 (c) - Get Averaging Mode	132
5.3 (d) - Set Averaging Mode	133
5.3 (e) - Get Average Count	134
5.3 (f) - Set Average Count	135
5.3 (g) - Get Compensation Frequency	136
5.3 (h) - Set Compensation Frequency	137
5.3 (i) - Get Compensation Frequency Mode	138
5.3 (j) - Set Compensation Frequency Mode	139
5.3 (k) - Read Power	140
5.3 (l) - Read Voltage	141
5.3 (m) - Get LCD Offset Mode	142
5.3 (n) - Set LCD Offset Mode	143
5.3 (o) - Get LCD Offset Value	144
5.3 (p) - Set LCD Offset Value	145
5.4 - SCPI - Frequency Measurement Commands	146
5.4 (a) - Get Range	146
5.4 (b) - Get Requested Range	147

- 5.4 (c) - Set Range 148
- 5.4 (d) - Get Sample Time 149
- 5.4 (e) - Set Sample Time 150
- 5.4 (f) - Get Frequency 151
- 5.4 (g) - Get Frequency & Power 152
- 5.4 (h) - Get Reference Mode 153

1 - Overview

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' USB and Ethernet controlled, integrated frequency and power meters. For instructions on using the supplied GUI program, or connecting the PTE hardware, please see the User Guide at:

<https://www.minicircuits.com/app/AN49-010.pdf>

Mini-Circuits offers support over a variety of operating systems, programming environments and third party applications.

Support for Windows® operating systems is provided through the Microsoft®.NET® and ActiveX® frameworks to allow the user to develop customized control applications. Support for Linux® operating systems is accomplished using the standard libhid and libusb libraries.

Mini-Circuits has experience with a wide variety of environments including (but not limited to):

- Visual Basic®, Visual C#®, Visual C++®
- Delphi®
- Borland C++®
- CVI®
- LabVIEW®
- MATLAB®
- Python®
- Agilent VEE®

The integrated frequency and power meter software package includes a GUI program, ActiveX and .NET DLL files, Linux support, project examples for third party software, and detailed user manuals. The latest package is available for download at:

https://www.minicircuits.com/softwaredownload/software_download.html

For details on individual models, application notes, GUI installation instructions and user guides please see:

<https://www.minicircuits.com/WebStore/PortableTestEquipment.html>

Files made available for download from the Mini-Circuits website are subject to Mini-Circuits' terms of use which are available on the website.

2 - Operating in a Windows Environment via USB

When connected by USB, the computer will recognize the FCPM as a Human Interface Device (HID). In this mode of operation the DLL file provides the method of control. Alternatively, the sensor can be operated over an Ethernet TCP/IP Network (see [Ethernet Control over IP Networks](#) for details).

2.1 - The DLL (Dynamic Link Library) Concept

The Dynamic Link Library concept is Microsoft's implementation of the shared library concept in the Windows environment.

DLLs provide a mechanism for shared code and data, intended to allow a developer to distribute applications without requiring code to be re-linked or recompiled.

Mini-Circuits' CD package provides DLL Objects designed to allow your own software application to interface with the functions of the Mini-Circuits integrated frequency and power meter.

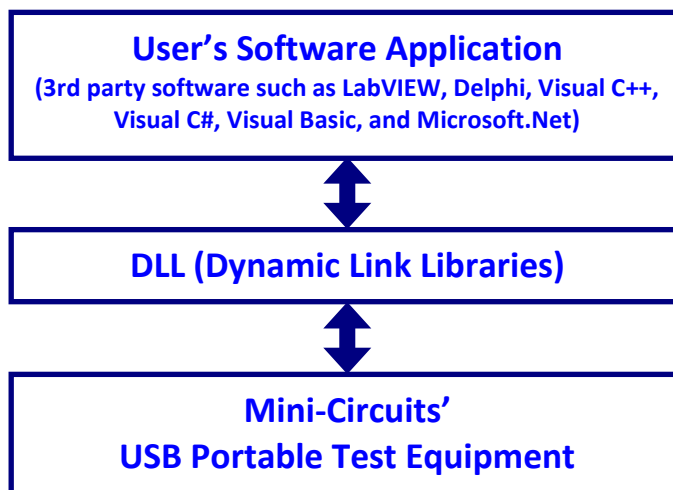


Fig 2.1-a: DLL Interface Concept

The software package provides two DLL files, the choice of which file to use is dictated by the user's operating system:

1. ActiveX com object

Designed to be used in any programming environment that supports third party ActiveX COM (Component Object Model) compliant applications.

The ActiveX file should be registered using RegSvr32 (see following sections for details).

2. Microsoft.NET Class Library

A logical unit of functionality that runs under the control of the Microsoft.NET system.

2.1 (a) - ActiveX COM Object

ActiveX COM object DLL files are designed to be used with both 32-bit and 64-bit Windows operating systems. A 32-bit programming environment that is compatible with ActiveX is required. To develop 64-bit applications, the Microsoft.NET Class library should be used instead.

Supported Programming Environments

Mini-Circuits' integrated frequency and power meters have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support ActiveX functionality. Please contact Mini-Circuits for support.

- Visual Studio® 6 (Visual C++ and Visual Basic)
- LabVIEW 8.0 or newer
- MATLAB 7 or newer
- Delphi
- Borland C++
- Agilent VEE
- Python

Installation

1. Copy the DLL file to the correct directory:
For 32-bit Windows operating systems this is C:\WINDOWS\System32
For 64-bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. Open the Command Prompt:
 - a. For Windows XP® (see Fig 2.1-b):
 - i. Select "All Programs" and then "Accessories" from the Start Menu
 - ii. Click on "Command Prompt" to open
 - b. For later versions of the Windows operating system you will need to have Administrator privileges in order to run the Command Prompt in "Elevated" mode (see Fig 2.1-c for Windows 7 and Windows 8):
 - i. Open the Start Menu/Start Screen and type "Command Prompt"
 - ii. Right-click on the shortcut for the Command Prompt
 - iii. Select "Run as Administrator"
 - iv. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3. Use regsvr32 to register the DLL:
For 32-bit Windows operating systems type (see Fig 2.1-d):
`\WINDOWS\System32\Regsvr32 \WINDOWS\System32\mc1_pm.dll`
For 64-bit Windows operating systems type (see Fig 2.1-e):
`\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mc1_pm.dll`
4. Hit enter to confirm and a message box will appear to advise of successful registration.

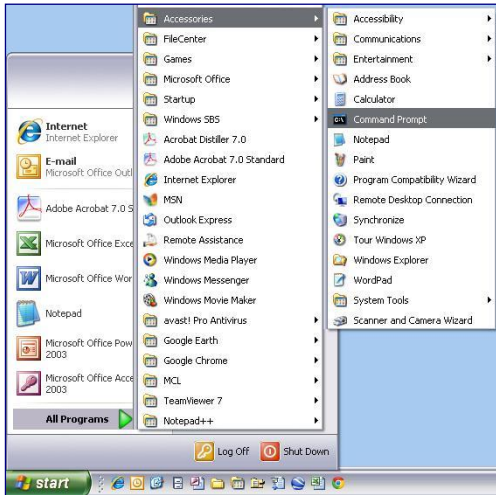


Fig 2.1-b: Opening the Command Prompt in Windows XP

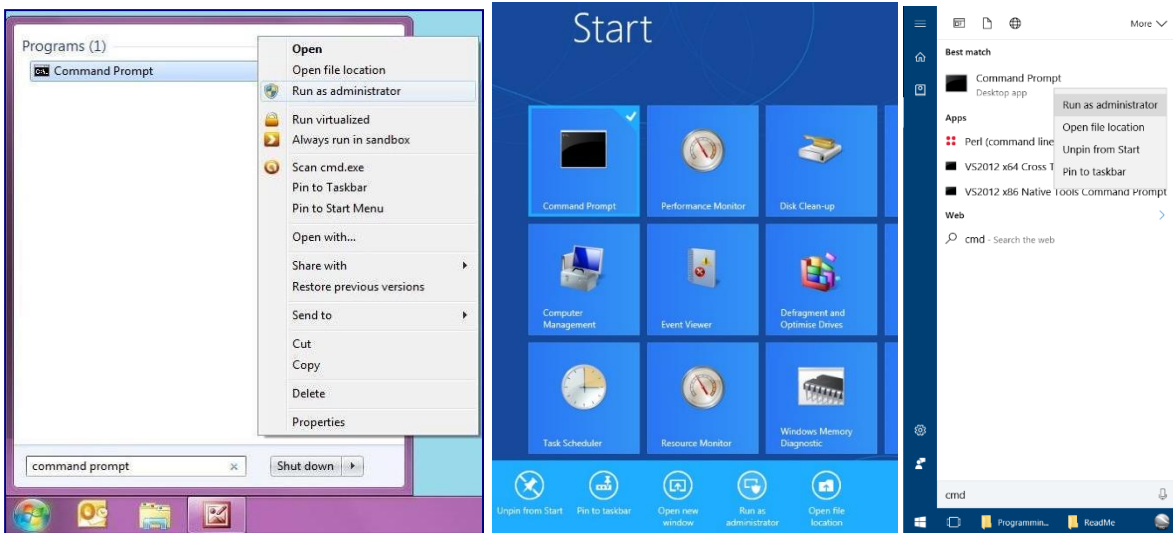


Fig 2.1-c: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

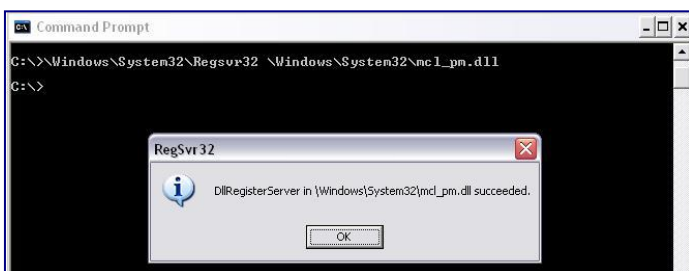


Fig 2.1-d: Registering the DLL in a 32-bit environment

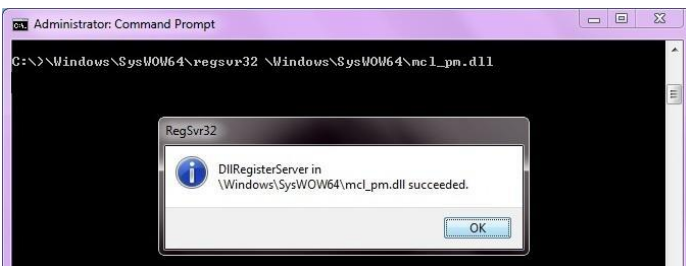


Fig 2.1-e: Registering the DLL in a 64-bit environment

2.1 (b) - Microsoft.NET Class Library

Microsoft.NET class libraries are designed to be used with both 32-bit and 64-bit Windows operating systems. To develop 64-bit applications the user must have both a 64-bit operating system and 64-bit programming environment. However, the Microsoft.NET class library is also compatible with 32-bit programming environments.

Supported Programming Environments

Mini-Circuits' integrated frequency and power meters have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support Microsoft.NET functionality. Please contact Mini-Circuits for support.

- National Instruments CVI
- Microsoft.NET (Visual C++, Visual Basic.NET, Visual C# 2003 or newer)
- LabVIEW 2009 or newer
- MATLAB 2008 or newer
- Delphi
- Borland C++

Installation

1. Copy the DLL file to the correct directory
 - a. For 32 bit Windows operating systems this is [C:\WINDOWS\System32](#)
 - b. For 64 bit Windows operating systems this is [C:\WINDOWS\SysWOW64](#)
2. **No registration is required**

2.2 - Referencing the DLL Library

The DLL file should be installed in the host PC's system folders using the steps outlined above. 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, a new instance of the USB sensor class just needs to be created for each physical sensor to control. The details of this vary greatly between programming environments and languages but Mini-Circuits can provide detailed support on request. The names "MyPTE1" and "MyPTE2" have been assigned to 2 connected sensors in the examples below.

Example Declarations using the ActiveX DLL

Visual Basic

```
Public MyPTE1 As New mcl_pm.USB_PM
    ' Initialize new sensor object, assign to MyPTE1
Public MyPTE2 As New mcl_pm.USB_PM
    ' Initialize new sensor object, assign to MyPTE2
```

Visual C++

```
mcl_pm::USB_PM ^MyPTE1 = gcnew mcl_pm::USB_PM();
    // Initialize new sensor instance, assign to MyPTE1
mcl_pm::USB_PM ^MyPTE2 = gcnew mcl_pm::USB_PM();
    // Initialize new sensor instance, assign to MyPTE2
```

Visual C#

```
mcl_pm.USB_PM MyPTE1 = new mcl_pm.USB_PM();
    // Initialize new sensor instance, assign to MyPTE1
mcl_pm.USB_PM MyPTE2 = new mcl_pm.USB_PM();
    // Initialize new sensor instance, assign to MyPTE2
```

Matlab

```
MyPTE1 = actxserver('mcl_pm.USB_PM')
    % Initialize new sensor instance, assign to MyPTE1
MyPTE2 = actxserver('mcl_pm.USB_PM')
    % Initialize new sensor instance, assign to MyPTE2
```

Example Declarations using the .NET DLL

Visual Basic

```
Public MyPTE1 As New mcl_pm64.usb_pm
    ' Initialize new sensor object, assign to MyPTE1
Public MyPTE2 As New mcl_pm64.usb_pm
    ' Initialize new sensor object, assign to MyPTE2
```

Visual C++

```
mcl_pm64::usb_pm ^MyPTE1 = gcnew mcl_pm64::usb_pm();
    // Initialize new sensor instance, assign to MyPTE1
mcl_pm64::usb_pm ^MyPTE2 = gcnew mcl_pm64::usb_pm();
    // Initialize new sensor instance, assign to MyPTE2
```

Visual C#

```
mcl_pm64.usb_pm MyPTE1 = new mcl_pm64.usb_pm();
    // Initialize new sensor instance, assign to MyPTE1
mcl_pm64.usb_pm MyPTE2 = new mcl_pm64.usb_pm();
    // Initialize new sensor instance, assign to MyPTE2
```

Matlab

```
MCL_PM = NET.addAssembly('C:\Windows\SysWOW64\mcl_pm64.dll')
MyPTE1 = mcl_pm64.usb_pm      % Initialize new sensor instance
MyPTE2 = mcl_pm64.usb_pm      % Initialize new sensor instance
```

2.3 - Summary of DLL Properties/Functions

The following functions and “global” properties are defined in both of the DLL files to allow full control over the sensor. Please see the following sections for a description of their usage.

1.1 (a) - DLL - General Functions

- a) int `Open_Sensor`(Optional string `SN_Request`) (ActiveX)
 short `Open_Sensor`(Optional string `SN_Request`) (.NET)
- b) void `Close_Sensor`()
- c) string `GetSensorModelName`()
- d) string `GetSensorSN`()
- e) short `Get_Available_SN_List`(ByRef string `SN_List`)
- f) short `GetStatus`()
- g) short `Check_Connection`()
- h) float `GetDeviceTemperature`(Optional string
 TemperatureFormat) (ActiveX)
 float `GetDeviceTemperature`(Optional ByRef string
 TemperatureFormat) (.NET)
- i) short `GetFirmwareInfo`(ByRef short `FirmwareID`,
 ByRef string `FirmwareRev`, ByRef short `FirmwareNo`)
- j) short `GetFirmwareVer`(ByRef short `FirmwareVer`)
- k) string `GetUSBDeviceName`()
- l) string `GetUSBDeviceHandle`()
- m) short `Open_AnySensor`()
- n) void `Init_PM`()
- o) void `CloseConnection`()

2.3 (a) - DLL - Power Measurement Properties & Functions

- a) double `Freq`
- b) short `AVG`
- c) short `AvgCount`
- d) bool `Format_mw`
- e) single `OffsetValue`
- f) short `OffsetValue_Enable`
- g) void `SetFasterMode`(short `S_A`) (ActiveX)
 void `SetFasterMode`(ByRef short `S_A`) (.NET)
- h) void `SetRange`(short `Range`)
- i) float `ReadPower`()
- j) float `ReadImmediatePower`()
- k) float `ReadVoltage`()
- l) short `GetOffsetValues`(ByRef int `NoOfPoints`,
 ByRef double `FreqArray`(), ByRef single `LossArray`())
- m) int `SetOffsetValues`(int `NoOfPoints`, double `FreqArray`(),
 single `LossArray`()) (ActiveX)
 int `SetOffsetValues`(int `NoOfPoints`, ByRef double `FreqArray`(),
 ByRef single `LossArray`) (.NET)
- n) short `FC_SetAutoFreq`(short `AutoFreq`)
- o) short `FC_GetAutoFreq`()

2.3 (b) - DLL - Frequency Measurement Functions

- a) short `FC_SetRange`(short `Range`)
- b) short `FC_GetRange`()
- c) short `FC_GetRequestedRange`()
- d) short `FC_SetSampleTime`(short `SampleTime`)
- e) int `FC_GetSampleTime`()
- f) Double `FC_ReadFreq`()
- g) short `FC_GetRef`()

2.3 (c) - DLL - Ethernet Configuration Functions

- a) int `GetEthernet_CurrentConfig`(ByRef int `IP1`, int `IP2`,
ByRef int `IP3`, ByRef int `IP4`, ByRef int `Mask1`,
ByRef int `Mask2`, ByRef int `Mask3`, ByRef int `Mask4`,
ByRef int `Gateway1`, ByRef int `Gateway2`,
ByRef int `Gateway3`, ByRef int `Gateway4`)
- b) int `GetEthernet_IPAddress`(ByRef int `b1`, ByRef int `b2`,
ByRef int `b3`, int `b4`)
- c) int `GetEthernet_MACAddress`(ByRef int `MAC1`, ByRef int `MAC2`,
ByRef int `MAC3`, ByRef int `MAC4`,
ByRef int `MAC5`, ByRef int `MAC6`)
- d) int `GetEthernet_NetworkGateway`(ByRef int `b1`, ByRef int `b2`,
ByRef int `b3`, ByRef int `b4`)
- e) int `GetEthernet_SubNetMask`(ByRef int `b1`, ByRef int `b2`,
ByRef int `b3`, ByRef int `b4`)
- f) int `GetEthernet_TCPIPPort`(ByRef int `port`)
- g) int `GetEthernet_UseDHCP`()
- h) int `GetEthernet_UsePWD`()
- i) int `GetEthernet_PWD`(ByRef string `Pwd`)
- j) int `SaveEthernet_IPAddress`(int `b1`, int `b2`, int `b3`, int `b4`)
- k) int `SaveEthernet_NetworkGateway`(int `b1`, int `b2`, int `b3`, int `b4`)
- l) int `SaveEthernet_SubnetMask`(int `b1`, int `b2`, int `b3`, int `b4`)
- m) int `SaveEthernet_TCPIPPort`(int `port`)
- n) int `SaveEthernet_UseDHCP`(int `UseDHCP`)
- o) int `SaveEthernet_UsePWD`(int `UsePwd`)
- p) int `SaveEthernet_PWD`(string `Pwd`)

2.4 - DLL - General Functions

2.4 (a) - Open USB Connection

ActiveX Declaration (mcl_pm.dll)

```
short Open_Sensor(Optional string SN)
```

.NET Declaration (mcl_pm64.dll)

```
short Open_Sensor(Optional ByRef string SN)
```

Description

This function is called to initialize the connection to a USB sensor head. 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 seconds 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 [Close_Sensor](#) function.

Parameters

Data Type	Variable	Description
string	SN	Optional. A string containing the serial number of the sensor. Can be omitted if only one sensor 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.Open_Sensor(SN_Request)
```

Visual C++

```
Status = MyPTE1->Open_Sensor(SN_Request);
```

Visual C#

```
Status = MyPTE1.Open_Sensor(SN_Request);
```

Matlab

```
Status = MyPTE1.Open_Sensor(SN_Request)
```

See Also

[Close USB Connection](#)

2.4 (b) - Close USB Connection

Declaration

```
void Close_Sensor ()
```

Description

This function is called to close the connection to the sensor head. 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 sensor from the computer, then reconnect before attempting to start again.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

Examples

```

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

See Also

[Open USB Connection](#)

2.4 (c) - Read Model Name

Declaration

```
string GetSensorModelName ()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
string	Model	Mini-Circuits model name of the connected sensor

Examples

Visual Basic

```
MsgBox ("The connected sensor is " & MyPTE1.GetSensorModelName)
```

Visual C++

```
MessageBox::Show ("The connected sensor is " + MyPTE1->GetSensorModelName());
```

Visual C#

```
MessageBox.Show ("The connected sensor is " + MyPTE1.GetSensorModelName());
```

Matlab

```
ModelName = MyPTE1.GetSensorModelName  
h = msgbox('The connected sensor is ', ModelName)
```

See Also

[Read Serial Number](#)

2.4 (d) - Read Serial Number

Declaration

```
string GetSensorSN()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
string	SN	Serial number of the connected sensor

Examples

```

Visual Basic
    MsgBox ("The connected sensor is " & MyPTE1.GetSensorSN)

Visual C++
    MessageBox::Show ("The connected sensor is " + MyPTE1->GetSensorSN());

Visual C#
    MessageBox.Show ("The connected sensor is " + MyPTE1.GetSensorSN());

Matlab
    SN = MyPTE1.GetSensorSN
    h = msgbox('The connected sensor is ', SN)
    
```

See Also

[Read Model Name](#)

2.4 (e) - Get List of Connected Serial Numbers

Declaration

```
short Get_Available_SN_List(ByRef 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) integrated frequency & power sensors.

Parameters

Data Type	Variable	Description
string	SN_List	Required. string variable which the function will update with a list of all available 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
    % split the List into array of SN's
end
```

See Also

[Open USB Connection](#)
[Read Serial Number](#)

2.4 (f) - Get Status

Declaration

```
short GetStatus ()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	No connection
short	1	USB connection to power sensor is active

Examples

Visual Basic

```
Status = MyPTE1.Get_Status
```

Visual C++

```
Status= MyPTE1->Get_Status ();
```

Visual C#

```
Status= MyPTE1.Get_Status ();
```

Matlab

```
Status= MyPTE1.Get_Status
```

See Also

[Read Power](#)

1.1 (b) - Check Connection

Declaration

```
short Check_Connection ()
```

Description

This function checks whether the USB connection to the power sensor is still active.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	No connection
short	1	USB connection to power sensor is active

Examples

```

Visual Basic
    Status = MyPTE1.Check_Connection
Visual C++
    Status= MyPTE1->Check_Connection();
Visual C#
    Status= MyPTE1.Check_Connection();
Matlab
    Status= MyPTE1.Check_Connection
    
```

See Also

[Read Power](#)

2.4 (g) - Get Internal Temperature

ActiveX Declaration (mcl_pm.dll)

```
float GetDeviceTemperature (Optional string TemperatureFormat)
```

.NET Declaration (mcl_pm64.dll)

```
float GetDeviceTemperature (Optional ByRef string TemperatureFormat)
```

Description

This function returns the internal temperature of the sensor in degrees Celsius (default) or Fahrenheit.

Parameters

Data Type	Variable	Description
string	Temperature _Format	Optional. string (not case sensitive) to set the temperature measurement units: F - Set temperature units to Fahrenheit C - Set temperature units to Celsius (default)

Return Values

Data Type	Value	Description
float	Temperature	The device internal temperature in degrees Celsius

Examples

Visual Basic

```
MsgBox ("Temperature is " & MyPTE1.GetDeviceTemperature)
```

Visual C++

```
MessageBox::Show ("Temperature is " + MyPTE1->GetDeviceTemperature());
```

Visual C#

```
MessageBox.Show ("Temperature is " + MyPTE1.GetDeviceTemperature());
```

Matlab

```
temp = MyPTE1.GetDeviceTemperature  
h = msgbox ("Temperature is, temp)
```

See Also

[Read Power](#)

[Read Immediate Power](#)

2.4 (h) - Get Firmware

Declaration

```
short GetFirmwareInfo(ByRef short FirmwareID,
                     ByRef string FirmwareRev, ByRef short FirmwareNo)
```

Description

This function returns the internal firmware version of the sensor.

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
short	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
        h = msgbox('Firmware version is ', fRev)
    end

```

2.4 (i) - Get Firmware Version (Antiquated)

Declaration

```
short GetFirmwareVer (ByRef short FirmwareVer)
```

Description

This function is antiquated, [GetFirmwareInfo](#) should be used instead. `GetFirmwareVer` returns a numeric value which indicates the internal firmware version of the sensor.

Parameters

Data Type	Variable	Description
short	FirmwareVer	Required. User defined variable which will be updated with the firmware version number

Return Values

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

Examples

Visual Basic

```
status = MyPTE1.GetFirmwareVer (FirmwareVer)
```

Visual C++

```
status = MyPTE1->GetFirmwareVer (FirmwareVer) ;
```

Visual C#

```
status = MyPTE1.GetFirmwareVer (FirmwareVer) ;
```

Matlab

```
status = MyPTE1.GetFirmwareVer (FirmwareVer)
```

See Also

[Get Firmware](#)

2.4 (j) - Get USB Device Name

Declaration

```
string GetUSBDeviceName ()
```

Description

This function is for advanced users to identify the USB device name of the sensor for direct communication.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
string	DeviceName	Device name of the sensor head

Examples

Visual Basic

```
UsbName = MyPTE1.GetUSBDeviceName
```

Visual C++

```
UsbName = MyPTE1->GetUSBDeviceName ();
```

Visual C#

```
UsbName = MyPTE1.GetUSBDeviceName ();
```

Matlab

```
UsbName = MyPTE1.GetUSBDeviceName
```

See Also

[Get USB Device Handle](#)

2.4 (k) - Get USB Device Handle

Declaration

```
string GetUSBDeviceHandle()
```

Description

This function is for advanced users to identify the handle to the USB sensor for direct communication.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
string	HandleToUSB	USB handle of the power sensor head

Examples

```

Visual Basic
    UsbHandle = MyPTE1.GetUSBDeviceHandle

Visual C++
    UsbHandle = MyPTE1->GetUSBDeviceHandle();

Visual C#
    UsbHandle = MyPTE1.GetUSBDeviceHandle();

Matlab
    UsbHandle = MyPTE1.GetUSBDeviceHandle
    
```

See Also

[Get USB Device Name](#)

2.4 (I) - Open Any Sensor (Antiquated)

Declaration

```
short Open_AnySensor ()
```

Description

This function is included for compatibility with early models, [Open_Sensor](#) is the recommended method to connect to a sensor head.

This function initializes the connection to a USB sensor head. If multiple sensors are connected to the same computer, it is not possible to determine which sensor will be initialized. 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 [Close_Sensor](#) function.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	No connection was possible
	1	Connection successfully established

See Also

[Open USB Connection](#)

2.4 (m) - Initialize Any Sensor (Antiquated)

Declaration

```
void Init_PM()
```

Description

This function is included for compatibility with early models, [Open_Sensor](#) is the recommended method to connect to a sensor head.

This function initializes the connection to a USB sensor head. If multiple sensors are connected to the same computer, it is not possible to determine which sensor will be initialized. 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 [Close_Sensor](#) function.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

See Also

[Open USB Connection](#)

2.4 (n) - Close Sensor Connection (Antiquated)

Declaration

```
void CloseConnection()
```

Description

This function is included for compatibility with early models, [Close_Sensor](#) is the recommended method to disconnect from a USB sensor head.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

See Also

[Close USB Connection](#)

2.5 - DLL - Power Measurement Properties & Functions

2.5 (a) - Set Compensation Frequency

Property

Double Freq

Description

This property sets the compensation frequency of the sensor head during operation in manual compensation mode; this needs to be set in order to achieve the specified power measurement accuracy. In automatic frequency compensation mode, the power reading is automatically compensated based on the sensor's simultaneous frequency measurement.

Note: This property will not filter out unwanted signals.

Accepted Values

Data Type	Value	Description
Double	Frequency	A frequency within the power sensor's specified range

Examples

```

Visual Basic
status = MyPTE1.FC_SetAutoFreq(0)
MyPTE1.Freq = 1000

Visual C++
status = MyPTE1->FC_SetAutoFreq(0);
MyPTE1->Freq = 1000;

Visual C#
status = MyPTE1.FC_SetAutoFreq(0);
MyPTE1.Freq = 1000;

Matlab
status = MyPTE1.FC_SetAutoFreq(0)
MyPTE1.Freq = 1000
    
```

See Also

[Set Compensation Frequency Mode](#)
[Get Compensation Frequency Mode](#)

2.5 (b) - Set Averaging Mode

Property

`short AVG`

Description

This property enables the “averaging” mode of the sensor so that power readings will be averaged over a number of measurements (defined by the [AvgCount](#) property). The default value is 0 (averaging disabled).

Accepted Values

Data Type	Value	Description
<code>short</code>	0	Disable averaging mode
<code>short</code>	1	Enable averaging mode

Examples

Visual Basic

```
MyPTE1.AVG = 1
```

Visual C++

```
MyPTE1->AVG = 1;
```

Visual C#

```
MyPTE1.AVG = 1;
```

Matlab

```
MyPTE1.AVG = 1
```

See Also

[Set Average Count](#)

2.5 (c) - Set Average Count

Property

`short AvgCount`

Description

This property defines the number of power readings over which to average the measurement when averaging mode is enabled (defined by the [AVG](#) property). The default value is 1 (average the reading over 1 measurement).

Accepted Values

Data Type	Value	Description
<code>short</code>	Count	The number of power measurements to average (1 to 16)

Examples

Visual Basic

```
MyPTE1.AvgCount = 10
```

Visual C++

```
MyPTE1->AvgCount = 10;
```

Visual C#

```
MyPTE1.AvgCount = 10;
```

Matlab

```
MyPTE1.AvgCount = 10
```

See Also

[Set Averaging Mode](#)

2.5 (d) - Set Power Format

Property

`bool Format_mw`

Description

This property sets the power measurement units to either mW or dBm. The default is dBm.

Accepted Values

Data Type	Value	Description
<code>bool</code>	False	Power reading in dBm
<code>bool</code>	True	Power reading in mW

Examples

Visual Basic

```
MyPTE1.Format_mw = TRUE
```

Visual C++

```
MyPTE1->Format_mw = TRUE;
```

Visual C#

```
MyPTE1.Format_mw = TRUE;
```

Matlab

```
MyPTE1.Format_mw = TRUE
```

See Also

[Read Power](#)

[Read Immediate Power](#)

2.5 (e) - Set Offset Value

Property

Double OffsetValue

Description

This property sets a single offset value to be used for power readings. The sensor's offset type must be set to "1" in order to use this (see [OffsetValue_Enable](#)).

Accepted Values

Data Type	Value	Description
Double	Offset	The power offset in either dBm or mW (as specified by Format_mw)

Examples

Visual Basic

```
MyPTE1.OffsetValue_enable = 1
MyPTE1.OffsetValue = 5.4
' Set a 5.4dB offset to the power readings
```

Visual C++

```
MyPTE1->OffsetValue_enable = 1;
MyPTE1->OffsetValue = 5.4;
// Set a 5.4dB offset to the power readings
```

Visual C#

```
MyPTE1.OffsetValue_enable = 1;
MyPTE1.OffsetValue = 5.4;
// Set a 5.4dB offset to the power readings
```

Matlab

```
MyPTE1.OffsetValue_enable = 1
MyPTE1.OffsetValue = 5.4
% Set a 5.4dB offset to the power readings
```

See Also

[Enable Offset](#)

2.5 (f) - Enable Offset

Property

`short OffsetValue_enable`

Description

This property defines whether an offset is to be used for power readings. The sensor can use either a single offset value (set using the [Set Offset Value](#) property) or an array of offset values (set by the [Set Offset Values](#) function).

Accepted Values

Data Type	Value	Description
short	0	Offset disabled
short	1	Use single value offset (see Set Offset Value)
short	2	Use array of offset values (see Set Offset Values)

Examples

Visual Basic

```
MyPTE1.OffsetValue_enable = 1
MyPTE1.OffsetValue = 5.4
' Set a 5.4dB offset to the power readings
```

Visual C++

```
MyPTE1->OffsetValue_enable = 1;
MyPTE1->OffsetValue = 5.4;
// Set a 5.4dB offset to the power readings
```

Visual C#

```
MyPTE1.OffsetValue_enable = 1;
MyPTE1.OffsetValue = 5.4;
// Set a 5.4dB offset to the power readings
```

Matlab

```
MyPTE1.OffsetValue_enable = 1
MyPTE1.OffsetValue = 5.4
% Set a 5.4dB offset to the power readings
```

See Also

[Set Offset Value](#)
[Get Offset Values](#)
[Set Offset Values](#)

2.5 (g) - Set Measurement Mode

ActiveX Declaration (mcl_pm.dll)

```
void SetFasterMode(short S_A)
```

.NET Declaration (mcl_pm64.dll)

```
void SetFasterMode(ByRef short S_A)
```

Description

This function sets the power measurement mode of the sensor between "low noise" and "fast sampling" modes. The specifications for these modes are defined in the individual model datasheets. The default is "low noise" mode.

Parameters

Data Type	Variable	Description
short	S_A	Reference to a user defined variable which determines the noise/sampling modes. The options are: 0 = Low noise mode 1 = Fast sampling mode

Return Values

Data Type	Value	Description
None		

Examples

Visual Basic <code>MyPTE1.SetFasterMode(S_A)</code>
Visual C++ <code>MyPTE1->SetFasterMode(S_A);</code>
Visual C# <code>MyPTE1.SetFasterMode(S_A);</code>
Matlab <code>MyPTE1.SetFasterMode(S_A)</code>

See Also

[Set Power Range](#)

2.5 (h) - Set Power Range

Declaration

```
void SetRange (short Range)
```

Description

This function optimizes the power measurement for the expected input power range. It is recommended that the sensor be left in the default "Auto" mode.

Parameters

Data Type	Variable	Description
short	Range	Reference to a user defined variable which determines the input power range. The options are: 0 = Auto 1 = Low power 2 = High power

Return Values

Data Type	Value	Description
None		

Examples

```

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

See Also

[Set Faster Mode](#)

2.5 (i) - Read Power

Declaration

```
float ReadPower ()
```

Description

This function returns the sensor power measurement. The default units are dBm but this can be set to mW using the [Format_mw](#) property.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
float	Power	Numerical value containing the current power measurement at the sensor head
	-99	The measured power level is too low (outside of the sensor's input dynamic range)
	99	The measured power level is too high (outside of the sensor's input dynamic range)

Examples

Visual Basic

```
Pwr = MyPTE1.ReadPower
```

Visual C++

```
Pwr = MyPTE1->ReadPower ();
```

Visual C#

```
Pwr = MyPTE1.ReadPower ();
```

Matlab

```
Pwr = MyPTE1.ReadPower
```

See Also

[Set Power Format](#)

[Read Immediate Power](#)

2.5 (j) - Read Immediate Power

Declaration

```
float ReadImmediatePower ()
```

Description

This function returns the power measurement with a faster response but reduced accuracy compared to [ReadPower](#). This function does not measure the temperature in the same process so temperature compensation is based on the last recorded reading (taken when the [ReadPower](#) or [GetDeviceTemperature](#) functions were last called). For greatest accuracy, [ReadPower](#) should be used. The default units are dBm but this can be set to mW using the [Format_mw](#) property.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
float	Power	Uncompensated power measurement

Examples

```

Visual Basic
Pwr = MyPTE1.ReadImmediatePower

Visual C++
Pwr = MyPTE1->ReadImmediatePower ();

Visual C#
Pwr = MyPTE1.ReadImmediatePower ();

Matlab
Pwr = MyPTE1.ReadImmediatePower
    
```

See Also

[Set Power Format](#)
[Read Immediate Power](#)

2.5 (k) - Read Voltage

Declaration

```
float ReadVoltage ()
```

Description

This function returns the raw voltage detected at the sensor head. There is no calibration for temperature or frequency.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
float	Voltage	Voltage detected at the sensor head

Examples

Visual Basic

```
Voltage = MyPTE1.ReadVoltage
```

Visual C++

```
Voltage = MyPTE1->ReadVoltage ();
```

Visual C#

```
Voltage = MyPTE1.ReadVoltage ();
```

Matlab

```
Voltage = MyPTE1.ReadVoltage
```

See Also

[Read Power](#)

[Read Immediate Power](#)

2.5 (I) - Get Offset Values

Declaration

```
short GetOffsetValues (ByRef int NoOfPoints, ByRef double FreqArray(),
                      ByRef single LossArray())
```

Description

This function returns the values used in the offset array when the sensor has been set to operate in “array offset” mode (see [Enable Offset](#)).

Parameters

Data Type	Variable	Description
int	NoOfPoints	Required. User defined variable which will be updated with the number of offset points specified in the array.
Double	FreqArray	Required. User defined array variable which will be updated with the list of frequency values (MHz) specified for the array offset.
float	LossArray	Required. User defined array variable which will be updated with the list of loss values (dB) specified for the array offset.

Return Values

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

Examples

Visual Basic

```
MyPTE1.GetOffsetValues(pts, freq, loss)
For i=0 To pts - 1
    MsgBox (i & ": " & freq(i) & "MHz, " & loss(i) & "dB")
Next
```

Visual C++

```
MyPTE1->GetOffsetValues(pts, freq, loss);
for (i = 0; i < pts; i++)
{
    MessageBox::Show(i + ": " + freq[i] + "MHz, " + loss[i] + "dB");
}
```

Visual C#

```
MyPTE1.GetOffsetValues(ref(pts, freq, loss));
for (i = 0; i < pts; i++)
{
    MessageBox.Show(i + ": " + freq[i] + "MHz, " + loss[i] + "dB");
}
```

Matlab

```
[status, pts, freq, loss]=MyPTE1.GetOffsetValues(pts, freq, loss)
maxi=pts-1
for i=0:maxi
    h = msgbox([i,': ',freq(i),'MHz ',loss(i),'dB'])
end
```

See Also

[Enable Offset](#)

[Set Offset Values](#)

2.5 (m) - Set Offset Values

ActiveX Declaration (mcl_pm.dll)

```
short SetOffsetValues(int NoOfPoints, double FreqArray(),
                     _single LossArray())
```

.NET Declaration (mcl_pm64.dll)

```
short SetOffsetValues(int NoOfPoints, ByRef double FreqArray(),
                     ByRef single LossArray())
```

Description

This function sets the array of offset values to be used for power measurements. The sensor must be set to operate in “array offset” mode (see [Enable Offset](#)).

Parameters

Data Type	Variable	Description
int	NoOfPoints	Required. The number of offset points to be defined in the array.
Double	FreqArray	Required. Array of size “NoOfPoints” containing the frequency (MHz) values of the respective offset points.
float	LossArray	Required. Array of size “NoOfPoints” containing the loss 9dB) values of the respective offset points.

Return Values

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

Examples

Visual Basic

```
Dim pts As Integer = 4
Dim freq(1000, 2000, 3000, 4000) As Double
Dim loss(0, 0.5, 1, 1.5) As float
MyPTE1.SetOffsetValues(pts, freq, loss)
' Set 4 offset values:
' 0dB @ 1000MHz; 0.5dB @ 2000MHz; 1dB @ 3000MHz; 1.5dB @ 4000MHz
```

Visual C++

```
int pts = 4;
double freq [pts] = {1000, 2000, 3000, 4000};
float loss [pts] = {0, 0.5, 1, 1.5};
MyPTE1->SetOffsetValues(pts, freq, loss);
// Set 4 offset values:
// 0dB @ 1000MHz; 0.5dB @ 2000MHz; 1dB @ 3000MHz; 1.5dB @ 4000MHz
```

Visual C#

```
int pts = 4;
double[] freq = {1000, 2000, 3000, 4000};
float[] loss = {0, 0.5, 1, 1.5};
MyPTE1->SetOffsetValues(pts, freq, loss);
// Set 4 offset values:
// 0dB @ 1000MHz; 0.5dB @ 2000MHz; 1dB @ 3000MHz; 1.5dB @ 4000MHz
```

Matlab

```
pts=4
freq=[1000,2000,3000,4000]
loss=[0,0.5,1,1.5]
[status]=MyPTE1.SetOffsetValues(pts, freq, loss)
% Set 4 offset values:
% 0dB @ 1000MHz; 0.5dB @ 2000MHz; 1dB @ 3000MHz; 1.5dB @ 4000MHz
```

See Also

[Enable Offset](#)

[Get Offset Values](#)

2.5 (n) - Set Compensation Frequency Mode

Declaration

```
short FC_SetAutoFreq(short AutoFreq)
```

Description

Sets whether the sensor is to use automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Parameters

Data Type	Variable	Description
short	AutoFreq	The frequency compensation mode to use: 0 = Manual compensation (the sensor's input frequency property must be set by the user) 1 = Automatic compensation (the sensor will automatically compensate power measurements based on the measured frequency)

Return Values

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

Examples

```

Visual Basic
    status = MyPTE1.FC_SetAutoFreq(1)
Visual C++
    status = MyPTE1->FC_SetAutoFreq(1);
Visual C#
    status = MyPTE1.FC_SetAutoFreq(1);
Matlab
    status = MyPTE1.FC_SetAutoFreq(1)
    
```

See Also

[Set Compensation Frequency](#)
[Get Compensation Frequency Mode](#)

2.5 (o) - Get Compensation Frequency Mode

Declaration

```
short FC_GetAutoFreq()
```

Description

Indicates whether the sensor is using automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	Manual compensation (the sensor's input frequency property must be set by the user)
short	1	Automatic compensation (the sensor will automatically compensate power measurements based on the measured frequency)

Examples

```

Visual Basic
mode = MyPTE1.FC_GetAutoFreq()
Visual C++
mode = MyPTE1->FC_GetAutoFreq();
Visual C#
mode = MyPTE1.FC_GetAutoFreq();
Matlab
mode = MyPTE1.FC_GetAutoFreq()

```

See Also

[Set Compensation Frequency](#)
[Set Compensation Frequency Mode](#)

2.6 - DLL - Frequency Measurement Functions

2.6 (a) - Set Range

Declaration

```
short FC_SetRange (short Range)
```

Description

Sets the frequency measurement range of the combined sensor. 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.

Parameters

Data Type	Value	Description
short	0	Set automatic measurement range mode
	1	Set range 1 (for input frequencies 1 to 40 MHz)
	2	Set range 2 (for input frequencies 40 to 190 MHz)
	3	Set range 3 (for input frequencies 190 to 1400 MHz)
	4	Set range 4 (for input frequencies 1400 to 6000 MHz)

Return Values

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

Examples

```

Visual Basic
    status = MyPTE1.FC_SetRange(0)
Visual C++
    status = MyPTE1->FC_SetRange(0);
Visual C#
    status = MyPTE1.FC_SetRange(0);
Matlab
    status = MyPTE1.FC_SetRange(0)
    
```

See Also

[Get Range](#)

[Get Requested Range](#)

2.6 (b) - Get Range

Declaration

```
short FC_GetRange ()
```

Description

Returns an ASCII character code indicating the frequency range of the sensor when operating in automatic range mode.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	49	ASCII code for range 1 (input frequency in range 1 to 40 MHz)
	50	ASCII code for range 2 (input frequency in range 40 to 190 MHz)
	51	ASCII code for range 3 (input frequency in range 190 to 1400 MHz)
	52	ASCII code for range 4 (input frequency in range 1400 to 6000 MHz). This is the default range if no input signal is detected.

Examples

Visual Basic

```
range = MyPTE1.FC_GetRange ()
```

Visual C++

```
range = MyPTE1->FC_GetRange ();
```

Visual C#

```
range = MyPTE1.FC_GetRange ();
```

Matlab

```
range = MyPTE1.FC_GetRange ()
```

See Also

[Set Range](#)

[Get Requested Range](#)

2.6 (c) - Get Requested Range

Declaration

```
short FC_GetRequestedRange ()
```

Description

Returns the measurement range of the sensor. By default the frequency counter is in “Auto Range” mode and will automatically adjust for the input frequency measurement, this process will take typically 50ms. The user can choose to override this if the frequency range of the input signal is known.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	Counter is in automatic range mode
	1	Range 1 (input frequency in range 1 to 40 MHz)
	2	Range 2 (input frequency in range 40 to 190 MHz)
	3	Range 3 (input frequency in range 190 to 1400 MHz)
	4	Range 4 (input frequency in range 1400 to 6000 MHz)

Examples

```

Visual Basic
    range = MyPTE1.FC_GetRequestedRange ()
Visual C++
    range = MyPTE1->FC_GetRequestedRange ();
Visual C#
    range = MyPTE1.FC_GetRequestedRange ();
Matlab
    range = MyPTE1.FC_GetRequestedRange ()
    
```

See Also

[Set Range](#)
[Get Range](#)

2.6 (d) - Set Sample Time

Declaration

```
short FC_SetSampleTime(short SampleTime)
```

Description

Sets the sample time to be used for frequency measurements, from 100 to 3000 ms, in 100 ms steps. The default sample time is 1000 ms (1 second).

Parameters

Data Type	Variable	Description
short	SampleTime	The sample time in ms

Return Values

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

Examples

Visual Basic

```
status = MyPTE1.FC_SetSampleTime(1000)
```

Visual C++

```
status = MyPTE1->FC_SetSampleTime(1000);
```

Visual C#

```
status = MyPTE1.FC_SetSampleTime(1000);
```

Matlab

```
status = MyPTE1.FC_SetSampleTime(1000)
```

See Also

[Get Sample Time](#)

2.6 (e) - Get Sample Time

Declaration

```
int FC_GetSampleTime ()
```

Description

Returns the time in milliseconds over which the input frequency will be sampled.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Variable	Description
int	STime	The sample time in milliseconds

Examples

```

Visual Basic
time = MyPTE1.FC_GetSampleTime ()

Visual C++
time = MyPTE1->FC_GetSampleTime ();

Visual C#
time = MyPTE1.FC_GetSampleTime ();

Matlab
time = MyPTE1.FC_GetSampleTime ()
    
```

See Also

[Set Sample Time](#)

2.6 (f) - Read Frequency

Declaration

```
Double FC_ReadFreq()
```

Description

Returns the frequency in MHz of the input signal.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Variable	Description
Double	Frequency	The measured frequency in MHz

Examples

```

Visual Basic
    freq = MyPTE1.FC_ReadFreq()
Visual C++
    freq = MyPTE1->FC_ReadFreq();
Visual C#
    freq = MyPTE1.FC_ReadFreq();
Matlab
    freq = MyPTE1.FC_ReadFreq()
    
```

See Also

[Read Power](#)

2.6 (g) - Get Reference Source

Declaration

```
short FC_GetRef ()
```

Description

Indicates whether the sensor is using the internal reference for frequency measurements or an external source. The reference source will automatically switch to external if a suitable signal is detected at the Ref In port.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Variable	Description
short	Source	The reference source currently in use for frequency measurements: 0 = Internal reference in use 1 = External reference in use

Examples

Visual Basic

```
source = MyPTE1.FC_GetRef ()
```

Visual C++

```
source = MyPTE1->FC_GetRef ();
```

Visual C#

```
source = MyPTE1.FC_GetRef ();
```

Matlab

```
source = MyPTE1.FC_GetRef ()
```

2.7 - DLL - Ethernet Configuration Functions

These functions provide a means for identifying or configuring the Ethernet settings such as IP address, TCP/IP port and network gateway. They can only be called while the device is connected via the USB interface.

2.7 (a) - Get Ethernet Configuration

Declaration

```
int GetEthernet_CurrentConfig (ByRef int IP1, ByRef int IP2,
                               ByRef int IP3, ByRef int IP4,
                               ByRef int Mask1, ByRef int Mask2,
                               ByRef int Mask3, ByRef int Mask4,
                               ByRef int Gateway1, ByRef int Gateway2,
                               ByRef int Gateway3, ByRef int Gateway4)
```

Description

This function returns the current IP configuration of the sensor in a series of user defined variables. The settings checked are IP address, subnet mask and network gateway.

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address.
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address.
int	IP3	Required. Integer variable which will be updated with the third octet of the IP address.
int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address.
int	Mask1	Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask.
int	Mask2	Required. Integer variable which will be updated with the second octet of the subnet mask.
int	Mask3	Required. Integer variable which will be updated with the third octet of the subnet mask.
int	Mask4	Required. Integer variable which will be updated with the last (lowest order) octet of the subnet mask.
int	Gateway1	Required. Integer variable which will be updated with the first (highest order) octet of the network gateway.
int	Gateway2	Required. Integer variable which will be updated with the second octet of the network gateway.
int	Gateway3	Required. Integer variable which will be updated with the third octet of the network gateway.
int	Gateway4	Required. Integer variable which will be updated with the last (lowest order) octet of the network gateway.

Return Values

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

Example

Visual Basic

```
If MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0 Then

    MsgBox ("IP address: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
    MsgBox ("Subnet Mask: " & M1 & "." & M2 & "." & M3 & "." & M4)
    MsgBox ("Gateway: " & GW1 & "." & GW2 & "." & GW3 & "." & GW4)

End If
```

Visual C++

```
if (MyPTE1->GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0)
{
    MessageBox::Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
        + IP4);
    MessageBox::Show("Subnet Mask: " + M1 + "." + M2 + "." + M3 + "." +
        M4);
    MessageBox::Show("Gateway: " + GW1 + "." + GW2 + "." + GW3 + "." +
        GW4);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0)
{
    MessageBox.Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
        + IP4);
    MessageBox.Show("Subnet Mask: " + M1 + "." + M2 + "." + M3 + "." +
        M4);
    MessageBox.Show("Gateway: " + GW1 + "." + GW2 + "." + GW3 + "." +
        GW4);
}
```

Matlab

```
[status, IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1, GW2, GW3, GW4] =
MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1,
GW2, GW3, GW4)
if status > 0
    h = msgbox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
    h = msgbox ("Subnet Mask: ", M1, "." & M2, "." & M3, ".", M4)
    h = msgbox ("Gateway: ", GW1, ".", GW2, ".", GW3, ".", GW4)
end
```

See Also

[Get MAC Address](#)

[Get TCP/IP Port](#)

2.7 (b) - Get IP Address

Declaration

```
int GetEthernet_IPAddress (ByRef int b1, ByRef int b2, ByRef int b3,
                          ByRef int b4)
```

Description

This function returns the current IP address of the sensor in a series of user defined variables (one per octet).

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example "192" for the IP address "192.168.1.0").
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example "168" for the IP address "192.168.1.0").
int	IP3	Required. Integer variable which will be updated with the third octet of the IP address (for example "1" for the IP address "192.168.1.0").
int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address (for example "0" for the IP address "192.168.1.0").

Return Values

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

Example

Visual Basic

```
If MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0 Then
    MsgBox ("IP address: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox::Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
                    + IP4);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox.Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
                    + IP4);
}
```

Matlab

```
[status, IP1, IP2, IP3, IP4] = MyPTE1.GetEthernet_CurrentConfig(IP1, IP2,
IP3, IP4)
if status > 0
    h = msgbox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
end
```

See Also[Get Ethernet Configuration](#)[Get TCP/IP Port](#)[Save IP Address](#)[Save TCP/IP Port](#)

2.7 (c) - Get MAC Address

Declaration

```
int GetEthernet_MACAddress (ByRef int MAC1, ByRef int MAC2,
    ByRef int MAC3, ByRef int MAC4, ByRef int MAC5, ByRef int MAC6)
```

Description

This function returns the MAC (media access control) address, the physical address, of the sensor as a series of decimal values (one for each of the 6 numeric groups).

Parameters

Data Type	Variable	Description
int	MAC1	Required. Integer variable which will be updated with the decimal value of the first numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC1=11
int	MAC2	Required. Integer variable which will be updated with the decimal value of the second numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC2=47
int	MAC3	Required. Integer variable which will be updated with the decimal value of the third numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC3=165
int	MAC4	Required. Integer variable which will be updated with the decimal value of the fourth numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC4=103
int	MAC5	Required. Integer variable which will be updated with the decimal value of the fifth numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC5=137
int	MAC6	Required. Integer variable which will be updated with the decimal value of the last numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC6=171

Return Values

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

Example

```

Visual Basic
If MyPTE1.GetEthernet_MACAddress (M1, M2, M3, M4, M5, M6) > 0 Then
    MsgBox ("MAC address: " & M1 & ":" & M2 & ":" & M3 & ":" & M4 & ":"
           & M5 & ":" & M6)
End If

Visual C++
if (MyPTE1->GetEthernet_MACAddress (M1, M2, M3, M4, M5, M6) > 0)
{
    MessageBox::Show("MAC address: " + M1 + "." + M2 + "." + M3 + "."
                    + M4 + "." + M5 + "." + M6);
}

Visual C#
if (MyPTE1.GetEthernet_MACAddress (M1, M2, M3, M4, M5, M6) > 0)
{
    MessageBox.Show("MAC address: " + M1 + "." + M2 + "." + M3 + "."
                   + M4 + "." + M5 + "." + M6);
}

Matlab
[status, M1, M2, M3, M4, M5, M6] = MyPTE1.GetEthernet_MACAddress (M1, M2, M3,
M4, M5, M6)
if status > 0
    h = msgbox("MAC address: ",M1,".",M2,".",M3,".",M4,".",M5,".",M6)
end
    
```

See Also

[Get Ethernet Configuration](#)

2.7 (d) - Get Network Gateway

Declaration

```
int GetEthernet_NetworkGateway (ByRef int b1, ByRef int b2,
                                ByRef int b3, ByRef int b4)
```

Description

This function returns the IP address of the network gateway to which the sensor is currently connected. A series of user defined variables are passed to the function to be updated with the IP address (one per octet).

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example "192" for the IP address "192.168.1.0").
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example "168" for the IP address "192.168.1.0").
int	IP3	Required. Integer variable which will be updated with the third octet of the IP address (for example "1" for the IP address "192.168.1.0").
int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address (for example "0" for the IP address "192.168.1.0").

Return Values

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

Example

Visual Basic

```
If MyPTE1.GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0 Then
    MsgBox ("Gateway: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox::Show("Gateway: " + IP1 + "." + IP2 + "." + IP3 + "."
                    + IP4);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox.Show("Gateway: " + IP1 + "." + IP2 + "." + IP3 + "."
                    + IP4);
}
```

Matlab

```
[status, IP1, IP2, IP3, IP4] = MyPTE1.GetEthernet_NetworkGateway(IP1, IP2,
IP3, IP4)
if status > 0
    h = msgbox ("Gateway: ", IP1, ".", IP2, ".", IP3, ".", IP4)
end
```

See Also

[Get Ethernet Configuration](#)

[Save Network Gateway](#)

2.7 (e) - Get Subnet Mask

Declaration

```
int GetEthernet_SubNetMask (ByRef int b1, ByRef int b2, ByRef int b3,
                            ByRef int b4)
```

Description

This function returns the subnet mask used by the network gateway to which the sensor is currently connected. A series of user defined variables are passed to the function to be updated with the subnet mask (one per octet).

Parameters

Data Type	Variable	Description
int	b1	Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	b2	Required. Integer variable which will be updated with the second octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	b3	Required. Integer variable which will be updated with the third octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	b4	Required. Integer variable which will be updated with the last (lowest order) octet of the subnet mask (for example "0" for the subnet mask "255.255.255.0").

Return Values

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

Example

Visual Basic

```
If MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4) > 0 Then
    MsgBox ("Subnet mask: " & b1 & "." & b2 & "." & b3 & "." & b4)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_SubNetMask(b1, b2, b3, b4) > 0)
{
    MessageBox::Show("Subnet mask: " + b1 + "." + b2 + "." + b3 + "."
                    + b4);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4) > 0)
{
    MessageBox.Show("Subnet mask: " + b1 + "." + b2 + "." + b3 + "."
                  + b4);
}
```

Matlab

```
[status, b1, b2, b3, b4] = MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4)
if status > 0
    h = msgbox ("Subnet mask: ", b1, ".", b2, ".", b3, ".", b4)
end
```

See Also

[Get Ethernet Configuration](#)

[Save Subnet Mask](#)

2.7 (f) - Get TCP/IP Port

Declaration

```
int GetEthernet_TCPIPPort (ByRef int port)
```

Description

This function returns the TCP/IP port used by the sensor for HTTP communication. The default is port 80.

Note: Port 23 is reserved for Telnet communication and cannot be set as the HTTP port.

Parameters

Data Type	Variable	Description
int	port	Required. Integer variable which will be updated with the TCP/IP port.

Return Values

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

Example

```

Visual Basic
    If MyPTE1.GetEthernet_SubNetMask(port) > 0 Then
        MsgBox ("Port: " & port)
    End If

Visual C++
    if (MyPTE1->GetEthernet_SubNetMask(port) > 0)
    {
        MessageBox::Show("Port: " + port);
    }

Visual C#
    if (MyPTE1.GetEthernet_SubNetMask(port) > 0)
    {
        MessageBox.Show("Port: " + port);
    }

Matlab
    [status, port] = MyPTE1.GetEthernet_SubNetMask(port)
    if status > 0
        h = msgbox ("Port: ", port)
    end

```

See Also

[Get Ethernet Configuration](#)
[Save TCP/IP Port](#)

2.7 (g) - Get DHCP Status

Declaration

```
short GetEthernet_UseDHCP ()
```

Description

This function indicates whether the sensor is using DHCP (dynamic host control protocol), in which case the IP configuration is derived from a network server; or user defined “static” IP settings.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	DHCP not in use (IP settings are static and manually configured)
short	1	DHCP in use (IP settings are assigned automatically by the network)

Example

Visual Basic

```
DHCPstatus = MyPTE1.GetEthernet_UseDHCP ()
```

Visual C++

```
DHCPstatus = MyPTE1->GetEthernet_UseDHCP ();
```

Visual C#

```
DHCPstatus = MyPTE1.GetEthernet_UseDHCP ();
```

Matlab

```
[DHCPstatus] = MyPTE1.GetEthernet_UseDHCP
```

See Also

[Get Ethernet Configuration](#)

[Use DHCP](#)

2.7 (h) - Get Password Status

Declaration

```
short GetEthernet_UsePWD ()
```

Description

This function indicates whether the sensor is currently configured to require a password for HTTP/Telnet communication.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	Password not required
short	1	Password required

Example

Visual Basic

```
PWDstatus = MyPTE1.GetEthernet_UsePWD ()
```

Visual C++

```
PWDstatus = MyPTE1->GetEthernet_UsePWD ();
```

Visual C#

```
PWDstatus = MyPTE1.GetEthernet_UsePWD ();
```

Matlab

```
[PWDstatus] = MyPTE1.GetEthernet_UsePWD
```

See Also

[Get Password](#)

[Use Password](#)

[Set Password](#)

2.7 (i) - Get Password

Declaration

```
int GetEthernet_PWD (ByRef string Pwd)
```

Description

This function returns the current password used by the sensor for HTTP/Telnet communication. The password will be returned even if the device is not currently configured to require a password.

Parameters

Data Type	Variable	Description
string	Pwd	Required. string variable which will be updated with the password.

Return Values

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

Example

```

Visual Basic
    If MyPTE1.GetEthernet_PWD(pwd) > 0 Then
        MsgBox ("Password: " & pwd)
    End If

Visual C++
    if (MyPTE1->GetEthernet_PWD(pwd) > 0)
    {
        MessageBox::Show("Password: " + pwd);
    }

Visual C#
    if (MyPTE1.GetEthernet_PWD(pwd) > 0)
    {
        MessageBox.Show("Password: " + pwd);
    }

Matlab
    [status, pwd] = MyPTE1.GetEthernet_PWD(pwd)
    if status > 0
        h = msgbox ("Password: ", pwd)
    end

```

See Also

- [Get Password Status](#)
- [Use Password](#)
- [Set Password](#)

2.7 (j) - Save IP Address

Declaration

```
short SaveEthernet_IPAddress(int b1, int b2, int b3, int b4)
```

Description

This function sets a static IP address to be used by the connected sensor.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the IP address to set (for example "192" for the IP address "192.168.1.0").
int	IP2	Required. Second octet of the IP address to set (for example "168" for the IP address "192.168.1.0").
int	IP3	Required. Third octet of the IP address to set (for example "1" for the IP address "192.168.1.0").
int	IP4	Required. Last (lowest order) octet of the IP address to set (for example "0" for the IP address "192.168.1.0").

Return Values

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

Example

Visual Basic

```
status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)
```

Visual C++

```
status = MyPTE1->SaveEthernet_IPAddress(192, 168, 1, 0);
```

Visual C#

```
status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0);
```

Matlab

```
[status] = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)
```

See Also

[Get Ethernet Configuration](#)

[Get IP Address](#)

2.7 (k) - Save Network Gateway

Declaration

```
short SaveEthernet_NetworkGateway(int b1, int b2, int b3, int b4)
```

Description

This function sets the IP address of the network gateway to which the sensor should connect.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the network gateway IP address (for example "192" for the IP address "192.168.1.0").
int	IP2	Required. Second octet of the network gateway IP address (for example "168" for the IP address "192.168.1.0").
int	IP3	Required. Third octet of the network gateway IP address (for example "1" for the IP address "192.168.1.0").
int	IP4	Required. Last (lowest order) octet of the network gateway IP address (for example "0" for the IP address "192.168.1.0").

Return Values

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

Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0)
Visual C++
    status = MyPTE1->SaveEthernet_NetworkGateway(192, 168, 1, 0);
Visual C#
    status = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0);
Matlab
    [status] = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0)
    
```

See Also

[Get Ethernet Configuration](#)
[Get Network Gateway](#)

2.7 (I) - Save Subnet Mask

Declaration

```
short SaveEthernet_SubnetMask(int b1, int b2, int b3, int b4)
```

Description

This function sets the subnet mask of the network to which the sensor should connect.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	IP2	Required. Second octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	IP3	Required. Third octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
int	IP4	Required. Last (lowest order) octet of the subnet mask (for example "0" for the subnet mask "255.255.255.0").

Return Values

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

Example

Visual Basic

```
status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)
```

Visual C++

```
status = MyPTE1->SaveEthernet_SubnetMask(255, 255, 255, 0);
```

Visual C#

```
status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0);
```

Matlab

```
[status] = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)
```

See Also

[Get Ethernet Configuration](#)
[Get Subnet Mask](#)

2.7 (m) - Save TCP/IP Port

Declaration

```
short SaveEthernet_TCPIPPort(int port)
```

Description

This function sets the TCP/IP port used by the sensor for HTTP communication. The default is port 80.

Note: Port 23 is reserved for Telnet communication and cannot be set as the HTTP port.

Parameters

Data Type	Variable	Description
int	port	Required. Numeric value of the TCP/IP port.

Return Values

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

Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_TCPIPPort(70)
Visual C++
    status = MyPTE1->SaveEthernet_TCPIPPort(70);
Visual C#
    status = MyPTE1.SaveEthernet_TCPIPPort(70);
Matlab
    [status] = MyPTE1.SaveEthernet_TCPIPPort(70)
    
```

See Also

[Get TCP/IP Port](#)

2.7 (n) - Use DHCP

Declaration

```
short SaveEthernet_UseDHCP (int UseDHCP)
```

Description

This function enables or disables DHCP (dynamic host control protocol). When enabled the IP configuration of the sensor is assigned automatically by the network server; when disabled the user defined "static" IP settings apply.

Parameters

Data Type	Variable	Description
int	UseDHCP	Required. Integer value to set the DHCP mode: 0 - DHCP disabled (static IP settings used) 1 - DHCP enabled (IP setting assigned by network)

Return Values

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

Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_UseDHCP (1)
Visual C++
    status = MyPTE1->SaveEthernet_UseDHCP (1) ;
Visual C#
    status = MyPTE1.SaveEthernet_UseDHCP (1) ;
Matlab
    [status] = MyPTE1.SaveEthernet_UseDHCP (1)
    
```

See Also

[Get DHCP Status](#)

2.7 (o) - Use Password

Declaration

```
short SaveEthernet_UsePWD (int UsePwd)
```

Description

This function enables or disables the password requirement for HTTP/Telnet communication with the sensor.

Parameters

Data Type	Variable	Description
int	UseDHCP	Required. Integer value to set the password mode: 0 – Password not required 1 – Password required

Return Values

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

Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_UsePWD (1)
Visual C++
    status = MyPTE1->SaveEthernet_UsePWD (1) ;
Visual C#
    status = MyPTE1.SaveEthernet_UsePWD (1) ;
Matlab
    [status] = MyPTE1.SaveEthernet_UsePWD (1)
    
```

See Also

[Get Password Status](#)

[Get Password](#)

[Set Password](#)

2.7 (p) - Set Password

Declaration

```
short SaveEthernet_PWD (string Pwd)
```

Description

This function sets the password used by the power sensor for HTTP/Telnet communication. The password will not affect sensor operation unless [Use Password](#) is also enabled.

Parameters

Data Type	Variable	Description
string	Pwd	Required. The password to set (20 characters maximum).

Return Values

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

Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_PWD("123")
Visual C++
    status = MyPTE1->SaveEthernet_PWD("123");
Visual C#
    status = MyPTE1.SaveEthernet_PWD("123");
Matlab
    [status] = MyPTE1.SaveEthernet_PWD("123")
    
```

See Also

[Get Password Status](#)
[Get Password](#)
[Use Password](#)

3 - Operating in a Linux Environment via USB

When connected by USB, the computer will recognize the FCPM as a Human Interface Device (HID). In this mode of operation the following USB interrupt codes can be used.

Alternatively, the sensor can be operated over an Ethernet TCP/IP Network (see [Ethernet Control over IP Networks](#) for details).

To open a connection to the integrated frequency & power meter, the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- Power Sensor Product ID: 0x11

Communication with the sensor 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 sensor.

Worked examples can be found in the [Programming Examples & Troubleshooting Guide](#), downloadable from the Mini-Circuits website. The examples use the libhid and libusb libraries to interface with the device as a USB HID (Human Interface Device).

3.1 - Interrupts - General Functions

#	Description	Command Code (Byte 0)
a	Get Device Model Name	104
b	Get Device Serial Number	105
c	Get Internal Temperature	103
d	Get Firmware	99

3.1 (a) - Get Device Model Name

Description

Returns the full Mini-Circuits part number of the connected sensor.

Transmit Array

Byte	Data	Description
0	104	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	104	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' FCPM-6000RC integrated frequency & power meter. See [Appendix A](#) for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6
Value	104	70	67	80	77	45	54
ASCII Character	N/A	F	C	P	M	-	6

Byte	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Description	Char 7	Char 8	Char 9	Char 10	Char 11	End Marker
Value	48	48	48	82	67	0
ASCII Character	0	0	0	R	C	N/A

See Also

[Get Device Serial Number](#)

3.1 (b) - Get Device Serial Number

Description

Returns the serial number of the connected sensor.

Transmit Array

Byte	Data	Description
0	105	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	105	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 current power sensor 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	105	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](#)

3.1 (c) - Get Internal Temperature

Description

This function returns the internal temperature of the sensor in degrees Celsius, to two decimal places.

Transmit Array

Byte	Data	Description
0	103	Interrupt code for Get Internal Temperature
1-63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	103	Interrupt code for Get Internal Temperature
1	Temp_1	ASCII character code for the first character of the temperature reading
2	Temp_2	ASCII character code for the second character of the temperature reading
3	Temp_3	ASCII character code for the third character of the temperature reading
4	Temp_4	ASCII character code for the fourth character of the temperature reading
5	Temp_5	ASCII character code for the fifth character of the temperature reading
6	Temp_6	ASCII character code for the sixth character of the temperature reading
7-63	Not significant	"Don't care" bytes, can be any value

Example

The below returned array would indicate a temperature of +28.43°C:

Byte	Data	Description
0	103	Interrupt code for Get Internal Temperature
1	43	ASCII character code for "+"
2	50	ASCII character code for "2"
3	56	ASCII character code for "8"
4	46	ASCII character code for "."
5	52	ASCII character code for "4"
6	51	ASCII character code for "3"
7 to 63	Not significant	"Don't care" bytes, can be any value

3.1 (d) - Get Firmware

Description

Returns the internal firmware version of the sensor.

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 power sensor 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

3.2 - Interrupts - Power Measurement Functions

#	Description	Command Code (Byte 0)
a	Set Measurement Mode	15
b	Set Compensation Frequency Mode	116
c	Get Compensation Frequency Mode	113
d	Read Power	102

3.2 (a) - Set Measurement Mode

Description

Sets the power measurement mode of the sensor between "low noise" and "fast sampling" modes; the default is "low noise" mode. See the individual model datasheets for specifications.

Transmit Array

Byte	Data	Description
0	15	Interrupt code for Set Measurement Mode
1	Mode	Integer value to set the required mode: 0 = Low noise mode 1 = Fast sampling mode
2- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	15	Interrupt code for Set Measurement Mode
1 to 63	Not significant	"Don't care" bytes, can be any value

Example

Byte	Data	Description
0	15	Interrupt code for Set Measurement Mode
1	1	Set power sensor to "fast sampling" mode
2- 63	Not significant	"Don't care" bytes, can be any value

3.2 (b) - Set Compensation Frequency Mode

Description

Sets whether the frequency and power meter is to use automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Transmit Array

Byte	Data	Description
0	116	Interrupt code for Set Compensation Frequency Mode
1	Mode	Integer value to set the compensation frequency mode: 0 = Manual (user must specify frequency when reading power) 1 = Automatic (power reading will be compensated based on measured frequency)
2- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	116	Interrupt code for Set Compensation Frequency Mode
1 to 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array sets the sensor to automatic frequency compensation mode:

Byte	Data	Description
0	116	Interrupt code for Set Compensation Frequency Mode
1	1	Set power sensor to "automatic" mode
2- 63	Not significant	"Don't care" bytes, can be any value

See Also

[Read Power](#)

[Get Compensation Frequency Mode](#)

3.2 (c) - Get Compensation Frequency Mode

Description

Indicates whether the frequency and power meter is using automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Transmit Array

Byte	Data	Description
0	113	Interrupt code for Get Compensation Frequency Mode
1- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	113	Interrupt code for Get Compensation Frequency Mode
1	Mode	Integer value indicating the compensation frequency mode: 0 = Manual (user must specify frequency when reading power) 1 = Automatic (power reading will be compensated based on measured frequency)
2 to 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the sensor is using manual frequency compensation mode so the user must specify frequency when reading the power:

Byte	Data	Description
0	113	Interrupt code for Get Compensation Frequency Mode
1	0	Power sensor operating in "manual" mode
2- 63	Not significant	"Don't care" bytes, could be any value

See Also

[Read Power](#)
[Set Compensation Frequency Mode](#)

3.2 (d) - Read Power

Description

Returns the power measurement at the sensor head, compensated for frequency and temperature. In manual frequency compensation mode, the user must enter the compensation frequency in order to achieve the specified power accuracy. In automatic frequency compensation mode, the frequency & power meter uses the simultaneous frequency measurement for compensating the power reading.

Transmit Array

Byte	Data	Description
0	102	Interrupt code for Read Power
1	Frequency_1	In automatic frequency compensation mode this is a "don't care" byte and can be given any value. In manual frequency compensation mode this is the first byte of the compensation frequency for the power reading: $\text{Frequency}_1 = \text{INT}(\text{FREQUENCY} / 256)$
2	Frequency_2	In automatic frequency compensation mode this is a "don't care" byte and can be given any value. In manual frequency compensation mode this is the second byte of the compensation frequency for the power reading: $\text{Frequency}_2 = \text{FREQUENCY} - (\text{Frequency}_1 * 256)$
3	Freq_Units	In automatic frequency compensation mode this is a "don't care" byte and can be given any value. In manual frequency compensation mode this is the ASCII character code representing the units for the compensation frequency: 75 = ASCII code for "K" (frequency units are KHz) 77 = ASCII code for "M" (frequency units are MHz)
4- 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	102	Interrupt code for Read Power
1	Power_1	ASCII character code for the first character of the power reading
2	Power_2	ASCII character code for the second character of the power reading
3	Power_3	ASCII character code for the third character of the power reading
4	Power_4	ASCII character code for the fourth character of the power reading
5	Power_5	ASCII character code for the fifth character of the power reading
6	Power_6	ASCII character code for the sixth character of the power reading
7 to 63	Not significant	"Don't care" bytes, can be any value

Example

The following transmit array would be sent to read the power for an expected signal at 1250 MHz when operating in manual frequency compensation mode:

Byte	Data	Description
0	102	Interrupt code for Read Power
1	4	Frequency_1 = INT (1250 / 256)
2	226	Frequency_2 = 1250 - (4 * 256)
3	77	ASCII code for "M" (frequency units are MHz)
4 - 63	Not significant	"Don't care" bytes, can be any value

The following transmit array would be sent to read the power when operating in automatic frequency compensation mode:

Byte	Data	Description
0	102	Interrupt code for Read Power
1 - 63	Not significant	"Don't care" bytes, can be any value

The following array would be returned in either case to indicate a power reading of -10.65dBm:

Byte	Data	Description
0	102	Interrupt code for Read Power
1	45	ASCII character code for "-"
2	49	ASCII character code for "1"
3	48	ASCII character code for "0"
4	46	ASCII character code for "."
5	54	ASCII character code for "6"
6	53	ASCII character code for "5"
7 to 63	Not significant	"Don't care" bytes, can be any value

See Also

[Read Frequency](#)
[Set Compensation Frequency Mode](#)
[Get Compensation Frequency Mode](#)

3.3 - Interrupts - Frequency Measurement Functions

#	Description	Command Code (Byte 0)
a	Set Range	115
b	Get Range	111
c	Get Requested Range	135
d	Set Sample Time	117
e	Get Sample Time	134
f	Get Reference Source	112
g	Read Frequency	110

3.3 (a) - Set Range

Description

Sets the frequency measurement range of the sensor. 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.

Transmit Array

Byte	Data	Description
0	115	Interrupt code for Set Range
1	Range	Integer value corresponding to the input frequency range: 0 = Automatic (1 to 6000 MHz) 1 = 1 to 40 MHz 2 = 40 to 190 MHz 3 = 190 to 1400 MHz 4 = 1400 to 6000 MHz
2- 63	Not significant	“Don’t care” bytes, can be any value

Returned Array

Byte	Data	Description
0	115	Interrupt code for Set Range
1 to 63	Not significant	“Don’t care” bytes, could be any value

Example

The following transmit array sets the sensor to range 3 (expecting an input signal between 190 MHz and 1400 MHz):

Byte	Data	Description
0	115	Interrupt code for Set Range
1	3	Range 3 (190 to 1400 MHz)
2- 63	Not significant	“Don’t care” bytes, can be any value

See Also

[Get Range](#)
[Get Requested Range](#)

3.3 (b) - Get Range

Description

Returns the actual frequency measurement range of the sensor when in automatic range mode.

Transmit Array

Byte	Data	Description
0	111	Interrupt code for Get Range
1 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	111	Interrupt code for Get Range
1	Range	Integer value corresponding to the input frequency range: 1 = 1 to 40 MHz 2 = 40 to 190 MHz 3 = 190 to 1400 MHz 4 = 1400 to 6000 MHz
2 to 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array indicates that the sensor is detecting an input signal in range 2 (between 40 MHz and 190 MHz):

Byte	Data	Description
0	111	Interrupt code for Get Range
1	2	Range 2 (40 to 190 MHz)
2 to 63	Not significant	"Don't care" bytes, could be any value

See Also

[Set Range](#)

[Get Requested Range](#)

3.3 (c) - Get Requested Range

Description

Returns the user requested frequency measurement range of the sensor.

Transmit Array

Byte	Data	Description
0	135	Interrupt code for Get Requested Range
1 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	135	Interrupt code for Get Requested Range
1	Range	Integer value corresponding to the input frequency range: 0 = Automatic (1 to 6000 MHz) 1 = 1 to 40 MHz 2 = 40 to 190 MHz 3 = 190 to 1400 MHz 4 = 1400 to 6000 MHz
2 to 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array indicates that the sensor is automatically setting the input range:

Byte	Data	Description
0	135	Interrupt code for Get Requested Range
1	0	Automatic range mode
2 to 63	Not significant	"Don't care" bytes, could be any value

See Also

[Set Range](#)

[Get Range](#)

3.3 (d) - Set Sample Time

Description

Sets the sample time to be used for frequency measurements, from 100 to 3000 ms, in 100 ms steps. The default sample time is 1000 ms (1 second).

Transmit Array

Byte	Data	Description
0	117	Interrupt code for Set Sample Time
1 to (n-1)	Sample_Time	The sample time in ms as a string of characters represented by ASCII character codes, 1 code per byte
n	0	Zero value byte to indicate the end of the sample time string
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	117	Interrupt code for Set Sample Time
1 to 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array sets the frequency sample time to 1500 ms:

Byte	Data	Description
0	117	Interrupt code for Set Sample Time
1	49	ASCII character code for "1"
2	53	ASCII character code for "5"
3	48	ASCII character code for "0"
4	48	ASCII character code for "0"
5	0	Zero value byte to indicate the end of the sample time string
6 to 63	Not significant	"Don't care" bytes, can be any value

See Also

[Get Sample Time](#)

3.3 (e) - Get Sample Time

Description

Returns the time in milliseconds over which the input frequency will be sampled.

Transmit Array

Byte	Data	Description
0	134	Interrupt code for Get Sample Time
1 to 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	134	Interrupt code for Get Sample Time
1 to (n-1)	Sample_Time	The sample time in ms as a string of characters represented by ASCII character codes, 1 code per byte
n	0	Zero value byte to indicate the end of the sample time string
(n+1) to 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the frequency sample time is currently set to 300 ms:

Byte	Data	Description
0	134	Interrupt code for Get Sample Time
1	51	ASCII character code for "3"
2	48	ASCII character code for "0"
3	48	ASCII character code for "0"
4	0	Zero value byte to indicate the end of the sample time string
5 to 63	Not significant	"Don't care" bytes, could be any value

See Also

[Set Sample Time](#)

3.3 (f) - Get Reference Source

Description

Indicates whether the sensor is using the internal reference for frequency measurements or an external source. The reference source will automatically switch to external if a suitable signal is detected at the Ref In port.

Transmit Array

Byte	Data	Description
0	112	Interrupt code for Get Reference Source
1 to 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	112	Interrupt code for Get Reference Source
1 to (n-1)	Ref_Source	The reference source as a string of characters represented by ASCII character codes, 1 code per byte. The string representation will be either of: <ul style="list-style-type: none"> "ExtRef" for external reference "IntRef" for internal reference
n	0	Zero value byte to indicate the end of the string
(n+1) to 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the sensor is using the internal reference source:

Byte	Data	Description
0	112	Interrupt code for Get Reference Source
1	73	ASCII character code for "I"
2	110	ASCII character code for "n"
3	116	ASCII character code for "t"
4	82	ASCII character code for "R"
5	101	ASCII character code for "e"
6	102	ASCII character code for "f"
7	0	Zero value byte to indicate the end of the sample time string
8 to 63	Not significant	"Don't care" bytes, could be any value

3.3 (g) - Read Frequency

Description

Returns the measured input frequency in MHz.

Transmit Array

Byte	Data	Description
0	110	Interrupt code for Read Frequency
1 to 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	110	Interrupt code for Read Frequency
1 to (n-1)	Frequency	The frequency in MHz as a string of characters represented by ASCII character codes, 1 code per byte
n	0	Zero value byte to indicate the end of the frequency string
(n+1) to 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the measured frequency is 3101.25 MHz:

Byte	Data	Description
0	110	Interrupt code for Read Frequency
1	51	ASCII character code for "3"
2	49	ASCII character code for "1"
3	48	ASCII character code for "0"
4	49	ASCII character code for "1"
5	46	ASCII character code for "."
6	50	ASCII character code for "2"
7	53	ASCII character code for "5"
8	0	Zero value byte to indicate the end of the frequency string
9 to 63	Not significant	"Don't care" bytes, could be any value

See Also

[Read Power](#)

3.4 - Interrupts - Ethernet Configuration Functions

	Description	Command Code	
		Byte 0	Byte 1
a	Set Static IP Address	250	201
b	Set Static Subnet Mask	250	202
c	Set Static Network Gateway	250	203
d	Set HTTP Port	250	204
e	Set Telnet Port	250	214
f	Use Password	250	205
g	Set Password	250	206
h	Use DHCP	250	207
i	Get Static IP Address	251	201
j	Get Static Subnet Mask	251	202
k	Get Static Network Gateway	251	203
l	Get HTTP Port	251	204
m	Get Telnet Port	251	214
n	Get Password Status	251	205
o	Get Password	251	206
p	Get DHCP Status	251	207
q	Get Dynamic Ethernet Configuration	253	
r	Get MAC Address	252	
s	Reset Ethernet Configuration	101	101

3.4 (a) - Set Static IP Address

Description

Sets the static IP address to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	201	Interrupt code for Set IP Address
2	IP_Byte0	First byte of IP address
3	IP_Byte1	Second byte of IP address
4	IP_Byte2	Third byte of IP address
5	IP_Byte3	Fourth byte of IP address
6 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To set the static IP address to 192.168.100.100, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	201	Interrupt code for Set IP Address
2	192	First byte of IP address
3	168	Second byte of IP address
4	100	Third byte of IP address
5	100	Fourth byte of IP address

See Also

[Use DHCP](#)
[Get Static IP Address](#)
[Reset Ethernet Configuration](#)

3.4 (b) - Set Static Subnet Mask

Description

Sets the static subnet mask to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	202	Interrupt code for Set Subnet Mask
2	IP_Byte0	First byte of subnet mask
3	IP_Byte1	Second byte of subnet mask
4	IP_Byte2	Third byte of subnet mask
5	IP_Byte3	Fourth byte of subnet mask
6 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To set the static subnet mask to 255.255.255.0, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	202	Interrupt code for Set Subnet Mask
2	255	First byte of subnet mask
3	255	Second byte of subnet mask
4	255	Third byte of subnet mask
5	0	Fourth byte of subnet mask

See Also

[Use DHCP](#)
[Get Static Subnet Mask](#)
[Reset Ethernet Configuration](#)

3.4 (c) - Set Static Network Gateway

Description

Sets the network gateway IP address to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	203	Interrupt code for Set Network Gateway
2	IP_Byte0	First byte of network gateway IP address
3	IP_Byte1	Second byte of network gateway IP address
4	IP_Byte2	Third byte of network gateway IP address
5	IP_Byte3	Fourth byte of network gateway IP address
6 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To set the static IP address to 192.168.100.0, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	203	Interrupt code for Set Network Gateway
2	192	First byte of IP address
3	168	Second byte of IP address
4	100	Third byte of IP address
5	0	Fourth byte of IP address

See Also

[Use DHCP](#)
[Get Static Network Gateway](#)
[Reset Ethernet Configuration](#)

3.4 (d) - Set HTTP Port

Description

Sets the port to be used for HTTP communication (default is port 80).

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	204	Interrupt code for Set HTTP Port
2	Port_Byte0	First byte (MSB) of HTTP port value: Port_Byte0 = INTEGER (Port / 256)
3	Port_Byte1	Second byte (LSB) of HTTP port value: Port_byte1 = Port - (Port_Byte0 * 256)
4 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To set the HTTP port to 8080, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	204	Interrupt code for Set HTTP Port
2	31	Port_Byte0 = INTEGER (8080 / 256)
3	144	Port_byte1 = 8080 - (31 * 256)

See Also

[Set Telnet Port](#)
[Get HTTP Port](#)
[Get Telnet Port](#)
[Reset Ethernet Configuration](#)

3.4 (e) - Set Telnet Port

Description

Sets the port to be used for Telnet communication (default is port 23).

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	214	Interrupt code for Set Telnet Port
2	Port_Byte0	First byte (MSB) of Telnet port value: Port_Byte0 = INTEGER (Port / 256)
3	Port_Byte1	Second byte (LSB) of Telnet port value: Port_byte1 = Port - (Port_Byte0 * 256)
4 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To set the Telnet port to 22, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	214	Interrupt code for Set Telnet Port
2	0	Port_Byte0 = INTEGER (22 / 256)
3	22	Port_byte1 = 22 - (0 * 256)

See Also

[Set HTTP Port](#)
[Get HTTP Port](#)
[Get Telnet Port](#)
[Reset Ethernet Configuration](#)

3.4 (f) - Use Password

Description

Enables or disables the requirement to password protect the HTTP / Telnet communication.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	205	Interrupt code for Use Password
2	PW_Mode	0 = password not required (default) 1 = password required
3 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To enable the password requirement for Ethernet communication, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	205	Interrupt code for Use Password
2	1	Enable password requirement

See Also

[Set Password](#)
[Get Password Status](#)
[Get Password](#)
[Reset Ethernet Configuration](#)

3.4 (g) - Set Password

Description

Sets the password to be used for Ethernet communication (when password security is enabled, maximum 20 characters).

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	206	Interrupt code for Set Password
2	PW_Length	Length (number of characters) of the password
3 to n	PW_Char	Series of ASCII character codes (1 per byte) for the Ethernet password
n + 1 to 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 to 63	Not significant	Any value

Example

To set the password to *Pass_123*, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	206	Interrupt code for Set Password
2	8	Length of password (8 characters)
3	80	ASCII character code for P
4	97	ASCII character code for a
5	115	ASCII character code for s
6	115	ASCII character code for s
7	95	ASCII character code for _
8	49	ASCII character code for 1
9	50	ASCII character code for 2
10	51	ASCII character code for 3

See Also

- [Use Password](#)
- [Get Password Status](#)
- [Get Password](#)
- [Reset Ethernet Configuration](#)

3.4 (h) - Use DHCP

Description

Enables or disables DHCP (dynamic host control protocol). With DHCP enabled, the attenuators Ethernet / IP configuration is assigned by the network and any user defined static IP settings are ignored. With DHCP disabled, the user defined static IP settings are used.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	207	Interrupt code for Use DHCP
2	DHCP_Mode	0 = DHCP disabled (static IP settings in use) 1 = DHCP enabled (default - dynamic IP in use)
3 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1 - 63	Not significant	Any value

Example

To enable DHCP for Ethernet communication, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	207	Interrupt code for Use DHCP
2	1	Enable DHCP

See Also

[Use DHCP](#)
[Get DHCP Status](#)
[Get Dynamic Ethernet Configuration](#)
[Reset Ethernet Configuration](#)

3.4 (i) - Get Static IP Address

Description

Gets the static IP address (configured by the user) to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	201	Interrupt code for Get IP Address
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	IP_Byte0	First byte of IP address
2	IP_Byte1	Second byte of IP address
3	IP_Byte2	Third byte of IP address
4	IP_Byte3	Fourth byte of IP address
5 - 63	Not significant	Any value

Example

The following returned array would indicate that a static IP address of 192.168.100.100 has been configured:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	192	First byte of IP address
2	168	Second byte of IP address
3	100	Third byte of IP address
4	100	Fourth byte of IP address

See Also

[Use DHCP](#)
[Set Static IP Address](#)

3.4 (j) - Get Static Subnet Mask

Description

Gets the subnet mask (configured by the user) to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	202	Interrupt code for Get Subnet Mask
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	IP_Byte0	First byte of subnet mask
2	IP_Byte1	Second byte of subnet mask
3	IP_Byte2	Third byte of subnet mask
4	IP_Byte3	Fourth byte of subnet mask
5 - 63	Not significant	Any value

Example

The following returned array would indicate that a subnet mask of 255.255.255.0 has been configured:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	255	First byte of subnet mask
2	255	Second byte of subnet mask
3	255	Third byte of subnet mask
4	0	Fourth byte of subnet mask

See Also

[Use DHCP](#)
[Set Static Subnet Mask](#)

3.4 (k) - Get Static Network Gateway

Description

Gets the static IP address (configured by the user) of the network gateway to be used when DHCP (dynamic host control protocol) is disabled.

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	203	Interrupt code for Get Network Gateway
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	IP_Byte0	First byte of IP address
2	IP_Byte1	Second byte of IP address
3	IP_Byte2	Third byte of IP address
4	IP_Byte3	Fourth byte of IP address
5 - 63	Not significant	Any value

Example

The following returned array would indicate that a network gateway IP address of 192.168.100.0 has been configured:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	192	First byte of IP address
2	168	Second byte of IP address
3	100	Third byte of IP address
4	0	Fourth byte of IP address

See Also

[Use DHCP](#)
[Set Static Network Gateway](#)

3.4 (I) - Get HTTP Port

Description

Gets the port to be used for HTTP communication (default is port 80).

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	204	Interrupt code for Get HTTP Port
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	Port_Byte0	First byte (MSB) of HTTP port value:
2	Port_Byte1	Second byte (LSB) of HTTP port value: Port = (Port_Byte0 * 256) + Port_Byte1
3 - 63	Not significant	Any value

Example

The following returned array would indicate that the HTTP port has been configured as 8080:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	31	
2	144	Port = (31 * 256) + 144 = 8080

See Also

[Set HTTP Port](#)
[Set Telnet Port](#)
[Get Telnet Port](#)

3.4 (m) - Get Telnet Port

Description

Gets the port to be used for Telnet communication (default is port 23).

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	214	Interrupt code for Get Telnet Port
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	Port_Byte0	First byte (MSB) of Telnet port value:
2	Port_Byte1	Second byte (LSB) of Telnet port value: Port = (Port_Byte0 * 256) + Port_Byte1
3 - 63	Not significant	Any value

Example

The following returned array would indicate that the Telnet port has been configured as 22:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	0	
2	22	Port = (0 * 256) + 22 = 22

See Also

[Set HTTP Port](#)
[Set Telnet Port](#)
[Get HTTP Port](#)

3.4 (n) - Get Password Status

Description

Checks whether the attenuators has been configured to require a password for HTTP / Telnet communication.

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	205	Interrupt code for Get Password Status
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Set Ethernet Configuration
1	PW_Mode	0 = password not required (default) 1 = password required
2 - 63	Not significant	Any value

Example

The following returned array indicates that password protection is enabled:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	1	Password protection enabled

See Also

[Use Password](#)
[Set Password](#)
[Get Password](#)

3.4 (o) - Get Password

Description

Gets the password to be used for Ethernet communication (when password security is enabled, maximum 20 characters).

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	206	Interrupt code for Get Password
2 to 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	PW_Length	Length (number of characters) of the password
2 to n	PW_Char	Series of ASCII character codes (1 per byte) for the Ethernet password
n to 63	Not significant	Any value

Example

The following returned array indicated that the password has been set to *Pass_123*:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	8	Length of password (8 characters)
2	80	ASCII character code for P
3	97	ASCII character code for a
4	115	ASCII character code for s
5	115	ASCII character code for s
6	95	ASCII character code for _
7	49	ASCII character code for 1
8	50	ASCII character code for 2
9	51	ASCII character code for 3

See Also

[Use Password](#)
[Set Password](#)
[Get Password Status](#)

3.4 (p) - Get DHCP Status

Description

Checks whether DHCP (dynamic host control protocol) is enabled or disabled. With DHCP enabled, the attenuators Ethernet / IP configuration is assigned by the network and any user defined static IP settings are ignored. With DHCP disabled, the user defined static IP settings are used.

Transmit Array

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	207	Interrupt code for Get DHCP Status
2 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	251	Interrupt code for Set Ethernet Configuration
1	DCHP_Mode	0 = DCHP disabled (static IP settings in use) 1 = DCHP enabled (default - dynamic IP in use)
2 - 63	Not significant	Any value

Example

The following returned array indicates that DHCP is enabled:

Byte	Data	Description
0	251	Interrupt code for Get Ethernet Configuration
1	1	DHCP enabled

See Also

[Use DHCP](#)
[Get Dynamic Ethernet Configuration](#)

3.4 (q) - Get Dynamic Ethernet Configuration

Description

Returns the IP address, subnet mask and default gateway currently used by the device. If DHCP is enabled then these values are assigned by the network DHCP server. If DHCP is disabled then these values are the static configuration defined by the user.

Transmit Array

Byte	Data	Description
0	253	Interrupt code for Get Dynamic Ethernet Configuration
1 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	253	Interrupt code for Get Dynamic Ethernet Configuration
1	IP_Byte0	First byte of IP address
2	IP_Byte1	Second byte of IP address
3	IP_Byte2	Third byte of IP address
4	IP_Byte3	Fourth byte of IP address
5	SM_Byte0	First byte of subnet mask
6	SM_Byte1	Second byte of subnet mask
7	SM_Byte2	Third byte of subnet mask
8	SM_Byte3	Fourth byte of subnet mask
9	NG_Byte0	First byte of network gateway IP address
10	NG_Byte1	Second byte of network gateway IP address
11	NG_Byte2	Third byte of network gateway IP address
12	NG_Byte3	Fourth byte of network gateway IP address
13 - 63	Not significant	Any value

Example

The following returned array would indicate the below Ethernet configuration is active:

- IP Address: 192.168.100.100
- Subnet Mask: 255.255.255.0
- Network Gateway: 192.168.100.0

Byte	Data	Description
0	253	Interrupt code for Get Dynamic Ethernet Configuration
1	192	First byte of IP address
2	168	Second byte of IP address
3	100	Third byte of IP address
4	100	Fourth byte of IP address
5	255	First byte of subnet mask
6	255	Second byte of subnet mask
7	255	Third byte of subnet mask
8	0	Fourth byte of subnet mask
9	192	First byte of network gateway IP address
10	168	Second byte of network gateway IP address
11	100	Third byte of network gateway IP address
12	0	Fourth byte of network gateway IP address

See Also

- [Use DHCP](#)
- [Get DHCP Status](#)

3.4 (r) - Get MAC Address

Description

Returns the MAC address of the device.

Transmit Array

Byte	Data	Description
0	252	Interrupt code for Get MAC Address
1 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	252	Interrupt code for Get MAC Address
1	MAC_Byte0	First byte of MAC address
2	MAC_Byte1	Second byte of MAC address
3	MAC_Byte2	Third byte of MAC address
4	MAC_Byte3	Fourth byte of MAC address
5	MAC_Byte4	Fifth byte of MAC address
6	MAC_Byte5	Sixth byte of MAC address
7 - 63	Not significant	Any value

Example

The following returned array would indicate a MAC address (in decimal notation) of 11:47:165:103:137:171:

Byte	Data	Description
0	252	Interrupt code for Get MAC Address
1	11	First byte of MAC address
2	47	Second byte of MAC address
3	165	Third byte of MAC address
4	103	Fourth byte of MAC address
5	137	Fifth byte of MAC address
6	171	Sixth byte of MAC address

See Also

[Get Dynamic Ethernet Configuration](#)

3.4 (s) - Reset Ethernet Configuration

Description

Forces the device to reset and adopt the latest Ethernet configuration. Must be sent after any changes are made to the configuration.

Transmit Array

Byte	Data	Description
0	101	Reset Ethernet configuration sequence
1	101	Reset Ethernet configuration sequence
2	102	Reset Ethernet configuration sequence
3	103	Reset Ethernet configuration sequence
4 - 63	Not significant	Any value

Returned Array

Byte	Data	Description
0	101	Confirmation of reset Ethernet configuration sequence
1 - 63	Not significant	Any value

4 - Ethernet Control over IP Networks

Mini-Circuits' integrated frequency and power meter models have an RJ45 connector for remote control over Ethernet TCP/IP networks. HTTP (Get/Post commands) and Telnet communication are supported. UDP transmission is also supported for discovering available sensors on the network.

The device can be configured manually with a static IP address or automatically by the network using DHCP (Dynamic Host Control Protocol):

- Dynamic IP (factory default setting)
 - Subnet Mask, Network Gateway and local IP Address are assigned by the network server on each connection
 - The only user controllable parameters are:
 - TCP/IP Port for HTTP communication (the default is port 80)
 - Password (up to 20 characters; default is no password)
- Static IP
 - All parameters must be specified by the user:
 - IP Address (must be a legal and unique address on the local network)
 - Subnet Mask (subnet mask of the local network)
 - Network gateway (the IP address of the network gateway/router)
 - TCP/IP Port for HTTP communication (the default is port 80)
 - Password (up to 20 characters; default is no password)

Notes:

1. The TCP/IP port must be included in every HTTP command to the sensor unless the default port 80 is used
2. The password must be included in every HTTP command to the sensor if password security is enabled
3. Port 23 is reserved for Telnet communication
4. The device draws DC power through the USB type B connector; this can be connected to a computer or the AC mains adapter

4.1 (a) - Configuring Ethernet Settings via USB

The sensor must be connected via the USB interface in order to configure the Ethernet settings. Following initial configuration, the device can be controlled via the Ethernet interface with no further need for a USB connection. The API DLL provides the functionality for configuring the Ethernet settings over a USB connection (see [DLL Functions for Ethernet Configuration](#) for full details).

4.2 - Ethernet Communication Methodology

Communication over Ethernet is accomplished by sending SCPI commands using either HTTP (Get/Post commands) or Telnet communication. The HTTP and Telnet protocols are both commonly supported and simple to implement in most programming languages. Any Internet browser can be used as a console/tester for HTTP control by typing the commands/queries directly into the address bar. The SCPI commands that can be sent to integrated frequency and power meter series are detailed in the [SCPI Command Set for Ethernet Control](#) section.

4.2 (a) - Setting Sensor Properties Using HTTP and SCPI

The basic format of the HTTP command to set the sensor is:

<http://ADDRESS:PORT/PWD;COMMAND>

Where

- `http://` is required
- ADDRESS = IP address (required)
- PORT = TCP/IP port (can be omitted if port 80 is used)
- PWD = Password (can be omitted if password security is not enabled)
- COMMAND = Command to send to the power sensor

Example 1:

<http://192.168.100.100:800/PWD=123;FREQ:1000>

Explanation:

- The sensor has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to "123"
- The command is to set the compensation frequency to 1000MHz (see below for the full explanation of all commands/queries)

Example 2:

<http://10.10.10.10/:TEMP:FORMAT:F>

Explanation:

- The sensor has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to set the temperature format to Fahrenheit (see below for the full explanation of all commands/queries)

4.2 (b) - Querying Power Sensor Properties Using HTTP and SCPI

The basic format of the HTTP command to query the sensor is:

<http://ADDRESS:PORT/PWD;QUERY?>

Where

- `http://` is required
- `ADDRESS` = IP address (required)
- `PORT` = TCP/IP port (can be omitted if port 80 is used)
- `PWD` = Password (can be omitted if password security is not enabled)
- `QUERY?` = Query to send to the sensor

Example 1:

<http://192.168.100.100:800/PWD=123;;MN?>

Explanation:

- The sensor has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to “123”
- The query is to return the model name of the sensor (see below for the full explanation of all commands/queries)

Example 2:

<http://10.10.10.10/:POWER?>

Explanation:

- The sensor has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The query is to return the current power reading (see below for the full explanation of all commands/queries)

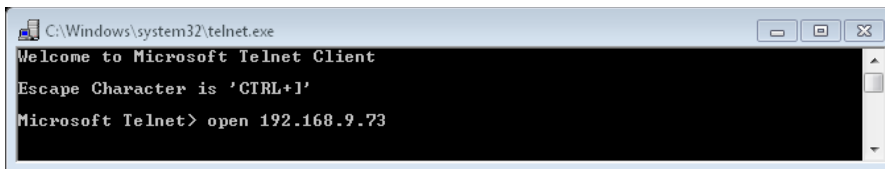
The device will return the result of the query as a string of ASCII characters.

4.2 (c) - Communication Using Telnet and SCPI

Communication with the device is started by creating a Telnet connection to the sensor's IP address. On successful connection the “line feed” character will be returned. If the sensor has a password enabled then this must be sent as the first command after connection.

The full list of all SCPI commands and queries is detailed in the following sections. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below:

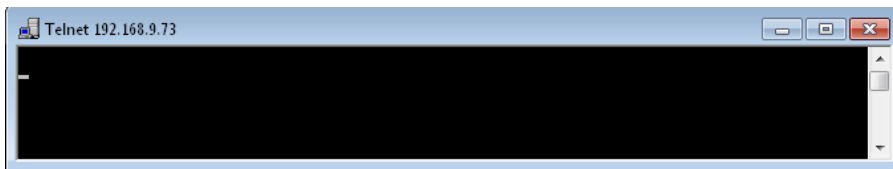
- 1) Set up Telnet connection to a sensor with IP address 192.168.9.73



```

C:\Windows\system32\telnet.exe
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+I'
Microsoft Telnet> open 192.168.9.73
    
```

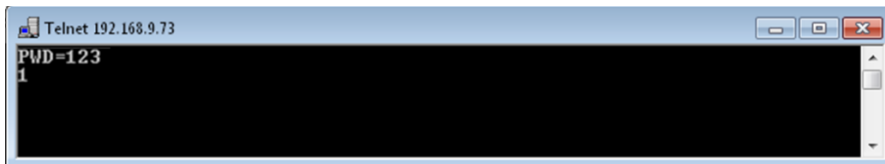
- 2) The “line feed” character is returned indicating the connection was successful:



```

Telnet 192.168.9.73
-
    
```

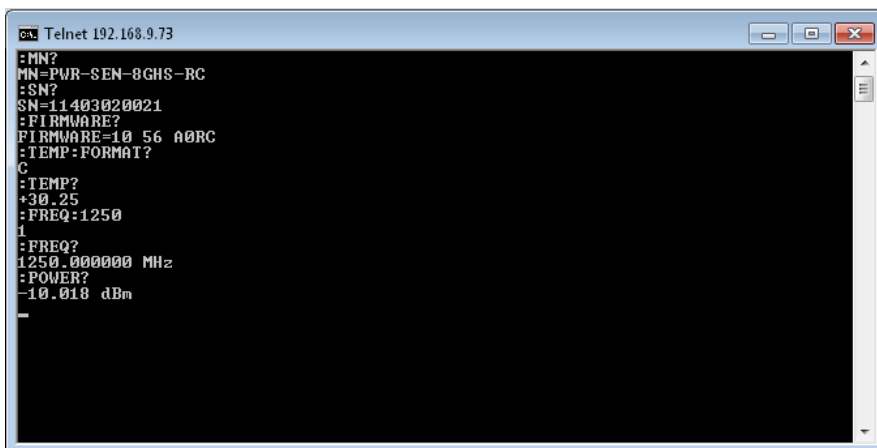
- 3) The password (if enabled) must be sent as the first command; a return value of 1 indicates success:



```

Telnet 192.168.9.73
PWD=123
1
    
```

- 4) Any number of commands and queries can be sent as needed:



```

Telnet 192.168.9.73
:MN?
MN=PWR-SEN-8GHS-RC
:SN?
SN=11403020021
:FIRMWARE?
FIRMWARE=10 56 A0RC
:TEMP:FORMAT?
C
:TEMP?
+30.25
:FREQ:1250
1
:FREQ?
1250.000000 MHz
:POWER?
-10.010 dBm
-
    
```

4.3 - Device Discovery Using UDP

In addition to HTTP and Telnet, the integrated frequency and power sensors also provide limited support of the UDP protocol for the purpose of “device discovery.” This allows a user to request the IP address and configuration of all Mini-Circuits sensors connected on the network; full control of those units is then accomplished using HTTP or Telnet, as detailed previously.

Alternatively, the IP configuration can be identified or changed by connecting the sensor with the USB interface (see [Configuring Ethernet Settings](#)).

Note: UDP is a simple transmission protocol that provides no method for error correction or guarantee of receipt.

UDP Ports

Mini-Circuits’ sensors are configured to listen on UDP port 4950 and answer on UDP port 4951. Communication on these ports must be allowed through the computer’s firewall in order to use UDP for device discovery. If the sensor’s IP address is already known it is not necessary to use UDP.

Transmission

The command [MCL_POWERSENSOR?](#) should be broadcast to the local network using UDP protocol on port 4950.

Receipt

All Mini-Circuits sensors that receive the request will respond with the following information (each field separated by CrLf) on port 4951:

- Model Name
- Serial Number
- IP Address/Port
- Subnet Mask
- Network Gateway
- MAC Address

Example

Sent Data:

MCL_POWERSENSOR?

Received Data:

Model Name: FCPM-6000RC
Serial Number: 11402120001
IP Address=192.168.9.101 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-01

Model Name: FCPM-6000RC
Serial Number: 11402120002
IP Address=192.168.9.102 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-02

Model Name: FCPM-6000RC
Serial Number: 11402120003
IP Address=192.168.9.103 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-03

5 - SCPI Commands for Ethernet Control

This section details the control functions applicable to Mini-Circuits' integrated frequency and power meters, using SCPI communication. SCPI (Standard Commands for Programmable Instruments) is a common method for controlling instrumentation products.

The SCPI commands are sent as an ASCII text string (up to 63 characters) in the below format:

```
:COMMAND : [value] : [suffix]
```

Where:

COMMAND = the command/query to send
[value] = the value (if applicable) to set
[suffix] = the units (if applicable) that apply to the value

Commands can be sent in upper or lower case and the return value will be an ASCII text string. If an unrecognized command/query is received the sensor will return:

```
-99 Unrecognized Command. Model=[ModelName] SN=[SerialNumber]
```

These functions can be called using HTTP get/post commands or Telnet over a TCP/IP network when the device is connected via the Ethernet RJ45 port (see [Ethernet Control over IP Networks](#)).

5.1 - Summary of SCPI Commands / Queries

5.1 (a) - SCPI - General Commands

	Description	Command/Query
a	Get Model Name	:MN?
b	Get Serial Number	:SN?
c	Get Firmware	:FIRMWARE?
d	Get Temperature Units	:TEMP:FORMAT?
e	Set Temperature Units	:TEMP:FORMAT:[units]
f	Get Internal Temperature	:TEMP?
g	Get LCD Format	:READ:FORMAT?
h	Set LCD Format	:READ:FORMAT:[units]

5.1 (b) - SCPI - Power Measurement Commands

	Description	Command/Query
a	Get Measurement Mode	:MODE?
b	Set Measurement Mode	:MODE: [speed]
c	Get Averaging Mode	:AVG:STATE?
d	Set Averaging Mode	:AVG:STATE: [mode]
e	Get Average Count	:AVG:COUNT?
f	Set Average Count	:AVG:COUNT: [count]
g	Get Compensation Frequency	:FREQ?
h	Set Compensation Frequency	:FREQ: [freq]
i	Get Compensation Frequency Mode	:FC:AUTOFREQ?
j	Set Compensation Frequency Mode	:FC:AUTOFREQ: [mode]
k	Read Power	:POWER?
l	Read Voltage	:VOLTAGE?
m	Get LCD Offset Mode	:OFFSETVALUE:ENABLE?
n	Set LCD Offset Mode	:OFFSETVALUE:ENABLE: [value]
o	Get LCD Offset Value	:OFFSETVALUE:VALUE?
p	Set LCD Offset Value	:OFFSETVALUE:VALUE: [value]

5.1 (c) - SCPI - Frequency Measurement Commands

	Description	Command/Query
a	Get Range	:FC:RANGE?
b	Get Requested Range	:FC:RRANGE?
c	Set Range	:FC:RANGE: [range]
d	Get Sample Time	:FC:SAMPLETIME?
e	Set Sample Time	:FC:SAMPLETIME: [time]
f	Get Frequency	:FC:FREQ?
g	Get Reference Mode	:FC:REF?

5.2 - SCPI - General Commands

5.2 (a) - Get Model Name

Description

Returns the full Mini-Circuits part number of the sensor.

Command Syntax

`:MN?`

Return string

`MN=[model]`

Variable	Description
<code>[model]</code>	Full model name of the sensor (for example, "FCPM-6000RC")

Examples

string to Send	string Returned
<code>:MN?</code>	<code>MN=FCPM-6000RC</code>

HTTP Implementation: <http://10.10.10.10/:MN?>

See Also

[Get Serial Number](#)

5.2 (b) - Get Serial Number

Description

Returns the serial number of the sensor.

Command Syntax

`:SN?`

Return string

`SN=[serial]`

Variable	Description
<code>[serial]</code>	Serial number of the sensor (for example, "11401010001")

Examples

string to Send	string Returned
<code>:SN?</code>	<code>SN=11401010001</code>

HTTP Implementation: `http://10.10.10.10/:SN?`

See Also

[Get Model Name](#)

5.2 (c) - Get Firmware

Description

Returns the firmware version of the sensor.

Command Syntax

`:FIRMWARE?`

Return string

`FIRMWARE=[firmware]`

Variable	Description
<code>[firmware]</code>	Firmware version name (for example, "A1")

Examples

string to Send	string Returned
<code>:FIRMWARE?</code>	<code>FIRMWARE=A1</code>

HTTP Implementation: `http://10.10.10.10/:FIRMWARE?`

5.2 (d) - Get Temperature Units

Description

Returns the units to be used by the sensor's internal temperature sensor, either degrees Celsius or Fahrenheit.

Command Syntax

`:TEMP:FORMAT?`

Return string

`[units]`

Variable	Value	Description
<code>[units]</code>	F	Temperature measurements in degrees Fahrenheit
	C	Temperature measurements in degrees Celsius

Examples

string to Send	string Returned
<code>:TEMP:FORMAT?</code>	C

HTTP Implementation: <http://10.10.10.10/:TEMP:FORMAT?>

See Also

[Set Temperature Units](#)
[Get Internal Temperature](#)

5.2 (e) - Set Temperature Units

Description

Sets the units to be used by the sensor's internal temperature sensor, either degrees Celsius or Fahrenheit.

Command Syntax

`:TEMP:FORMAT:[units]`

Variable	Value	Description
<code>[units]</code>	F	Set temperature measurements to degrees Fahrenheit
	C	Set temperature measurements to degrees Celsius

Return string

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:TEMP:FORMAT:F</code>	1
<code>:TEMP:FORMAT:C</code>	1

HTTP Implementation: <http://10.10.10.10/:TEMP:FORMAT?:C>

See Also

[Get Temperature Units](#)
[Get Internal Temperature](#)

5.2 (f) - Get Internal Temperature

Description

Returns the internal temperature of the sensor in degrees Celsius or Fahrenheit, as defined by the user.

Command Syntax

`:TEMP?`

Return string

`[temperature]`

Variable	Description
<code>[temperature]</code>	The temperature returned from the specified sensor

Examples

string to Send	string Returned
<code>:TEMP?</code>	<code>+25.50</code>

HTTP Implementation: `http://10.10.10.10/:TEMP?`

See Also

[Get Temperature Units](#)

[Set Temperature Units](#)

5.2 (g) - Get LCD Format

Description

Indicates whether power readings are to be displayed in dBm or Watts on the device's LCD.

Command Syntax

`:READ:FORMAT?`

Return string

`[units]`

Variable	Value	Description
<code>[units]</code>	D	LCD power readings in dBm
	W	LDC power readings in Watts

Examples

string to Send	string Returned
<code>:READ:FORMAT?</code>	D

HTTP Implementation: <http://10.10.10.10/:READ:FORMAT?>

See Also

[Set LCD Format](#)

5.2 (h) - Set LCD Format

Description

Sets whether power readings are to be displayed in dBm or Watts on the device's LCD.

Command Syntax

`:READ:FORMAT:[units]`

Variable	Value	Description
[units]	D	LCD power readings in dBm
	W	LDC power readings in Watts

Return string

`[status]`

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:READ:FORMAT:D</code>	1
<code>:READ:FORMAT:W</code>	1

HTTP Implementation: `http://10.10.10.10/:READ:FORMAT?:C`

See Also

[Get LCD Format](#)

5.3 - SCPI - Power Measurement Commands

5.3 (a) - Get Measurement Mode

Description

Returns an integer indicating the power measurement mode of the sensor; "low noise" or "fast sampling". The specifications for these modes are defined in the individual model datasheets. The default is "low noise" mode.

Command Syntax

`:MODE?`

Return string

`[speed]`

Variable	Value	Description
<code>[speed]</code>	0	Low noise mode
	1	Fast sampling mode

Examples

string to Send	string Returned
<code>:MODE?</code>	1

HTTP Implementation: `http://10.10.10.10/:MODE?`

See Also

[Set Measurement Mode](#)

5.3 (b) - Set Measurement Mode

Description

Sets the measurement mode of the sensor between "low noise" and "fast sampling" modes. The specifications for these modes are defined in the individual model datasheets. The default is "low noise" mode.

Command Syntax

:MODE: [*speed*]

Variable	Value	Description
[<i>speed</i>]	0	Low noise mode
	1	Fast sampling mode

Return string

[*status*]

Variable	Value	Description
[<i>status</i>]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
:MODE:0	1
:MODE:1	1
:MODE:2	1

HTTP Implementation: `http://10.10.10.10/:MODE:1`

See Also

[Get Measurement Mode](#)

5.3 (c) - Get Averaging Mode

Description

Indicates whether “averaging” mode is currently enable for the sensor. The default is averaging disabled.

Command Syntax

`:AVG:STATE?`

Return string

`[mode]`

Variable	Value	Description
<code>[mode]</code>	0	Averaging mode disabled
	1	Averaging mode enabled

Examples

string to Send	string Returned
<code>:AVG:STATE?</code>	1

HTTP Implementation: `http://10.10.10.10/:AVG:STATE?`

See Also

[Set Averaging Mode](#)
[Get Average Count](#)
[Set Average Count](#)

5.3 (d) - Set Averaging Mode

Description

Enables or disables the sensor “averaging” mode.

Command Syntax

:AVG:STATE:[mode]

Variable	Value	Description
[mode]	0	Averaging mode disabled
	1	Averaging mode enabled

Return string

[status]

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
:AVG:STATE:1	1

HTTP Implementation: `http://10.10.10.10/:AVG:STATE:1`

See Also

[Get Averaging Mode](#)
[Get Average Count](#)
[Set Average Count](#)

5.3 (e) - Get Average Count

Description

Returns the number of power readings over which the measurement will be averaged when averaging mode is enabled. The default value is 1 (average the reading over 1 measurement).

Command Syntax

`:AVG:COUNT?`

Return string

`[count]`

Variable	Description
<code>[count]</code>	The number of power readings over which to average the measurement

Examples

string to Send	string Returned
<code>:AVG:COUNT?</code>	3

HTTP Implementation: <http://10.10.10.10/:AVG:COUNT?>

See Also

[Get Averaging Mode](#)

[Set Averaging Mode](#)

[Set Average Count](#)

5.3 (f) - Set Average Count

Description

Sets the number of power readings over which to average the measurement when averaging mode is enabled. The default value is 1 (average the reading over 1 measurement).

Command Syntax

:AVG:COUNT: `[count]`

Variable	Description
<code>[count]</code>	The number of readings over which to average the power reading

Return string

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:AVG:COUNT:10</code>	1

HTTP Implementation: <http://10.10.10.10/:AVG:COUNT:10>

See Also

[Get Averaging Mode](#)
[Set Averaging Mode](#)
[Get Average Count](#)

5.3 (g) - Get Compensation Frequency

Description

Returns the frequency currently in use for calibrating the input power measurements.

Command Syntax

`:FREQ?`

Return string

`[freq]`

Variable	Description
<code>[freq]</code>	The current compensation frequency in MHz

Examples

string to Send	string Returned
<code>:FREQ?</code>	2500.000000 MHz

HTTP Implementation: <http://10.10.10.10/:FREQ?>

See Also

[Set Compensation Frequency](#)
[Read Power](#)

5.3 (h) - Set Compensation Frequency

Description

Sets the compensation frequency of the sensor head during operation in manual compensation mode; this needs to be set in order to achieve the specified power measurement accuracy. In automatic frequency compensation mode, the power reading is automatically compensated based on the sensor's simultaneous frequency measurement.

Note: This property will not filter out unwanted signals.

Command Syntax

:FREQ: [freq]

Variable	Description
[freq]	The current compensation frequency in MHz

Return string

[status]

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
:FREQ:2500	1

HTTP Implementation: <http://10.10.10.10/:FREQ:2500>

See Also

[Get Compensation Frequency](#)

[Read Power](#)

5.3 (i) - Get Compensation Frequency Mode

Description

Indicates whether the power sensor/frequency counter is using automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Command Syntax

:FC:AUTOFREQ?

Return string

[mode]

Variable	Value	Description
[mode]	0	Manual compensation mode (expected input frequency must be specified by user)
	1	Automatic compensation mode (the power reading compensation is set based on the measured input frequency)

Examples

string to Send	string Returned
:FC:AUTOFREQ?	1

HTTP Implementation: <http://10.10.10.10/:FC:AUTOFREQ?>

See Also

[Get Compensation Frequency](#)
[Set Compensation Frequency](#)
[Set Compensation Frequency Mode](#)

5.3 (j) - Set Compensation Frequency Mode

Description

Sets whether the power sensor/frequency counter is to use automatic or manual mode for compensating power readings. In automatic mode, the sensor monitors the frequency and uses this to compensate the power reading based on an internal look-up. In manual mode, the expected frequency must be set manually.

Command Syntax

`:FC:AUTOFREQ:[mode]`

Variable	Value	Description
[mode]	0	Set manual compensation mode (expected input frequency must be specified by user)
	1	Set Automatic compensation mode (the power reading compensation is set based on the measured input frequency)

Return string

`[status]`

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:FC:AUTOFREQ:1</code>	1

HTTP Implementation: `http://10.10.10.10/:FC:AUTOFREQ:1`

See Also

[Get Compensation Frequency](#)
[Set Compensation Frequency](#)
[Get Compensation Frequency Mode](#)

5.3 (k) - Read Power

Description

Returns the input power measurement in dBm.

Command Syntax

`:POWER?`

Return string

`[power]`

Variable	Description
<code>[power]</code>	Input power measurement in dBm

Examples

string to Send	string Returned
<code>:POWER?</code>	<code>-22.05 dBm</code>

HTTP Implementation: `http://10.10.10.10/:POWER?`

See Also

[Set Compensation Frequency](#)
[Read Voltage](#)

5.3 (I) - Read Voltage

Description

Returns the raw voltage detected at the sensor head. There is no calibration for temperature or frequency.

Command Syntax

`:VOLTAGE?`

Return string

`[volts]`

Variable	Description
<code>[volts]</code>	Input voltage reading in mV.

Examples

string to Send	string Returned
<code>:VOLTAGE?</code>	0.000105 Volt

HTTP Implementation: `http://10.10.10.10/:VOLTAGE?`

See Also

[Read Power](#)

5.3 (m) - Get LCD Offset Mode

Description

Indicates whether an offset value (in dB) should be included in power measurements displayed on the LCD.

Command Syntax

`:OFFSETVALUE:ENABLE?`

Return string

`[mode]`

Variable	Value	Description
<code>[mode]</code>	0	No offset applied to power measurements
	1	Offset value will be applied to power measurements

Examples

string to Send	string Returned
<code>:OFFSETVALUE:ENABLE?</code>	1

HTTP Implementation: <http://10.10.10.10/:OFFSETVALUE:ENABLE?>

See Also

[Set LCD Offset Mode](#)
[Get LCD Offset Value](#)
[Set LCD Offset Value](#)

5.3 (n) - Set LCD Offset Mode

Description

Sets whether an offset value (in dB) should be included in power measurements displayed on the LCD.

Command Syntax

`:OFFSETVALUE:ENABLE:[mode]`

Variable	Value	Description
<code>[mode]</code>	0	No offset applied to power measurements
	1	Offset value will be applied to power measurements

Return string

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:OFFSETVALUE:ENABLE:1</code>	1

HTTP Implementation: `http://10.10.10.10/:OFFSETVALUE:ENABLE:1`

See Also

[Get LCD Offset Mode](#)
[Get LCD Offset Value](#)
[Set LCD Offset Value](#)

5.3 (o) - Get LCD Offset Value

Description

Returns the value in dB to be used as an offset from power measurements displayed on the LCD. Only applied when offset mode is enabled.

Command Syntax

`:OFFSETVALUE:VALUE?`

Return string

`[value]`

Variable	Description
<code>[value]</code>	The offset value in dB

Examples

string to Send	string Returned
<code>:OFFSETVALUE:VALUE?</code>	<code>10.25</code>

HTTP Implementation: <http://10.10.10.10/:OFFSETVALUE:VALUE?>

See Also

[Get LCD Offset Mode](#)
[Set LCD Offset Mode](#)
[Set LCD Offset Value](#)

5.3 (p) - Set LCD Offset Value

Description

Sets a value in dB to be used as an offset from power measurements displayed on the LCD. Only applied when offset mode is enabled.

Command Syntax

`:OFFSETVALUE:VALUE:[value]`

Variable	Description
[value]	The offset value in dB

Return string

`[status]`

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:OFFSETVALUE:VALUE:10.25</code>	1

HTTP Implementation: `http://10.10.10.10/:OFFSETVALUE:VALUE:10.25`

See Also

[Get LCD Offset Mode](#)
[Set LCD Offset Mode](#)
[Get LCD Offset Value](#)

5.4 - SCPI - Frequency Measurement Commands

5.4 (a) - Get Range

Description

Returns the actual frequency measurement range of the sensor when in automatic range mode.

Command Syntax

`:FC:RANGE?`

Return string

`RANGE: [range]`

Variable	Value	Description
[range]	0	No input signal detected
	1	Mode 1; for input frequencies from 1 to 40 MHz
	2	Mode 2; for input frequencies from 40 to 190 MHz
	3	Mode 3; for input frequencies from 190 to 1400 MHz
	4	Mode 4; for input frequencies from 1400 to 6000 MHz.

Examples

string to Send	string Returned
<code>:FC:RANGE?</code>	<code>RANGE:1</code>

HTTP Implementation: `http://10.10.10.10/:FC:RANGE?`

See Also

[Get Requested Range](#)

[Set Range](#)

5.4 (b) - Get Requested Range

Description

Returns the user requested frequency measurement range.

Command Syntax

`:FC:RRANGE?`

Return string

`RANGE: [range]`

Variable	Value	Description
[range]	AUTO	Automatic mode; the counter will select the appropriate measurement range
	1	Mode 1; for input frequencies from 1 to 40 MHz
	2	Mode 2; for input frequencies from 40 to 190 MHz
	3	Mode 3; for input frequencies from 190 to 1400 MHz
	4	Mode 4; for input frequencies from 1400 to 6000 MHz

Examples

string to Send	string Returned
<code>:FC:RRANGE?</code>	<code>RANGE: 3</code>

HTTP Implementation: <http://10.10.10.10/:FC:RRANGE?>

See Also

[Get Range](#)
[Set Range](#)

5.4 (c) - Set Range

Description

Sets the frequency measurement range of the sensor. 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.

Command Syntax

:FC:RANGE: [range]

Variable	Value	Description
[range]	AUTO	Automatic mode; the counter will select the appropriate measurement range
	1	Mode 1; for input frequencies from 1 to 40 MHz
	2	Mode 2; for input frequencies from 40 to 190 MHz
	3	Mode 3; for input frequencies from 190 to 1400 MHz
	4	Mode 4; for input frequencies from 1400 to 6000 MHz

Return string

[status]

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:FC:RANGE:AUTO</code>	1
<code>:FC:RANGE:3</code>	1

HTTP Implementation: `http://10.10.10.10/:FC:RANGE:AUTO`

See Also

[Get Range](#)

[Get Requested Range](#)

5.4 (d) - Get Sample Time

Description

Returns the time in milliseconds over which the input frequency will be sampled.

Command Syntax

`:FC:SAMPLETIME?`

Return string

`[time]`

Variable	Description
<code>[time]</code>	The frequency sample time in milliseconds

Examples

string to Send	string Returned
<code>:FC:SAMPLETIME?</code>	1000

HTTP Implementation: <http://10.10.10.10/:FC:SAMPLETIME?>

See Also

[Set Sample Time](#)

5.4 (e) - Set Sample Time

Description

Sets the sample time to be used for frequency measurements, from 100 to 3000 ms, in 100 ms steps. The default sample time is 1000 ms (1 second).

Command Syntax

`:FC:SAMPLETIME:[time]`

Return string

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

Examples

string to Send	string Returned
<code>:FC:SAMPLETIME:300</code>	1

HTTP Implementation: `http://10.10.10.10/:FC:SAMPLETIME:300`

See Also

[Get Sample Time](#)

5.4 (f) - Get Frequency

Description

Returns the measured input frequency in MHz.

Command Syntax

`:FC:FREQ?`

Return string

`[freq]`

Variable	Description
<code>[freq]</code>	The measured frequency in MHz or "Unknown" if no measureable input signal is detected

Examples

string to Send	string Returned
<code>:FC:FREQ?</code>	Unknown
<code>:FC:FREQ?</code>	1000.0000 MHz

HTTP Implementation: `http://10.10.10.10/:FC:FREQ?`

See Also

[Read Power](#)
[Set Compensation Frequency Mode](#)
[Get Frequency & Power](#)

5.4 (g) - Get Frequency & Power

Description

Measures the frequency (MHz) and power level (dBm) of the input signal.

Command Syntax

```
:FC:FREQ?POWER?
```

Return string

```
[freq] AND [power]
```

Variable	Description
[freq]	The measured frequency in MHz
[power]	The measured power level in dBm

Examples

string to Send	string Returned
:FC:FREQ?POWER?	1000 MHz AND 10.5 dBm

HTTP Implementation: <http://10.10.10.10/:FC:FREQ?POWER?>

See Also

[Read Power](#)
[Get Frequency](#)

5.4 (h) - Get Reference Mode

Description

Indicates whether the sensor is using the internal reference for frequency measurements or an external source. The reference source will automatically switch to external if a suitable signal is detected at the Ref In port.

Command Syntax

`:FC:REF?`

Return string

`[ref]`

Variable	Value	Description
<code>[ref]</code>	IntRef	Internal reference in use
	ExtRef	External reference detected and in use

Examples

string to Send	string Returned
<code>:FC:REF?</code>	<code>ExtRef</code>

HTTP Implementation: `http://10.10.10.10/:FC:REF?`