PROGRAMMING MANUAL

Power Sensors

PWR Series

testsolutions@minicircuits.com | www.minicircuits.com

Contents

1. Overview

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' USB and Ethernet controlled power sensors. The contents apply to:

- PWR series CW power sensors
- PWR series RMS power sensors
- PWR series peak & average power sensors

The full software and documentation package including a GUI program, DLL files, user guide and programming examples is available for download from the Mini-Circuits website at:

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

For details and specifications of individual models please see:

https://www.minicircuits.com/WebStore/RF-Smart-Power-Sensors.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

1.1. Control Methods

Communication with the device can use any of the following methods:

1. For Ethernet connected models, using HTTP or Telnet communication over an Ethernet TCP / IP connection (see Ethernet Control API), which is largely independent of the operating system.
2. Using the provided API DLL files (.Net or ActiveX COM objects) for USB control on Microsoft Windows operating systems (see USB Control API for Microsoft Windows)
3. Using USB interrupt codes for direct programming on Linux operating systems (see USB Control via Direct Programming (Linux))

In all cases the full functionality of the system is accessible using a command set based on SCPI (see SCPI Commands for Power Sensor Control).

1.2. Programming Examples

Mini-Circuits provides examples for a range of programming environments and connection methods, these can be downloaded from our website at:

https://www.minicircuits.com/WebStore/pte_example_download.html

Mini-Circuits' Ethernet & USB controlled devices are designed to implement similar control interfaces, so it is usually the case that an example written for one product family can be adapted easily for use with another.

Please contact Mini-Circuit's application support team if an example is not available for the environment of interest.

1.3. Support Contacts

We are here to support you every step of the way. For technical support and assistance, please contact us at the email address below or refer to our website for your local support:

testsolutions@minicircuits.com

www.minicircuits.com/contact/worldwide_tech_support.html

2. USB Control API for Microsoft Windows

Mini-Circuits' API for USB control from a computer running Microsoft Windows is provided in the form of a DLL file. 3 DLL options are provided to offer the widest possible support, with the same functionality in each case.

1) .Net Framework 4.5 DLL - This is the recommended API for most modern operating systems
2) .Net Framework 2.0 DLL - Provided for legacy support of older computers / operating systems, with an installed version of the .Net framework prior to 4.5
3) ActiveX com object - Provided for legacy support of older environments which do not support .Net

The latest version of each DLL file can be downloaded from the Mini-Circuits website at:

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

2.1. DLL API Options

2.1.1. .NET FRAMEWORK 4.5 DLL (RECOMMENDED)

The recommended API option for USB control from most modern programming environments running on Windows.

Filename: mcl_pm_NET45.dll

Requirements

1) Microsoft Windows with .Net framework 4.5 or later
2) Programming environment with ability to support .Net components

Installation

1) Download the latest DLL file from the Mini-Circuits website
2) Copy the .dll file to the preferred directory (the recommendation is to use the same folder as the programming project, or `C:\WINDOWS\SysWOW64`
3) Right-click on the DLL file in the save location and select Properties to check that Windows has not blocked access to the file (check "Unblock" if the option is shown)
4) No registration or further installation action is required

2.1.2. .NET FRAMEWORK 2.0 DLL (LEGACY SUPPORT)

Provided for support of systems with an older version of the .Net framework installed (prior to 4.5).

Filename: mcl_pm64.dll

Requirements

1) Microsoft Windows with .Net framework 2.0 or later
2) Programming environment with ability to support .Net components

Installation

1) Download the latest DLL file from the Mini-Circuits website:
2) Copy the .dll file to the preferred directory (the recommendation is to use the same folder as the programming project, or `C:\WINDOWS\SysWOW64`
3) Right-click on the DLL file in the save location and select Properties to check that Windows has not blocked access to the file (check "Unblock" if the option is shown)
4) No registration or further installation action is required

2.1.3. ACTIVEX COM OBJECT DLL (LEGACY SUPPORT)

Provided for support of programming environments which do not support .Net components.

Filename: mcl_pm.dll

Requirements

1) Microsoft Windows
2) Programming environment with support for ActiveX components

Installation

1. Copy the DLL file to the correct directory:
   For 32-bit Windows operating systems: `C:\WINDOWS\System32`
   For 64-bit Windows operating systems: `C:\WINDOWS\SysWOW64`
2. Open the Command Prompt in "Elevated" mode:
   a. Open the Start Menu/Start Screen and type "Command Prompt"
   b. Right-click on the shortcut for the Command Prompt
   c. Select "Run as Administrator"
   d. 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:
   a. 32-bit PC: `\WINDOWS\System32\Regsvr32 \WINDOWS\System32\mcl_pm.dll`
   b. 64-bit PC: `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mcl_pm.dll`
4. Hit enter to confirm and a message box will appear to advise of successful registration.

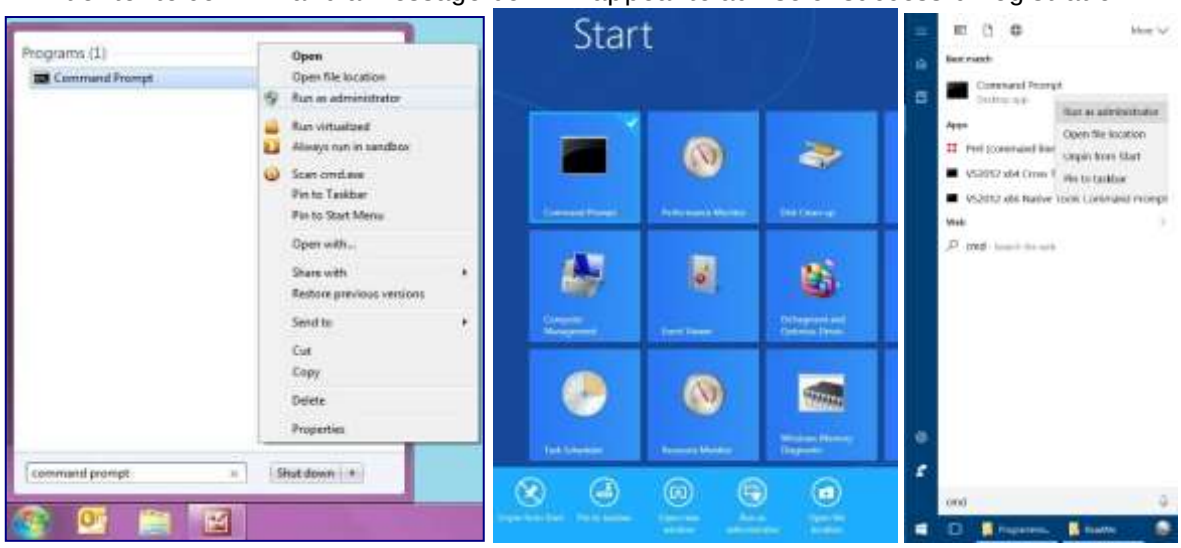


Fig 2.1-a: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

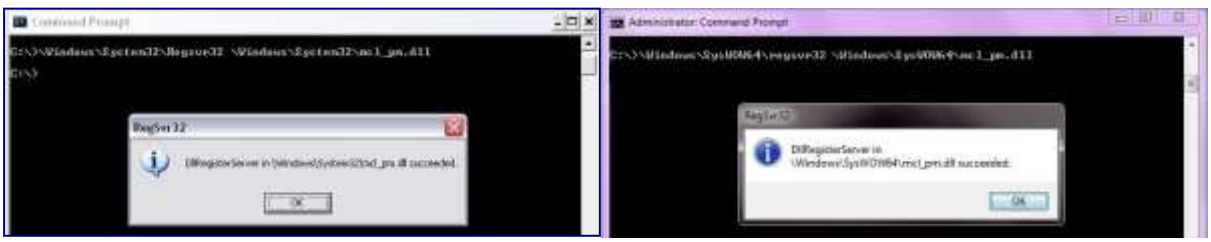


Fig 2.1-b: Registering the DLL in a 64-bit environment

2.2. Referencing the DLL

Most programming environments require a reference to be set to the relevant DLL file, either in the IDE menu or within the program. Multiple instances of the DLL control class can then be created (referred to as MyPTE1 and MyPTE2 below) in order to connect to as many devices as needed

Example Declarations Using the .NET 4.5 DLL (mcl_pm_NET45.dll)

(For operation with the .Net 2.0 DLL, replace "mcl_pm_NET45" with "mcl_ pm64")

Python	`import clr  # Import the pythonnet CLR library`
	`clr.AddReference('mcl_pm_NET45')`
	`from mcl_pm_NET45 import usb_pm`
	`MyPTE1 = usb_pm()`
	`MyPTE2 = usb_pm()`

Visual Basic	`Public MyPTE1 As New mcl_pm_NET45.usb_pm` `Public MyPTE2 As New mcl_pm_NET45.usb_pm`
Visual C++	`mcl_pm_NET45::usb_pm ^MyPTE1 = gcnew mcl_pm_NET45::usb_pm();` `mcl_pm_NET45::usb_pm ^MyPTE2 = gcnew mcl_pm_NET45::usb_pm();`
Visual C#	`mcl_pm_NET45.usb_pm MyPTE1 = new mcl_pm_NET45.usb_pm();` `mcl_pm_NET45.usb_pm MyPTE2 = new mcl_pm_NET45.usb_pm();`

MatLab	`MCL_ATT = NET.addAssembly('C:\Windows\SysWOW64\mcl_pm_NET45.dll')`
	`MyPTE1 = mcl_pm_NET45.usb_pm`
	`MyPTE2 = mcl_pm_NET45.usb_pm`

Example Declarations using the ActiveX DLL (mcl_pm.dll)

Visual Basic	`Public MyPTE1 As New mcl_pm.USB_PM` `Public MyPTE2 As New mcl_pm.USB_PM`
Visual C++	`mcl_pm::USB_pm ^MyPTE1 = gcnew mcl_pm::USB_pm();` `mcl_pm::USB_pm ^MyPTE2 = gcnew mcl_pm::USB_pm();`
Visual C#	`public mcl_pm.USB_PM MyPTE1 = new mcl_pm.USB_PM();` `public mcl_pm.USB_PM MyPTE2 = new mcl_pm.USB_PM();`
MatLab	`MyPTE1 = actxserver(mcl_pm.USB_PM)` `MyPTE2 = actxserver(mcl_pm.USB_PM)`

2.3. Additional DLL Considerations

Mini-Circuits' DLL API options are intended to support the widest possible range of modern programming environments, with the typical summaries and examples below applying in most cases. There are a few additional considerations to bear in mind for specific programming manuals, as summarized below.

2.3.1. MINI-CIRCUITS' DLL USE IN PYTHON / MATLAB

Some functions are defined within Mini-Circuits' DLL files with arguments to be passed by reference. This allows the variables (with their updated values) to be used later within the program, after the DLL function has executed. This methodology fits with many programming environments (including C#, C++ and VB) but is interpreted slightly differently by Python and MatLab:

- Typical operation (C#, C++, VB):
  - The function has an integer return value to indicate success / failure (1 or 0)
  - One or more variables are passed to the function by reference so that the updated values are available to the program once function execution has completed
- Python implementation:
  - Any parameters passed by reference to a function can be ignored (an empty string can be provided in place of the variable)
  - The return value from the function will change from the single integer value as defined in this manual, to a tuple
  - The tuple format will be [function_return_value, function_parameter]
- MatLab implementation:
  - Any parameters passed by reference to a function can be ignored (an empty string can be provided in place of the variable)
  - The return value from the function will change from the single integer value as defined in this manual to an array of values
  - The function must be assigned to an array variable of the correct size, in the format [function_return_value, function_parameter]

The examples below illustrate how a function of this type is defined in the DLL and how that same function is implemented in C#, Python and MatLab.

Definition	**Short Send_SCPI(String SndSTR, ByRef String RetSTR)**

Visual C#	`status = MyPTE1.Send_SCPI(":SN?", ref(RetSTR));`
	`if(status > 0)`
	`{`
	`  MessageBox.Show("The connected device is " + RetSTR);`
	`}`

Python	`status = MyPTE1.Send_SCPI(":SN?", "")`
	`if status[0] > 0:`
	`  RetSTR = str(status[1])`
	`  print('The connected device is ', RetSTR)`

MatLab	`[status, RetSTR] = MyPTE1.Send_SCPI(':SN?', '')`
	`if status > 0`
	`  h = msgbox('The connected device is ', RetSTR)`
	`end`

2.3.2. MINI-CIRCUITS' DLL USE IN LABWINDOWS / CVI

It is recommended to use one of the .Net DLL files for USB control of Mini-Circuits devices from CVI. The initial steps to set up the instrument driver in the CVI project are as below (note: there may be slight variations between CVI versions):

1. It is recommended to place the DLL into the CVI project folder (refer to the instructions above, to download the .Net DLL and ensure that Windows has not blocked it)
2. Open CVI:
    a. From the menu select Tools > Create .NET controller
    b. Check the option to specify the assembly by path and filename
    c. Browse to the working directory and select the DLL file
    d. Under the target instrument enter the working directory path
    e. CVI should now compile and create the instrument driver (.fp) file
    f. Under Instrument files on the left of the CVI screen there should now be an object for the DLL file
    g. Clicking on the object provides access to all methods and properties within the DLL.

The process above creates an instrument driver in CVI which has the effect of a "wrapper" around the Mini-Circuits DLL. This "wrapper" provides a set of definitions for each of the DLL functions which allow them to be used within CVI. The new definitions contain some additional CVI specific content which make them appear slightly different to the DLL definitions summarized in this manual, but the arguments and return values remain the same.

The example below demonstrates how the DLL definitions in this manual convert to the form used within the CVI "wrapper":

Mini-Circuits' DLL Definition	`short Open_Sensor(string [SN_Request])`

Implementation in CVI instrument driver	`int CVIFUNC mcl_pm64_usb_pm_Open_Sensor(`
	`  mcl_pm64_usb_pm __instance,`
	`  char ** SN_Request,`
	`  short * __returnValue,`
	`  CDotNetHandle * __exception);`

Explanation	The CVI function definition contains the following arguments: 1. An instance of the Mini-Circuits DLL class 2. The argument(s) defined in the Mini-Circuits DLL function 3. The return value from the Mini-Circuits DLL function 4. An error indicator object (part of the CVI / .Net instrument driver)

2.4. DLL – Properties

Function	Syntax
Compensation Frequency	`double Freq`
Averaging Mode	`short AVG`
Average Count	`short AvgCount`
Power Format	`bool Format_mw`
Offset Value	`single OffsetValue`
Offset Mode	`short OffsetValue_Enable`

2.4.1. COMPENSATION FREQUENCY

double Freq

Sets the power sensor frequency compensation to the correct frequency in MHz for the expected input signal. This parameter needs to be set in order to achieve the specified power measurement accuracy.

Note: PWR series power sensors do not have frequency selectivity.

Parameters

Value	Description
Frequency	Frequency value (MHz) within the power sensor's specified input range

Examples

Python	`MyPTE1.Freq = 1000` `Frequency = MyPTE1.Freq`
Visual Basic	`MyPTE1.Freq = 1000` `Frequency = MyPTE1.Freq`
Visual C++	`MyPTE1->Freq = 1000;` `Frequency = MyPTE1->Freq;`
Visual C#	`MyPTE1.Freq = 1000;` `Frequency = MyPTE1.Freq;`
MatLab	`MyPTE1.Freq = 1000;` `Frequency = MyPTE1.Freq;`

2.4.2. AVERAGING MODE

short AVG

Enables the "averaging" mode of the power 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).

Parameters

Value	Description
0	Disable averaging mode
1	Enable averaging mode

Examples

Python	`MyPTE1.Avg = 1` `Avg_on = MyTPE1.Avg`
Visual Basic	`MyPTE1.Avg = 1` `Avg_on = MyTPE1.Avg`
Visual C++	`MyPTE1->Avg = 1;` `Avg_on = MyTPE1->Avg;`
Visual C#	`MyPTE1.Avg = 1;` `Avg_on = MyTPE1.Avg;`
MatLab	`MyPTE1.Avg = 1;` `Avg_on = MyTPE1.Avg;`

2.4.3. AVERAGE COUNT

short AvgCount

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).

Parameters

Value	Description
Count	The number of measurements to average (1 to 16)

Examples

Python	`MyPTE1.AvgCount = 10` `Count = MyPTE1.AvgCount`
Visual Basic	`MyPTE1.AvgCount = 10` `Count = MyPTE1.AvgCount`
Visual C++	`MyPTE1->AvgCount = 10;` `Count = MyPTE1->AvgCount;`
Visual C#	`MyPTE1.AvgCount = 10;` `Count = MyPTE1.AvgCount;`
MatLab	`MyPTE1.AvgCount = 10;` `Count = MyPTE1.AvgCount;`

2.4.4. POWER FORMAT

bool Format_mW

Sets the power measurement units to between mW and dBm. The default is power measurements in dBm.

Parameters

Value	Description
False	Power reading in dBm
True	Power reading in mW

Examples

Python	`MyPTE1.Format_mW = True` `Format = MyPTE1.Format_mw`
Visual Basic	`MyPTE1.Format_mW = True` `Format = MyPTE1.Format_mw`
Visual C++	`MyPTE1->Format_mW = True;` `Format = MyPTE1->Format_mw;`
Visual C#	`MyPTE1.Format_mW = True;` `Format = MyPTE1.Format_mw;`
MatLab	`MyPTE1.Format_mW = True;` `Format = MyPTE1.Format_mw;`

2.4.5. OFFSET VALUE

single OffsetValue

Sets a single offset value to be used for power readings. The power meter offset mode must be set to "1" (single value).

Parameters

Value	Description
Offset	The power measurement offset in dB

Examples

Python	`MyPTE1.OffsetValue_Enable = 1` `MyPTE1.OffsetValue = 5.4`
Visual Basic	`MyPTE1.OffsetValue_Enable = 1` `MyPTE1.OffsetValue = 5.4`
Visual C++	`MyPTE1->OffsetValue_Enable = 1;` `MyPTE1->OffsetValue = 5.4;`
Visual C#	`MyPTE1.OffsetValue_Enable = 1;` `MyPTE1.OffsetValue = 5.4;`
MatLab	`MyPTE1.OffsetValue_Enable = 1;` `MyPTE1.OffsetValue = 5.4;`

See Also

Offset Mode

2.4.6. OFFSET MODE

short OffsetValue_Enable

Defines whether an offset is used for the power readings. The power 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).

Parameters

Value	Description
0	Offset disabled
1	Use single value offset
2	Use array of offset values

Examples

Python	`MyPTE1.OffsetValue_Enable = 1` `MyPTE1.OffsetValue = 5.4`
Visual Basic	`MyPTE1.OffsetValue_Enable = 1` `MyPTE1.OffsetValue = 5.4`
Visual C++	`MyPTE1->OffsetValue_Enable = 1;` `MyPTE1->OffsetValue = 5.4;`
Visual C#	`MyPTE1.OffsetValue_Enable = 1;` `MyPTE1.OffsetValue = 5.4;`
MatLab	`MyPTE1.OffsetValue_Enable = 1;` `MyPTE1.OffsetValue = 5.4;`

See Also

Offset Value

Set Offset Values

2.5. DLL - General Functions

Function	Syntax
Open Power Sensor Connection	`short Open_Sensor(Optional ByRef string SN_Request)`
Close Power Sensor Connection	`void Close_Sensor()`
Read Model Name of Power Sensor	`string GetSensorModelName()`
Read Serial Number of Power Sensor	`string GetSensorSN()`
Get List of Connected Serial Numbers	`short Get_Available_SN_List(ByRef string SN_List)`
Get Status	`short GetStatus()`
Check Connection	`short Check_Connection()`
Get Temperature of Power Sensor	`float GetDeviceTemperature(Optional ByRef string Temp_Unit)`
Get Firmware	`short GetFirmwareInfo(ByRef short FirmwareID, ByRef string FirmwareRev, ByRef short FirmwareNo)`
Get USB Device Name	`string GetUSBDeviceName()`
Get USB Device Handle	`string GetUSBDeviceHandle()`

2.5.1. OPEN POWER SENSOR CONNECTION

short Open_Sensor(Optional ByRef string SN) (.Net)

short Open_Sensor(Optional string SN) (ActiveX)

Initializes the connection to a USB power sensor. 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

Variable	Description
SN	Optional string containing the serial number of the USB power sensor. Can be omitted if only one sensor is connected but must be included otherwise.

Return Values

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

Examples

Python	`status = MyPTE1.Open_Sensor(SN)`
Visual Basic	`status = MyPTE1.Open_Sensor(SN)`
Visual C++	`status = MyPTE1->Open_Sensor(SN);`
Visual C#	`status = MyPTE1.Open_Sensor(ref(SN));`
MatLab	`status = MyPTE1.Open_Sensor(SN);`

2.5.2. CLOSE POWER SENSOR CONNECTION

void Close_Sensor()

Closes the connection to the power sensor.

Examples

Python	`status = MyPTE1.Close_Sensor()`
Visual Basic	`status = MyPTE1.Close_Sensor()`
Visual C++	`status = MyPTE1->Close_Sensor();`
Visual C#	`status = MyPTE1.Close_Sensor();`
MatLab	`status = MyPTE1.Close_Sensor();`

2.5.3. READ MODEL NAME OF POWER SENSOR

string GetSensorModelName()

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

Return Values

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

Examples

Python	`ModelName = MyPTE1.GetSensorModelName()` `print('The connected device is ', ModelName)`
Visual Basic	`ModelName = MyPTE1.GetSensorModelName()` `MsgBox ("The connected device is " & ModelName)`
Visual C++	`ModelName = MyPTE1->GetSensorModelName();` `MessageBox::Show("The connected device is " + ModelName);`
Visual C#	`ModelName = MyPTE1.GetSensorModelName();` `MessageBox.Show("The connected device is " + ModelName);`
MatLab	`ModelName = MyPTE1.GetSensorModelName();` `h = msgbox('The connected device is ', ModelName)`

2.5.4. READ SERIAL NUMBER OF POWER SENSOR

string GetSensorSN()

Returns the serial number of the connected power sensor.

Return Values

Value	Description
SN	Serial number of the connected sensor

Examples

Python	`SerialNo = MyPTE1.GetSensorSN()` `print('The connected device is ', SerialNo)`
Visual Basic	`SerialNo = MyPTE1.GetSensorSN()` `MsgBox ("The connected device is " & SerialNo)`
Visual C++	`SerialNo = MyPTE1->GetSensorSN();` `MessageBox::Show("The connected device is " + SerialNo);`
Visual C#	`SerialNo = MyPTE1.GetSensorSN();` `MessageBox.Show("The connected device is " + SerialNo);`
MatLab	`SerialNo = MyPTE1.GetSensorSN();` `h = msgbox('The connected device is ', SerialNo)`

2.5.5. GET LIST OF CONNECTED SERIAL NUMBERS

short Get_Available_SN_List(ByRef string SN_List)

Provides a list of serial numbers for all available (currently connected) power sensors.

Parameters

Variable	Description
SN_List	String variable which the function will update with a list of all available serial numbers, separated by a single space character, for example "11508280079 11508280080 11508280081".

Return Values

Value	Description
0	Command failed
>1	Command completed successfully

Examples

Python	`status = MyPTE1.Get_Available_SN_List("")`
	`if status[0] > 0:`
	`    SN_List = str(status[1])`
	`    print("Connected devices:", SN_List)`

Visual Basic	`If MyPTE1.Get_Available_SN_List(SN_List) > 0 Then`
	`    MsgBox ("Connected devices: " & SN_List)`
	`End If`

Visual C++	`if (MyPTE1->Get_Available_SN_List(SN_List) > 0 )`
	`{`
	`    MessageBox::Show("Connected devices: " + SN_List);`
	`}`

Visual C#	`if (MyPTE1.Get_Available_SN_List(ref(SN_List)) > 0 )`
	`{`
	`    MessageBox.Show("Connected devices: " + SN_List);`
	`}`

MatLab	`[status, SN_List] = MyPTE1.Get_Available_SN_List('')`
	`if status > 0`
	`    h = msgbox('Connected devices: ', SN_List)`
	`end`

See Also

Read Serial Number of Power Sensor

Open Power Sensor Connection

2.5.6. GET STATUS

short GetStatus()

Checks whether the USB connection to the power sensor is still active.

Return Values

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

Examples

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

See Also

Open Power Sensor Connection

2.5.7. CHECK CONNECTION

short Check_Connection()

Checks whether the USB connection to the power sensor is still active.

Return Values

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

Examples

Python	`status = MyPTE1.Check_Connection()`
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

Open Power Sensor Connection

2.5.8. GET TEMPERATURE OF POWER SENSOR

float GetDeviceTemperature(Optional ByRef string Temp_Unit) (.Net)

float GetDeviceTemperature(Optional string Temp_Unit) (ActiveX)

Returns the internal temperature of the power sensor. Note: The reading provides an indication but is not calibrated.

Parameters

Variable	Value	Description
Temp_Unit	C	Return temperature in Celsius (this is the default if omitted)
	F	Return temperature in Fahrenheit

Return Values

Value	Description
Temperature	The device internal temperature in the specified units

Examples

Python	`Temp = MyPTE1. GetDeviceTemperature('C')` `print('Temperature:', Temp[0])`
Visual Basic	`Temp = MyPTE1.GetSensorModelName('C')` `MsgBox ("Temperature: " & Temp)`
Visual C++	`Temp = MyPTE1->GetSensorModelName('C');` `MessageBox::Show("Temperature:" + Temp);`

Visual C#	`Temp_Format = 'C'`
	`Temp = MyPTE1.GetSensorModelName(ref(Temp_Format));`
	`MessageBox.Show("Temperature:" + Temp);`

MatLab	`[Temp, Temp_Format] = MyPTE1.GetSensorModelName('C');` `h = msgbox(' Temperature:', Temp)`

2.5.9. GET FIRMWARE

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

Returns the internal firmware version of the power sensor.

Parameters

Variable	Description
FirmwareID	Required. String variable passed by reference (no user significance).
FirmwareRev	Required. String variable passed by reference, to be updated with the current firmware version, for example "B3".
FirmwareNo	Required. String variable passed by reference (no user significance).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

```
status = MyPTE1.GetFirmwareInfo(FirmwareID, Firmware, FirmwareNo)
if status[0] > 0:
        Firmware = str(status[2])
        print("Firmware Version:", Firmware)
```

Visual Basic

```
If MyPTE1.GetFirmwareInfo(FirmwareID, Firmware, FirmwareNo) > 0 Then
        MsgBox ("Firmware Version: " & Firmware)
End If
```

Visual C++

```
if (MyPTE1->GetFirmwareInfo(FirmwareID, Firmware, FirmwareNo) > 0)
{
        MessageBox::Show("Firmware Version: " + Firmware);
}
```

Visual C#

```
if(MyPTE1.GetFirmwareInfo(ref(FirmwareID), ref(Firmware), ref(FirmwareNo)) > 0)
{
        MessageBox.Show("Firmware Version: " + Firmware);
}
```

MatLab

```
[status, FirmwareID, Firmware, FirmwareNo] = MyPTE1.GetFirmwareInfo('', '', '')
if status > 0
        h = msgbox('Firmware Version: ', Firmware)
end
```

2.5.10. GET FIRMWARE VERSION (ANTIQUATED)

short GetFirmwareVer(ByRef short FirmwareVer)

This function is antiquated, GetFirmwareInfo should be used instead.

2.5.11. GET USB DEVICE NAME

string GetUSBDeviceName()

Returns the USB device name of the sensor for direct communication.

Return Values

Value	Description
DeviceName	Device name of the power sensor

Examples

Python	`status = MyPTE1.GetUSBDeviceName()`
Visual Basic	`status = MyPTE1.GetUSBDeviceName()`
Visual C++	`status = MyPTE1->GetUSBDeviceName();`
Visual C#	`status = MyPTE1.GetUSBDeviceName();`
MatLab	`status = MyPTE1.GetUSBDeviceName();`

2.5.12. GET USB DEVICE HANDLE

string GetUSBDeviceHandle()

Returns the handle to the USB sensor for direct communication.

Return Values

Value	Description
HandleToUSB	USB handle of the power sensor head

Examples

Python	`status = MyPTE1.GetUSBDeviceHandle()`
Visual Basic	`status = MyPTE1.GetUSBDeviceHandle()`
Visual C++	`status = MyPTE1->GetUSBDeviceHandle();`
Visual C#	`status = MyPTE1.GetUSBDeviceHandle();`
MatLab	`status = MyPTE1.GetUSBDeviceHandle();`

2.5.13. OPEN ANY POWER SENSOR (ANTIQUATED)

short Open_AnySensor()

This function should not be used and is included only for compatibility with early models, Open_Sensor is the recommended method to connect to a power sensor.

2.5.14. OPEN ANY POWER SENSOR (ANTIQUATED)

void Init_PM()

This function should not be used and is included only for compatibility with early models, Open_Sensor is the recommended method to connect to a power sensor.

2.5.15. CLOSE POWER SENSOR CONNECTION (ANTIQUATED)

void CloseConnection()

This function should not be used and is included only for compatibility with early models, Close_Sensor is the recommended method to disconnect from a power sensor.

2.6. DLL – Measuring with Average Power Sensors

These functions apply to the following Mini-Circuits' power sensor models:

- PWR-xGHS Series (CW average power sensors)
- PWR-xFS Series (fast sampling CW average power sensors)
- PWR-xRMS Series (true RMS power sensors)

Function	Syntax
Set Measurement Mode	`void SetFasterMode(ByRef short S_A)`
Set Power Range	`void SetRange(short Range)`
Read Power	`float ReadPower()`
Read Immediate Power	`float ReadImmediatePower()`
Read Voltage	`float ReadVoltage()`
Get Offset Values	`short GetOffsetValues(ByRef int NoOfPoints, ByRef double FreqArray(), ByRef single LossArray())`
Set Offset Values	`int SetOffsetValues(int NoOfPoints, ByRef double FreqArray(), ByRef single LossArray())`

2.6.1. SET MEASUREMENT MODE

void SetFasterMode(ByRef short S_A) *(.Net)*

void SetFasterMode(short S_A) *(ActiveX)*

Sets the measurement mode of the power sensor between "low noise" and "fast sampling" modes. An additional "fastest sampling" mode is also available for PWR-8FS. The specifications for these modes are defined in the individual model datasheets. The default is "low noise" mode.

Parameters

Variable	Value	Description
S_A	0	Low noise mode (default)
	1	Fast sampling mode
	2	Fastest sampling mode (only available for PWR-8FS)

Examples

Python	`status = MyPTE1.SetFasterMode(S_A)`
Visual Basic	`status = MyPTE1.SetFasterMode(S_A)`
Visual C++	`status = MyPTE1->SetFasterMode(S_A);`
Visual C#	`status = MyPTE1.SetFasterMode(ref(S_A));`
MatLab	`status = MyPTE1.SetFasterMode(S_A);`

2.6.2. SET POWER RANGE

void SetRange(short Range)

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

Parameters

Variable	Value	Description
Range	0	Auto
	1	Low power
	2	High power

Examples

Python	`status = MyPTE1.SetRange(Range)`
Visual Basic	`status = MyPTE1.SetRange(Range)`
Visual C++	`status = MyPTE1->SetRange(Range);`
Visual C#	`status = MyPTE1.SetRange(Range);`
MatLab	`status = MyPTE1.SetRange(Range);`

2.6.3. READ POWER

float ReadPower()

Returns the sensor power measurement.

Return Values

Value	Description
Power	The power reading in either mW or dBm. Note: A power value below -900 dBm indicates that the input signal level is below the sensor's useable range.

Examples

Python	`status = MyPTE1.ReadPower`
Visual Basic	`status = MyPTE1.ReadPower()`
Visual C++	`status = MyPTE1->ReadPower();`
Visual C#	`status = MyPTE1.ReadPower();`
MatLab	`status = MyPTE1.ReadPower();`

See Also

Power Format

2.6.4. READ IMMEDIATE POWER

float ReadImmediatePower()

Returns the sensor power measurement with a faster response but reduced accuracy compared to ReadPower. This function does not account for the sensor's internal temperature so compensation is based on the last recorded reading (taken when the ReadPower or GetDeviceTemperature functions were last called).

Return Values

Value	Description
Power	Current power measurement

Examples

Python	`status = MyPTE1.ReadImmediatePower`
Visual Basic	`status = MyPTE1.ReadImmediatePower()`
Visual C++	`status = MyPTE1->ReadImmediatePower();`
Visual C#	`status = MyPTE1.ReadImmediatePower();`
MatLab	`status = MyPTE1.ReadImmediatePower();`

See Also

Power Format

ReadPower

2.6.5. READ VOLTAGE

float ReadVoltage()

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

Return Values

Value	Description
Voltage	Voltage detected at the sensor head

Examples

Python	`status = MyPTE1.ReadVoltage`
Visual Basic	`status = MyPTE1.ReadVoltage()`
Visual C++	`status = MyPTE1->ReadVoltage();`
Visual C#	`status = MyPTE1.ReadVoltage();`
MatLab	`status = MyPTE1.ReadVoltage();`

See Also

ReadPower

2.6.6. GET OFFSET VALUES

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

Returns the offset array values used which will be applied in "array offset" mode.

Parameters

Variable	Description
NoOfPoints	Integer passed by reference, to be updated with the number of offset points set
FreqArray	Array passed by reference, to be updated with the frequency offset values (MHz)
LossArray	Array passed by reference, to be updated with the corresponding loss values (dB) for each frequency point

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`response = MyPTE1.GetOffsetValues(pts, freq, loss)`
	`pts = response[1]`
	`freq = response[2]`
	`loss = response[3]`
	`for i in range(pts):`
	`  print(str(i), str(freq[i]), str(loss[i]))`

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), ref(freq), ref(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

Offset Mode

2.6.7. SET OFFSET VALUES

short SetOffsetValues(int NoOfPoints, ByRef double FreqArray(), ByRef single LossArray()) *(.Net)*

short SetOffsetValues(int NoOfPoints, double FreqArray(), single LossArray()) *(ActiveX)*

Sets the array of offset values which will be applied for "array offset" mode.

Parameters

Variable	Description
NoOfPoints	The number of offset points to be defined in the array
FreqArray	Array of frequency (MHz) values for the offset points
LossArray	Array of loss values (dB) values for the offset points.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Offset Mode

2.7. DLL - Measuring with Peak & Average Power Sensors

These functions apply to Mini-Circuits' PWR-xP series peak & average power sensor models.

Function	Syntax
Set Sample Time	`short PeakPS_SetSampleTime(long ST)`
Get Sample Time	`long PeakPS_GetSampleTime()`
Set Trigger Mode	`short PeakPS_SetTriggerMode(int TM)`
Get Trigger Mode	`short PeakPS_GetTriggerMode()`
Read Average Power	`float PeakPS_GetAvgPower()`
Read Peak Power	`float PeakPS_GetPeakPower()`
Read Peak & Average Power Array	`short PeakPS_GetPower(int NoOfPoints, float PowerArray(), float PeakPower)`
Send SCPI Command	`short  Send_SCPI(ByRef string SndSTR, ByRef string RetSTR)`

2.7.1. SET SAMPLE TIME

short PeakPS_SetSampleTime(long ST)

Sets the time period to be captured by the power sensor, from 10 µs to 1 s.

Parameters

Variable	Description
ST	Sample time (µs), from 10 to 1,000,000 µs

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.PeakPS_SetSampleTime(100)`
Visual Basic	`status = MyPTE1.PeakPS_SetSampleTime(100)`
Visual C++	`status = MyPTE1->PeakPS_SetSampleTime(100);`
Visual C#	`status = MyPTE1.PeakPS_SetSampleTime(100);`
MatLab	`status = MyPTE1.PeakPS_SetSampleTime(100);`

2.7.2. GET SAMPLE TIME

long PeakPS_GetSampleTime()

Returns the time period to be captured by the power sensor, from 10 µs to 1 s.

Return Values

Variable	Description
ST	Sample time (µs), from 10 to 1,000,000 µs

Examples

Python	`time = MyPTE1.PeakPS_GetSampleTime()`
Visual Basic	`time = MyPTE1.PeakPS_GetSampleTime`
Visual C++	`time = MyPTE1->PeakPS_GetSampleTime();`
Visual C#	`time = MyPTE1.PeakPS_GetSampleTime();`
MatLab	`time = MyPTE1.PeakPS_GetSampleTime();`

2.7.3. SET TRIGGER MODE

short PeakPS_SetTriggerMode(int TM)

Sets the event which triggers the start of the power sensor's sample period.

Parameters

Variable	Value	Description
TM	0	No trigger, power sampling will start on request
	1	Internal trigger, power sampling will start on the rising edge of the first pulse detected at the RF input. The read power activity will timeout after 0.5s in internal trigger mode and return the last power measurement if no trigger is detected. To capture pulse sequences with longer periods the read power command can be repeated, or the external trigger mode can be used.
	2	External trigger, power sampling will start when an external trigger input signal is detected

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.PeakPS_SetTriggerMode(1)`
Visual Basic	`status = MyPTE1.PeakPS_SetTriggerMode(1)`
Visual C++	`status = MyPTE1->PeakPS_SetTriggerMode(1);`
Visual C#	`status = MyPTE1.PeakPS_SetTriggerMode(1);`
MatLab	`status = MyPTE1.PeakPS_SetTriggerMode(1);`

2.7.4. GET TRIGGER MODE

short PeakPS_GetTriggerMode()

Indicates the event which triggers the start of the power sensor's sample period.

Return Values

Value	Description
0	No trigger, power sampling will start on request
1	Internal trigger, power sampling will start on the rising edge of the first pulse detected at the RF input
2	External trigger, power sampling will start when an external trigger input signal is detected

Examples

Python	`mode = MyPTE1.PeakPS_GetTriggerMode()`
Visual Basic	`mode = MyPTE1.PeakPS_GetTriggerMode`
Visual C++	`mode = MyPTE1->PeakPS_GetTriggerMode();`
Visual C#	`mode = MyPTE1.PeakPS_GetTriggerMode();`
MatLab	`mode = MyPTE1.PeakPS_GetTriggerMode();`

2.7.5. READ AVERAGE POWER

float PeakPS_GetAvgPower()

Returns the average power measurement in dBm for the complete sample period of the sensor. The compensation frequency must be set prior to reading power in order to achieve the specified accuracy.

Return Values

Value	Description
Power	Average power of the sampled signal

Examples

Python	`power = MyPTE1.PeakPS_GetAvgPower()`
Visual Basic	`power = MyPTE1.PeakPS_GetAvgPower`
Visual C++	`power = MyPTE1->PeakPS_GetAvgPower();`
Visual C#	`power = MyPTE1.PeakPS_GetAvgPower();`
MatLab	`power = MyPTE1.PeakPS_GetAvgPower();`

See Also

Compensation Frequency

2.7.6. READ PEAK POWER

float PeakPS_GetPeakPower()

Returns the peak power measurement in dBm for the complete sample period of the sensor. The compensation frequency must be set prior to reading power in order to achieve the specified accuracy.

Note: Download the latest DLL version from the Mini-Circuits website to access the SampleTime and TriggerDelay parameters.

Return Values

Value	Description
Power	Peak power of the sampled signal

Examples

Python	`power = MyPTE1.PeakPS_GetPeakPower()`
Visual Basic	`power = MyPTE1.PeakPS_GetPeakPower`
Visual C++	`power = MyPTE1->PeakPS_GetPeakPower();`
Visual C#	`power = MyPTE1.PeakPS_GetPeakPower();`
MatLab	`power = MyPTE1.PeakPS_GetPeakPower();`

See Also

Compensation Frequency

Read Peak & Average Power Array

2.7.7. READ PEAK & AVERAGE POWER ARRAY

short PeakPS_GetPower(ByRef int NoOfPoints, ByRef float PowerArray(), ByRef float PeakPower, long SampleTime, long TriggerDelay)

Captures a series of power measurements over the sensor's sample time to enable statistical analysis of the sampled signal. The number of discrete measurements taken is variable but approximately equally spaced in the time domain so that the total sample time / number of measurements = approximate time per measurement. The series of power measurements is returned as an array.

Note: Download the latest DLL version from the Mini-Circuits website to access the SampleTime and TriggerDelay parameters.

Parameters

Variable	Description
NoOfPoints	Integer variable passed by reference, to be updated with the number of power measurements taken (the array size of PowerArray)
PowerArray()	Float array passed by reference, to be updated with the array of discrete power measurements (dBm), equally spaced over the sensor's sample time
PeakPower	Float variable passed by reference, to be updated with the peak power (dBm) detected during the sensor's sample time
SampleTime	Integer value to specify the sample time (µs) to be captured by the power sensor, from 10 to 1,000,000 µs
TriggerDelay	Integer value to specify the delay time in microseconds (µS) to be applied between detection of a trigger signal and the start of power sampling

Return Values

Value	Description
Power	Peak power of the sampled signal

Examples

Python	response = MyPTE1.PeakPS_GetPower(NoOfPoints, PowerArray, PeakPower) NoOfPoints = response[1] PowerArray = response[2] PeakPower = response[3]
Visual Basic	power = MyPTE1.PeakPS_GetPower(NoOfPoints, PowerArray(), PeakPower)
Visual C++	power = MyPTE1->PeakPS_GetPower(NoOfPoints, PowerArray(), PeakPower);
Visual C#	power = MyPTE1.PeakPS_GetPower(ref(NoOfPoints),ref(PowerArray()),ref(PeakPower));
MatLab	[power, NoOfPoints, PowerArray(), PeakPower] = MyPTE1.PeakPS_GetPower(NoOfPoints, PowerArray(), PeakPower);

See Also

Compensation Frequency

Read Average Power

Read Peak Power

2.7.8. SEND SCPI COMMAND

Short Send_SCPI(String SndSTR, ByRef String RetSTR)

Sends a SCPI (Standard Commands for Programmable Instruments) command to the power sensor and collects the response. This function only applies to Mini-Circuits' RC series of Ethernet enabled power sensors, using the ASCII / SCPI commands detailed in SCPI Commands for Power Sensor Control.

Parameters

Variable	Description
SndSTR	The SCPI command / query to send
RetSTR	String variable passed by reference, to be updated with the power sensor's response to the command / query

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.Send_SCPI(":MN?", "")` `response = str(status[1])`
Visual Basic	`status = MyPTE1.Send_SCPI(":MN?", response)`
Visual C++	`status = MyPTE1->Send_SCPI(":MN?", response);`
Visual C#	`status = MyPTE1.Send_SCPI(":MN?", ref(response));`
MatLab	`[status, response] = MyPTE1.Send_SCPI(":MN?", response)`

See Also

SCPI Commands for Power Sensor Control

2.8. DLL - Ethernet Configuration Functions

These functions provide a method of configuring the device's Ethernet IP settings, they can only be sent using the USB connection. The controller must be reset after updating Ethernet parameters in order to load the new configuration, this can be achieved with a power cycle or by using the ResetDevice command.

Refer to Ethernet Control API for additional details on the Ethernet configuration and default behavior.

Function	Syntax
Get Ethernet Configuration	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 GW1, ByRef int GW2, ByRef int GW3, ByRef int GW4)
Get DHCP Status	int GetEthernet_UseDHCP()
Use DHCP	int SaveEthernet_UseDHCP(int UseDHCP)
Get IP Address	int GetEthernet_IPAddress(ByRef int b1, b2, b3, b4)
Save IP Address	int SaveEthernet_IPAddress(int b1, int b2, int b3, int b4)
Get MAC Address	int GetEthernet_MACAddress(ByRef int m1, m2, m3, m4, m5, m6)
Get Network Gateway	int GetEthernet_NetworkGateway(ByRef int b1, b2, b3, b4)
Save Network Gateway	int SaveEthernet_NetworkGateway(int b1, int b2, int b3, int b4)
Get Subnet Mask	int GetEthernet_SubNetMask(ByRef int b1, b2, b3, b4)
Save Subnet Mask	int SaveEthernet_SubnetMask(int b1, int b2, int b3, int b4)
Get TCP/IP Port	int GetEthernet_TCPIPPort(ByRef int port)
Save TCP/IP Port	int SaveEthernet_TCPIPPort(int port)
Get Password Requirement	int GetEthernet_UsePWD()
Set Password Requirement	int SaveEthernet_UsePWD(int UsePwd)
Get Password	int GetEthernet_PWD(ByRef string Pwd)
Set Password	int SaveEthernet_PWD(string Pwd)
Get Ethernet Status	int GetEthernet_EnableEthernet()
Enable / Disable Ethernet	int SaveEthernet_EnableEthernet(short Enable)
Reset Device	byte ResetDevice()

2.8.1. GET ETHERNET CONFIGURATION

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)

Returns the IP configuration that is currently use, either the static IP entered by the user, or the server assigned dynamic IP configuration when DHCP is enabled.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

```
status = MyPTE1.GetEthernet_CurrentConfig("", "", "", "", "", "", "", "", "", "", "", "")
if status[0] > 0:
        print("IP:", str(status[1]), str(status[2]), str(status[3]), str(status[4]))
        print("Mask:", str(status[1]), str(status[2]), str(status[3]), str(status[4]))
        print("Gateway:", str(status[1]), str(status[2]), str(status[3]), str(status[4]))
```

Visual Basic

```
If MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
                                                                G1, G2, G3, G4) > 0 Then
        MsgBox ("IP: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
        MsgBox ("Mask: " & M1 & "." & M2 & "." & M3 & "." & M4)
        MsgBox ("Gateway: " & G1 & "." & G2 & "." & G3 & "." & G4)
End If
```

Visual C++

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

Visual C#

```
if (MyPTE1.GetEthernet_CurrentConfig(ref(IP1), ref(IP2), ref(IP3), ref(IP4),
           ref(M1), ref(M2), ref(M3), ref(M4), ref(GW1), ref(GW2), ref(GW3), ref(GW4)) > 0)
{
        MessageBox.Show("IP: " + IP1 + "." + IP2 + "." + IP3 + "." + IP4);
        MessageBox.Show("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('', '', '', '', '', '', '', '', '', '', '', '')
if status > 0
        h = msgbox("IP: ", IP1, ".", IP2, ".", IP3, ".", IP4)
        h = msgbox("Mask: ", M1, ".", M2, ".", M3, ".", M4)
        h = msgbox("Gateway: ", M1, ".", M2, ".", M3, ".", M4)
end
```

See Also

Get DHCP Status
Get IP Address
Get Network Gateway
Get Subnet Mask

2.8.2. GET DHCP STATUS

int GetEthernet_UseDHCP()

Indicates whether DHCP (dynamic host control protocol) is currently enabled. When disabled, the device will attempt to connect using the user-entered static IP parameters.

Return Values

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

Examples

Python	`response = MyPTE1.GetEthernet_UseDHCP()`
Visual Basic	`response = MyPTE1.GetEthernet_UseDHCP()`
Visual C++	`response = MyPTE1->GetEthernet_UseDHCP();`
Visual C#	`response = MyPTE1.GetEthernet_UseDHCP();`
MatLab	`response = MyPTE1.GetEthernet_UseDHCP()`

See Also

Get Ethernet Configuration

2.8.3. USE DHCP

int SaveEthernet_UseDHCP(int UseDHCP)

Enables or disables DHCP (dynamic host control protocol). When disabled, the device will attempt to connect using the user-entered static IP parameters. By default, DHCP is enabled.

Parameters

Variable	Description
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_UseDHCP(1)`
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);`

2.8.4. GET IP ADDRESS

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

Returns the user-entered static IP address.

Parameters

Variable	Description
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").
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").
IP2	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").
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetEthernet_IPAddress("", "", "", "")`
	`if status[0] > 0:`
	`    print("IP:", str(status[1]), str(status[2]), str(status[3]), str(status[4]))`

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

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

Visual C#	`if (MyPTE1.GetEthernet_IPAddress(ref(IP1), ref(IP2), ref(IP3), ref(IP4)) > 0)`
	`{`
	`        MessageBox.Show("IP: " + IP1 + "." + IP2 + "." + IP3 + "." + IP4);`
	`}`

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

See Also

Get Ethernet Configuration
Get DHCP Status

2.8.5. SAVE IP ADDRESS

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

Sets the static IP address to be used when DHCP is disabled.

Parameters

Variable	Description
IP1	Required. First (highest order) octet of the IP address to set (for example "192" for the IP address "192.168.1.0").
IP2	Required. Second octet of the IP address to set (for example "168" for the IP address "192.168.1.0").
IP2	Required. Third octet of the IP address to set (for example "1" for the IP address "192.168.1.0").
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

Use DHCP

2.8.6. GET MAC ADDRESS

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

Returns the physical MAC (media access control) address of the device.

Parameters

Variable	Description
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
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
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
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
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
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

```
status = MyPTE1.GetEthernet_MACAddress("", "", "", "", "", "")
if status[0] > 0:
    print("MAC:", str(status[1]), str(status[2]), str(status[3]),
                                            str(status[4]), str(status[5]), str(status[6]))
```

Visual Basic

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

Visual C++

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

Visual C#

```
if (MyPTE1.GetEthernet_MACAddress(ref(M1), ref(M2), ref(M3), ref(M4),
                                                        ref(M5), ref(M6)) > 0)
{
        MessageBox.Show("Mask: " + M1 + "." + M2 + "." + M3
                                                    + "." + M4 + "." + M5 + "." + M6);
}
```

MatLab

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

2.8.7. GET NETWORK GATEWAY

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

Returns the user-entered network gateway IP address.

Parameters

Variable	Description
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").
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").
IP2	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").
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetEthernet_NetworkGateway("", "", "", "")`
	`if status[0] > 0:`
	`    print("IP:", str(status[1]), str(status[2]), str(status[3]), str(status[4]))`

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

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

Visual C#	`if (MyPTE1.GetEthernet_NetworkGateway(ref(IP1), ref(IP2), ref(IP3),ref(IP4)) > 0)`
	`{`
	`        MessageBox.Show("IP: " + IP1 + "." + IP2 + "." + IP3 + "." + IP4);`
	`}`

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

See Also

Get Ethernet Configuration
Get DHCP Status

2.8.8. SAVE NETWORK GATEWAY

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

Sets the IP address of the network gateway to be used when DHCP is disabled.

Parameters

Variable	Description
IP1	Required. First (highest order) octet of the network gateway IP address (for example "192" for the IP address "192.168.1.0").
IP2	Required. Second octet of the network gateway IP address (for example "168" for the IP address "192.168.1.0").
IP2	Required. Third octet of the network gateway IP address (for example "1" for the IP address "192.168.1.0").
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

Use DHCP

2.8.9. GET SUBNET MASK

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

Returns the user-entered subnet mask.

Parameters

Variable	Description
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").
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").
b2	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").
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

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetEthernet_SubNetMask("", "", "", "")`
	`if status[0] > 0:`
	`  print(str(status[1]), str(status[2]), str(status[3]), str(status[4]))`

Visual Basic	`If MyPTE1.GetEthernet_SubNetMask(IP1, IP2, IP3, IP4) > 0 Then`
	`  MsgBox (IP1 & "." & IP2 & "." & IP3 & "." & IP4)`
	`End If`

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

Visual C#	`if (MyPTE1.GetEthernet_SubNetMask(ref(IP1), ref(IP2), ref(IP3), ref(IP4)) > 0)`
	`{`
	`  MessageBox.Show(IP1 + "." + IP2 + "." + IP3 + "." + IP4);`
	`}`

MatLab	`[status, IP1, IP2, IP3, IP4] = MyPTE1.GetEthernet_SubNetMask('', '', '', '')`
	`if status > 0`
	`  h = msgbox(IP1, ".", IP2, ".", IP3, ".", IP4)`
	`end`

See Also

Get Ethernet Configuration
Get DHCP Status

2.8.10. SAVE SUBNET MASK

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

Sets the subnet mask to be used when DHCP is disabled.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

Use DHCP

2.8.11. GET TCP/IP PORT

int GetEthernet_TCPIPPort(ByRef int port)

Returns the TCP/IP port in use for HTTP communication. The default is port 80.

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetEthernet_TCPIPPort("")`
	`if status[0] > 0:`
	`  port = str(status[1])`
	`  print(port)`

Visual Basic	`If MyPTE1.GetEthernet_TCPIPPort(port) > 0 Then`
	`  MsgBox (port)`
	`End If`

Visual C++	`if (MyPTE1->GetEthernet_TCPIPPort(port) > 0)`
	`{`
	`  MessageBox::Show(port);`
	`}`

Visual C#	`if (MyPTE1.GetEthernet_TCPIPPort(ref(port)) > 0)`
	`{`
	`  MessageBox.Show(port);`
	`}`

MatLab	`[status, port] = MyPTE1.GetEthernet_TCPIPPort('')`
	`if status > 0`
	`  h = msgbox(port)`
	`end`

2.8.12. SAVE TCP/IP PORT

int SaveEthernet_TCPIPPort(int port)

Sets the TCP / IP port to be used 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

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_TCPIPPort(70)`
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);`

2.8.13. GET PASSWORD REQUIREMENT

int GetEthernet_UsePWD()

Indicates whether the password is currently enabled for HTTP / Telnet (the password is always required for SSH).

Return Values

Value	Description
0	Password not required
1	Password required

Examples

Python	`response = MyPTE1.GetEthernet_UsePWD()`
Visual Basic	`response = MyPTE1.GetEthernet_UsePWD()`
Visual C++	`response = MyPTE1->GetEthernet_UsePWD();`
Visual C#	`response = MyPTE1.GetEthernet_UsePWD();`
MatLab	`response = MyPTE1.GetEthernet_UsePWD()`

See Also

Get Password

2.8.14. SET PASSWORD REQUIREMENT

int SaveEthernet_UsePWD(int UsePwd)

Enables or disables the password requirement for HTTP / Telnet (the password is always required for SSH).

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_UsePWD(1)`
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

Set Password

2.8.15. GET PASSWORD

int GetEthernet_PWD(ByRef string Pwd)

Returns the current password for HTTP / Telnet communication. The password will be returned even if the device is not currently configured to require a password.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetEthernet_PWD("")`
	`if status[0] > 0:`
	`  pwd = str(status[1])`
	`  print(pwd)`

Visual Basic	`If MyPTE1.GetEthernet_PWD(pwd) > 0 Then`
	`  MsgBox (pwd)`
	`End If`

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

Visual C#	`if (MyPTE1.GetEthernet_PWD(ref(pwd)) > 0)`
	`{`
	`  MessageBox.Show(pwd);`
	`}`

MatLab	`[status, pwd] = MyPTE1.GetEthernet_PWD('')`
	`if status > 0`
	`  h = msgbox(pwd)`
	`end`

See Also

Get Password Requirement

2.8.16. SET PASSWORD

int SaveEthernet_PWD(string Pwd)

Sets the password used for HTTP / Telnet communication. The password will not affect operation unless Use Password is also enabled.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_PWD("123")`
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

Set Password Requirement

2.8.17. GET ETHERNET STATUS

int GetEthernet_EnableEthernet()

Indicates whether Ethernet communication is enabled or disabled. Disabling Ethernet control is recommended when not needed, in order to reduce current consumption.

Return Values

Value	Description
0	Ethernet control is disabled
1	Ethernet control is enabled

Examples

Python	`response = MyPTE1.GetEthernet_EnableEthernet()`
Visual Basic	`response = MyPTE1.GetEthernet_EnableEthernet()`
Visual C++	`response = MyPTE1->GetEthernet_EnableEthernet();`
Visual C#	`response = MyPTE1.GetEthernet_EnableEthernet();`
MatLab	`response = MyPTE1.GetEthernet_EnableEthernet()`

2.8.18. ENABLE / DISABLE ETHERNET

int SaveEthernet_EnableEthernet(short Enable)

Enable or disable Ethernet communication. Disabling Ethernet control is recommended when not needed, in order to reduce current consumption.

Parameters

Variable	Description
Enable	Required. Integer value to enable / disable Ethernet control: 0 – Ethernet control disabled 1 – Ethernet control enabled

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_EnableEthernet(1)`
Visual Basic	`status = MyPTE1.SaveEthernet_EnableEthernet(1)`
Visual C++	`status = MyPTE1->SaveEthernet_EnableEthernet(1);`
Visual C#	`status = MyPTE1.SaveEthernet_EnableEthernet(1);`
MatLab	`status = MyPTE1.SaveEthernet_EnableEthernet(1);`

2.8.19. RESET DEVICE

byte ResetDevice()

Called after updating Ethernet parameters, to reset the controller and reload with the updated Ethernet configuration.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`MyPTE1.ResetDevice()`
Visual Basic	`MyPTE1.ResetDevice()`
Visual C++	`MyPTE1->ResetDevice();`
Visual C#	`MyPTE1.ResetDevice();`
MatLab	`MyPTE1.ResetDevice();`

3. USB Control via Direct Programming (Linux)

Mini-Circuits' API DLL files require a programming environment which supports either .NET or ActiveX. Where this is not available (for example on a Linux operating system) the alternative method is "direct" USB programming using USB interrupts.

3.1. USB Interrupt Code Concept

To open a connection to Mini-Circuits programmable attenuators, the Vendor ID and Product ID are required:

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

Communication with the attenuator 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 attenuator.

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 programmable attenuator as a USB HID (Human Interface Device).

3.2. Interrupts - General Functions

Description	Command Code
Get Device Model Name	104
Get Device Serial Number	105
Set Measurement Mode	15
Read Power	102
Get Internal Temperature	103
Get Firmware	99
Send SCPI Command	42 or 121

3.2.1. GET DEVICE MODEL NAME

Description

Returns the full Mini-Circuits part number of the connected power 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' PWR-8FS power sensor.

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	104	80	87	82	45	56
ASCII Character	N/A	P	W	R	-	8

Byte	Byte 6	Byte 7	Byte 8
Description	Char 6	Char 7	End Marker
Value	70	83	0
ASCII Character	F	S	N/A

See Also

Get Device Serial Number

3.2.2. GET DEVICE SERIAL NUMBER

Description

Returns the serial number of the connected power 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

3.2.3. SET MEASUREMENT MODE

Description

Sets the measurement mode of an average power sensor between "low noise" and "fast sampling" modes; the default is "low noise" mode. Additionally, "fastest sampling" mode is also available for PWR-8FS. See the individual model datasheets for specifications.

Note: Does not apply to PWR-xP series of peak & average power sensors.

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 = Fastest sampling mode (PWR-8FS only)
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

3.2.4. READ POWER

Description

Returns the sensor power measurement based on a user specified compensation frequency.

The power value (in dBm) is represented in BYTE1 to BYTE6 of the returned array as a series of ASCII character codes in the format "+00.00".

Transmit Array

Byte	Data	Description
0	102	Interrupt code for Read Power
1	Frequency_1	The compensation frequency to be used for the power reading, split over 2 bytes: Frequency_1 = INT (FREQUENCY / 256)
2	Frequency_2	The compensation frequency to be used for the power reading, split over 2 bytes: Frequency_2 = FREQUENCY - (Frequency_1 * 256)
3	Freq_Units	ASCII character code representing the units for the compensation frequency, the 2 options are: 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 - 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:

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 array would be returned 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

3.2.5. GET INTERNAL TEMPERATURE

Description

This function returns the internal temperature of the power 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 - 63	Not significant	"Don't care" bytes, can be any value

3.2.6. GET FIRMWARE

Description

Returns the internal firmware version of the power 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	Firmware Letter	ASCII code for the first character in the firmware revision identifier
4	Firmware Number	ASCII code for the second character in the firmware revision identifier
5 - 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	1	Internal code for factory use only
2	12	Internal code for factory use only
3	65	ASCII code for the letter "A"
4	51	ASCII code for the number 3
5 - 63	Not significant	"Don't care" bytes, could be any value

3.2.7. SEND SCPI COMMAND

Description

Sends a SCPI (Standard Commands for Programmable Instruments) command to the power sensor and collects the response. This function only applies to Mini-Circuits' RC series of Ethernet enabled power sensors, using the ASCII / SCPI commands detailed in SCPI Commands for Power Sensor Control.

Transmit Array

Byte	Data	Description
0	42 or 121	Interrupt code for Send SCPI Command
1	SCPI_Length	The length (number of ASCII characters) of the SCPI string to send
2 to 63	SCPI Transmit String	The SCPI command to be sent represented as a series of ASCII character codes, one character code per byte

Returned Array

Byte	Data	Description
0	42 or 121	Interrupt code for Send SCPI Command
1	SCPI_Length	The length (number of ASCII characters) of the SCPI command sent in the transmit array
2 to 7	Transmit_Array	Bytes 2 to 7 of the transmit array repeated
8 to (n-1)	SCPI Return String	The SCPI return string, one character per byte, represented as ASCII character codes
n	0	Zero value byte to indicate the end of the SCPI return string
(n+1) to 63	Not significant	"Don't care" bytes, could be any value

Example (Get Model Name)

The SCPI command to request the model name is :MN? (see Get Model Name)

The ASCII character codes representing the 4 characters in this command should be sent in bytes 2 to 5 of the transmit array as follows:

Byte	Data	Description
0	42	Interrupt code for Send SCPI Command
1	4	Length of the SCPI command (four ASCII characters)
2	49	ASCII character code for :
3	77	ASCII character code for M
4	78	ASCII character code for N
5	63	ASCII character code for ?

See Also

SCPI Commands for Power Sensor Control

3.3. Interrupts - Ethernet Configuration Functions (RC Models Only)

Description	Command Code	
	Byte 0	Byte 1
Set Static IP Address	250	201
Set Static Subnet Mask	250	202
Set Static Network Gateway	250	203
Set HTTP Port	250	204
Use Password	250	205
Set Password	250	206
Use DHCP	250	207
Get Static IP Address	251	201
Get Static Subnet Mask	251	202
Get Static Network Gateway	251	203
Get HTTP Port	251	204
Get Password Status	251	205
Get Password	251	206
Get DHCP Status	251	207
Get Dynamic Ethernet Configuration	253	
Get MAC Address	252	
Enable / Disable Ethernet	250	208
Reset Ethernet Configuration	101	101

3.3.1. 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

3.3.2. 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

3.3.3. 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

3.3.4. 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

Get HTTP Port

Reset Ethernet Configuration

3.3.5. 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

3.3.6. SET PASSWORD

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

Reset Ethernet Configuration

3.3.7. 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 = DCHP 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

3.3.8. 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

3.3.9. 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

3.3.10. 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

3.3.11. 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

3.3.12. GET PASSWORD STATUS

Description

Checks whether the device 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

3.3.13. 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 - 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 - 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

3.3.14. 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 = DHCP 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

3.3.15. 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

3.3.16. 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

3.3.17. ENABLE / DISABLE ETHERNET

Description

Enable or disable Ethernet communication. Disabling Ethernet control is recommended when not needed, in order to reduce current consumption.

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	208	Interrupt code for Enable / Disable Ethernet
2	Mode	0 = Ethernet disabled 1 = Ethernet enabled
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 Ethernet control, the transmit array is:

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	208	Interrupt code for Enable / Disable Ethernet
2	1	Enable Ethernet control

3.3.18. 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 API (RC Models Only)

Control of the device via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed below via HTTP or Telnet. In addition, UDP is supported for discovering available systems on the network.

These protocols are widely supported and straightforward to implement in most programming environments. Any Internet browser can be used as a console / tester for HTTP control by typing the full URL directly into the address bar. Telnet is supported by a number of console applications, including PuTTY.

4.1. Configuring Ethernet Settings

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

4.2. HTTP Communication

HTTP Get / Post are supported. The basic format of the HTTP command:

```
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 / query to send to the device

Example 1:

```
http://192.168.100.100:800/PWD=123;:FREQ:1000
```

- The power 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

Example 2:

```
http://10.10.10.10/:POWER?
```

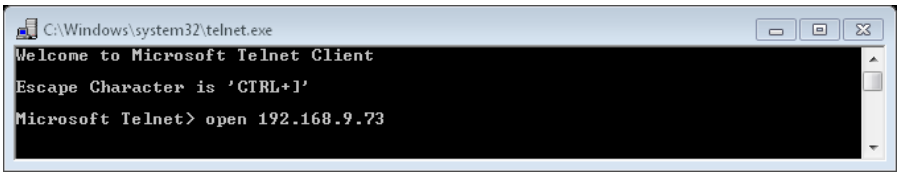
- The power sensor has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The query is to return the current power reading

4.2.1. TELNET COMMUNICATION

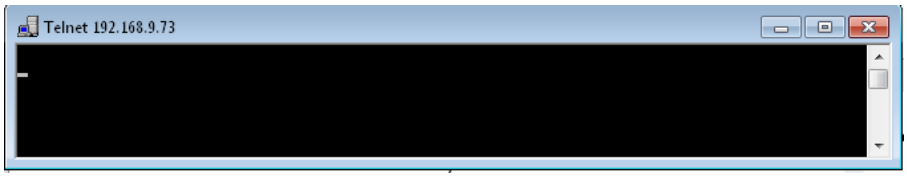
Communication is started by creating a Telnet connection to the device's IP address. On successful connection the "line feed" character will be returned. If the system has a password enabled then this must be sent as the first command after connection.

Each command must be terminated with the carriage return and line-feed characters (\r\n). Responses will be similarly terminated. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below:

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

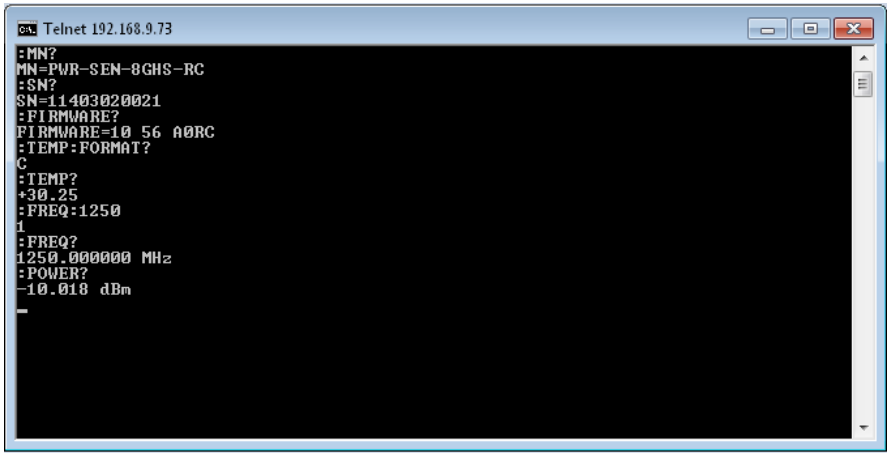


2) The "line feed" character is returned indicating the connection was successful:



3) The password (if enabled) must be sent as the first command in the format "PWD=x;". A return value of "1" indicates success:

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



5) Use the control and "]" keys to end the session.

4.3. Device Discovery Using UDP

Limited support of UDP is provided for the purpose of "device discovery." This allows a user to request the IP address and configuration of all Mini-Circuits' devices within the same family, 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 power sensor with the USB interface (see DLL - Ethernet Configuration Functions).

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

UDP Ports

Mini-Circuits' power 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 power 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: PWR-8GHS-RC
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: PWR-8GHS-RC
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
```

5. SCPI Commands for Power Sensor Control

This section describes a series of ASCII text commands based on SCPI (Standard Commands for Programmable Instruments) that provide an additional control approach for Mini-Circuits' RC series of Ethernet enabled power sensors. These commands can be sent using any of the APIs detailed in this manual.

The SCPI commands / queries are case insensitive and sent as an ASCII text string (up to 63 characters). The response from the system is also in the form of an ASCII text string.

If an unrecognized command/query is received the sensor will return:

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

5.1. SCPI - General Functions

These functions apply to Mini-Circuits' RC series of power sensors with an Ethernet interface.

Description	Command / Query
Get Model Name	:MN?
Get Serial Number	:SN?
Get Firmware	:FIRMWARE?
Get Temperature Units	:TEMP:FORMAT?
Set Temperature Units	:TEMP:FORMAT:[units]
Get Internal Temperature	:TEMP?
Get Compensation Frequency	:FREQ?
Set Compensation Frequency	:FREQ:[freq]

5.1.1. GET MODEL NAME

MN?

Return the Mini-Circuits model name.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return Value

`MN=[model]`

Value	Description
[model]	Model name of the connected device

Examples

String to Send	String Returned
MN?	MN=RC-2SPDT-A18

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

5.1.2. GET SERIAL NUMBER

SN?

Returns the serial number.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return Value

`SN=[serial]`

Value	Description
`[serial]`	Serial number of the connected device

Examples

String to Send	String Returned
`SN?`	`SN=12208010025`

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

5.1.3. GET FIRMWARE

FIRMWARE?

Returns the internal firmware version.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return Value

Value	Description
`[firmware]`	The current firmware version, for example "B3".

Examples

String to Send	String Returned
`FIRMWARE?`	`B3`

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

5.1.4. GET TEMPERATURE UNITS

:TEMP:FORMAT?

Returns the units for internal temperature readings.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return String

Value	Description
F	Temperature measurements in degrees Fahrenheit
C	Temperature measurements in degrees Celsius

Examples

String to Send	String Returned
:TEMP:FORMAT?	C

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

5.1.5. SET TEMPERATURE UNITS

:TEMP:FORMAT:[units]

Sets the units for internal temperature readings.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Parameters

Value	Description
F	Temperature readings in degrees Fahrenheit
C	Temperature readings in degrees Celsius

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TEMP:FORMAT:C	1

HTTP Implementation: http://10.10.10.10/:TEMP:FORMAT:C

5.1.6. GET INTERNAL TEMPERATURE

:TEMP?

Returns the internal temperature of the power sensor. Note: The reading provides an indication but is not calibrated.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return String

Variable	Description
[temperature]	Internal temperature reading

Examples

String to Send	String Returned
:TEMP?	+25.50

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

5.1.7. GET COMPENSATION FREQUENCY

:FREQ?

Returns the frequency currently set for calibrating the input power measurements.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Return String

Variable	Description
[freq]	Compensation frequency in MHz

Examples

String to Send	String Returned
:FREQ?	2500.000000 MHz

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

5.1.8. SET COMPENSATION FREQUENCY

:FREQ:[freq]

Sets the compensation frequency for calibrating input power measurements. This parameter must be set to ensure measurement accuracy.

Note: PWR series power sensors do not have frequency selectivity.

Applies To

Mini-Circuits' RC series of power sensors with an Ethernet interface.

Parameters

Variable	Description
[freq]	Compensation frequency in MHz

Return String

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:FREQ:2500	1

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

5.2. SCPI - Measuring with Average Power Sensors

These functions apply to the following Mini-Circuits' RC series power sensors with an Ethernet interface:

- PWR-xGHS-RC Series (CW average power sensors)
- PWR-xRMS-RC Series (true RMS power sensors)

Description	Command / Query
Get Measurement Mode	:MODE?
Set Measurement Mode	:MODE:[speed]
Get Averaging Mode	:AVG:STATE?
Set Averaging Mode	:AVG:STATE:[mode]
Get Average Count	:AVG:COUNT?
Set Average Count	:AVG:COUNT:[count]
Read Average Power	:POWER?
Read Voltage	:VOLTAGE?

5.2.1. GET MEASUREMENT MODE

:MODE?

Indicates the measurement mode of the power sensor; "low noise", "fast sampling" or "fastest sampling". The specifications for these modes are defined in the individual model datasheets. The default is "low noise" mode.

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Return String

Value	Description
0	Low noise mode
1	Fast sampling mode
2	Fastest sampling mode

Examples

String to Send	String Returned
:MODE?	1

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

5.2.2. SET MEASUREMENT MODE

:MODE:[speed]

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

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Parameters

Variable	Value	Description
[speed]	0	Low noise mode
	1	Fast sampling mode
	2	Fastest sampling mode

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:MODE:1	1

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

5.2.3. GET AVERAGING MODE

:AVG:STATE?

Indicates whether "averaging" mode is on or off (the default is averaging off).

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Return String

Value	Description
0	Averaging mode disabled
1	Averaging mode enabled

Examples

String to Send	String Returned
:AVG:STATE?	1

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

5.2.4. SET AVERAGING MODE

:AVG:STATE:[mode]

Enables or disables the power sensor "averaging" mode.

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Parameters

Value	Description
0	Averaging mode disabled
1	Averaging mode enabled

Return String

Value	Description
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

5.2.5. GET AVERAGE COUNT

:AVG:COUNT?

Returns the number of power readings (from 1 to 32) over which the measurement will be averaged when averaging mode is enabled.

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Return String

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

Examples

String to Send	String Returned
:AVG:COUNT?	3

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

5.2.6. SET AVERAGE COUNT

:AVG:COUNT:[count]

Sets the number of power readings (from 1 to 32) over which to average the measurement when averaging mode is enabled. The default value is 1 (average the reading over 1 measurement).

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Parameters

Variable	Description
[count]	The number of readings over which to average the power reading, from 1 to 32

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:AVG:COUNT:10	1

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

5.2.7. READ AVERAGE POWER

:POWER?

Returns the input power measurement in dBm. The compensation frequency should be set prior to reading power in order to achieve the specified accuracy.

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Return String

Variable	Description
[power]	Input power measurement in dBm. Note: a power value of -99.000 dBm indicates that the input signal level is below the sensor's useable range.

Examples

String to Send	String Returned
:POWER?	-22.050 dBm

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

See Also

Set Compensation Frequency

5.2.8. READ VOLTAGE

:VOLTAGE?

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

Applies To

- PWR-xGHS-RC Series (CW average power sensors with an Ethernet interface)
- PWR-xRMS-RC Series (true RMS power sensors with an Ethernet interface)

Return String

Variable	Description
[volts]	Input voltage reading in mV

Examples

String to Send	String Returned
:VOLTAGE?	0.000105 Volt

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

5.3. SCPI – Measuring with Peak & Average Power Sensors

These functions apply to Mini-Circuits' PWR-xP Series peak & average power sensor models.

Description	Command / Query
Get Trigger Mode	:TRIGGER:MODE?
Set Trigger Mode	:TRIGGER:MODE:[type]
Get External Trigger Type	:TRIGGER:EXTERNAL:[type]?
Set External Trigger Type	:TRIGGER:EXTERNAL:[type]
Get Trigger Delay	:TRIGGER:DELAY?
Set Trigger Delay	:TRIGGER:DELAY:[time]
Get Trigger Timeout	:TRIGGER:TIMEOUT?
Set Trigger Timeout	:TRIGGER:TIMEOUT:[period]
Get Internal Trigger Level	:TRIGGER:LEVEL?
Set Internal Trigger Level	:TRIGGER:LEVEL:[power]
Get Trigger Output Mode	:EXTOUT:SELECT?
Set Trigger Output Mode	:EXTOUT:SELECT:[type]
Get Sample Time	:SAMPLETIME?
Set Sample Time	:SAMPLETIME:[time]
Read Peak & Average Power	:POWER?
Read Initial Power Array	:POWER_ARRAY?
Read Subsequent Power Arrays	:POWER_ARRAY_EP[package]?

5.3.1. GET TRIGGER MODE

:TRIGGER:MODE?

Checks which trigger setting is currently in use.

Note: For internal / external trigger mode, the power measurement will time-out if no trigger is detected within the specified timeout period, following a "read power" request. In the event of a timeout, the last power measurement will be returned from the internal buffer.

Applies To

PWR-xP Series - Peak & average power sensors

Return String

Value	Description
FREE	Free-running trigger; power sampling begins as soon as a "read power" request is made
INTERNAL	Internal trigger; power sampling will start on the first rising edge which passes the trigger threshold, following a "read power" request.
EXTERNAL	External trigger; power sampling will start when an external trigger input signal is detected.

Examples

String to Send	String Returned
:TRIGGER:MODE?	FREE

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

See Also

Set Trigger Mode

Get External Trigger Type

Set External Trigger Type

5.3.2. SET TRIGGER MODE

:TRIGGER:MODE:[type]

Sets the event which triggers the start of the power sensor's sample period.

Note: For internal / external trigger mode, the power measurement will time-out if no trigger is detected within the specified timeout period, following a "read power" request. In the event of a timeout, the last power measurement will be returned from the internal buffer.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Value	Description
[type]	FREE	Free-running trigger, power sampling begins as soon as a "read power" request is made
	INTERNAL	Internal trigger, power sampling will start on the first rising edge which passes the trigger threshold, following a "read power" request.
	EXTERNAL	External trigger, power sampling will start when an external trigger input signal is detected.

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:MODE:FREE	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:MODE:FREE

See Also

Get Trigger Mode

Get External Trigger Type

Set External Trigger Type

5.3.3. GET EXTERNAL TRIGGER TYPE

:TRIGGER:EXTERNAL:[type]?

Indicates whether power sampling will start on the rising or falling edge of an external trigger input signal when the power sensor is operating in external trigger mode.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Value	Description
[type]	ONFALL	Query whether power sampling is to start on the falling edge of an external trigger input signal
	ONRISE	Query whether power sampling is to start on the rising edge of an external trigger input signal

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:EXTERNAL:ONFALL?	0
:TRIGGER:EXTERNAL:ONRISE?	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:EXTERNAL:ONFALL?

See Also

Get Trigger Mode

Set Trigger Mode

Set External Trigger Type

5.3.4. SET EXTERNAL TRIGGER TYPE

:TRIGGER:EXTERNAL:[type]

Sets whether power sampling will start on the rising or falling edge of an external trigger input signal when the power sensor is operating in external trigger mode.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Value	Description
[type]	ONFALL	Power sampling will start on the falling edge of an external trigger input signal
	ONRISE	Power sampling will start on the rising edge of an external trigger input signal

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:EXTERNAL:ONFALL	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:EXTERNAL:ONFALL

See Also

Get Trigger Mode

Set Trigger Mode

Get External Trigger Type

5.3.5. GET TRIGGER DELAY

:TRIGGER:DELAY?

Indicates the delay to be applied between detection of a trigger signal and the start of power sampling. Applies to internal and external trigger modes.

Applies To

PWR-xP Series - Peak & average power sensors

Return String

Variable	Description
[time]	Delay time in microseconds (µS) between detection of a trigger signal and the start of power sampling

Examples

String to Send	String Returned
:TRIGGER:DELAY?	100

HTTP Implementation: http://10.10.10.10/:TRIGGER:DELAY?

5.3.6. SET TRIGGER DELAY

:TRIGGER:DELAY:[time]

Sets the delay between detection of a trigger signal and the start of power sampling. Applies to internal and external trigger modes.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Description
[time]	Delay time in microseconds (µS) between detection of a trigger signal and the start of power sampling

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:DELAY:100	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:DELAY:0

5.3.7. GET TRIGGER TIMEOUT

:TRIGGER:TIMEOUT?

Returns the period in ms after which the power measurement will timeout if no trigger signal is detected in internal / external trigger mode. In the event of a timeout, the last power measurement will be returned from the internal buffer.

Applies To

Model	Firmware
PWR-8P-RC	A9 or later
PWR-8PW-RC	A1 or later
PWR-40PW-RC	A3 or later

Return Values

Variable	Description
[period]	The trigger timeout period in ms, from 1 to 5000. 500 ms is the default setting.

Examples

String to Send	String Returned
:TRIGGER:TIMEOUT?	500

HTTP Implementation: http://10.10.10.10/:TRIGGER:TIMEOUT?

5.3.8. SET TRIGGER TIMEOUT

:TRIGGER:TIMEOUT:[period]

Sets the period in ms after which the power measurement will timeout if no trigger signal is detected in internal / external trigger mode. In the event of a timeout, the last power measurement will be returned from the internal buffer.

Applies To

Model	Firmware
PWR-8P-RC	A9 or later
PWR-8PW-RC	A1 or later
PWR-40PW-RC	A3 or later

Parameters

Variable	Description
[period]	The trigger timeout period in ms, from 1 to 5000. 500 ms is the default setting.

Return Values

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:TIMEOUT:500	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:TIMEOUT:500

5.3.9. GET INTERNAL TRIGGER LEVEL

:TRIGGER:LEVEL?

Returns the input power threshold at which an internal trigger is registered.

Applies To

PWR-40PW-RC only

Return Values

Variable	Description
[power]	The input power level in dBm at which an internal trigger is registered, from -20 to +20.

Examples

String to Send	String Returned
:TRIGGER:LEVEL?	-10

HTTP Implementation: http://10.10.10.10/:TRIGGER:LEVEL?

5.3.10. SET INTERNAL TRIGGER LEVEL

:TRIGGER:LEVEL:[power]

Sets the input power threshold at which an internal trigger is registered.

Applies To

PWR-40PW-RC only

Parameters

Variable	Description
[power]	The input power level in dBm at which an internal trigger is registered, from -20 to +20.

Return Values

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:TRIGGER:LEVEL:-10	1

HTTP Implementation: http://10.10.10.10/:TRIGGER:LEVEL:-10

5.3.11. GET TRIGGER OUTPUT MODE

:EXTOUT:SELECT?

Indicate whether the trigger output is set to a TTL trigger signal (pulsing at the start of the sample period) or video output (corresponding to the modulation of the RF input).

Applies To

PWR-xP Series - Peak & average power sensors

Return String

Value	Description
TRIG	Trigger output port will provide a TTL signal at the start of the sampling period
VIDEO	Trigger output port will provide a video signal corresponding to the modulation of the RF input

Examples

String to Send	String Returned
:EXTOUT:SELECT?	TRIG

HTTP Implementation: http://10.10.10.10/:EXTOUT:TRIGGER?

5.3.12. SET TRIGGER OUTPUT MODE

:EXTOUT:SELECT:[mode]

Sets trigger output between a TTL trigger signal (pulsing at the start of the sample period) or a video output (corresponding to the modulation of the RF input).

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Value	Description
[mode]	TRIG	Trigger output port will provide a TTL signal at the start of the sampling period
	VIDEO	Trigger output port will provide a video signal corresponding to the modulation of the RF input

Return String

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:EXTOUT:SELECT:TRIG	1

HTTP Implementation: http://10.10.10.10/:EXTOUT:SELECT:TRIG

5.3.13. GET SAMPLE TIME

:SAMPLETIME?

Returns the sample period for power measurements, from 10 µs to 1 s.

Applies To

PWR-xP Series - Peak & average power sensors

Return String

Variable	Description
[time]	Sample time in microseconds (µS)

Examples

String to Send	String Returned
SAMPLETIME?	10

HTTP Implementation: http://10.10.10.10/:SAMPLETIME?

5.3.14. SET SAMPLE TIME

:SAMPLETIME:[time]

Sets the sample period for power measurements, from 10 µs to 1 s.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Description
[time]	Sample time in microseconds (µS)

Return String

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:SAMPLETIME:10	1

HTTP Implementation: http://10.10.10.10/:SAMPLETIME:10

5.3.15. GET VIDEO FILTER BANDWIDTH

:BWFILTER?

Returns the video filter bandwidth.

Applies To

PWR-8PW-RC & PWR-40PW-RC

Return Values

Value	Description
300KHz	Video filter set to 300 kHz
1.5MHZ	Video filter set to 1.5 MHz
5MHZ	Video filter set to 5 MHz
30MHZ	Video filter turned off (30 MHz is the max supported measurement bandwidth)

Examples

String to Send	String Returned
:BWFILTER?	300KHZ

HTTP Implementation: http://10.10.10.10/:BWFILTER?

5.3.16. SET VIDEO FILTER BANDWIDTH

:BWFILTER:[bandwidth]

Sets the video filter bandwidth.

Applies To

PWR-8PW-RC & PWR-40PW-RC

Parameters

Variable	Value	Description
[bandwidth]	300KHz	Video filter set to 300 kHz
	1.5MHZ	Video filter set to 1.5 MHz
	5MHZ	Video filter set to 5 MHz
	30MHZ	Video filter turned off (30 MHz is the max supported measurement bandwidth)

Return Values

Value	Description
0 - Failed	Command failed
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:BWFILTER:300KHZ	1 - Success

HTTP Implementation: http://10.10.10.10/:BWFILTER:300KHZ

5.3.17. READ PEAK & AVERAGE POWER

:POWER?

Returns the peak and average power measurement in dBm (separated by a space character) for the complete sample period of the sensor. The compensation frequency must be set prior to reading power in order to achieve the specified accuracy.

Note: In the event of a trigger timeout (no valid trigger detected within the specified timeout period), the measurement cycle will end and the last buffered result will be returned from memory. Consider adding a timer in the automation program to monitor the response time to a measurement request and then discard the returned result if it is unchanged and the elapsed time is equal to the timeout period.

Applies To

PWR-xP Series - Peak & average power sensors

Return String

`[peak] [avg]`

Variable	Description
`[peak]`	Peak power level (dBm) measured over the sample period
`[avg]`	Average power level (dBm) measurement over the sample period

Examples

String to Send	String Returned
`:POWER?`	`-2.050 -25.250`

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

See Also

Set Compensation Frequency

Read Power Array

5.3.18. READ INITIAL POWER ARRAY (ETHERNET CONTROL)

:POWER_ARRAY?

Carries out a power measurement and stores the series of discrete measurements values into internal memory, grouped into packages of 160 measurements.

The query returns the number of packages (p), the total number of measurements (m) and the first 160 measured values. The compensation frequency must be set prior to reading power in order to achieve the specified accuracy.

The number of discrete measurements taken is variable but approximately equally spaced in time so that the number of measurements / total sample time = approximate time per measurement.

If p is greater than 1 (ie: more than 160 discrete measurement values were stored) then the Read Subsequent Power Arrays methodology should be used to iteratively return each subsequent package of 160 measurement values.

Once all packages have been obtained, the full array of discrete power measurement values can be used for calculated analysis of the measured input signal, including pulse width, crest factor, rise / fall time and duty cycle for example.

Note: In the event of a trigger timeout (no valid trigger detected within the specified timeout period), the measurement cycle will end and the last buffered result will be returned from memory. Consider adding a timer in the automation program to monitor the response time to a measurement request and then discard the returned result if it is unchanged and the elapsed time is equal to the timeout period.

Applies To

PWR-xP Series - Peak & average power sensors

Return String

`[p] [m] [val0] [val1] [val2] …[ val159]`

Variable	Description
`[p]`	Total number of packages (including this initial package) that the power measurement data has been split over
`[m]`	Total number of discrete measurements, contained in all `p` packages
`[valn]`	Series of discrete, absolute power measurements (dBm), each multiplied by 100 to give an integer value

Examples

The power measurement is split over 5 packages, containing 804 discrete values and the first 160 values are -60.25 dBm, -60.00 dBm, -60.50 dBm... -60.25 dBm:

String to Send	String Returned
`:POWER_ARRAY?`	`5 804 -6025 -6000 -6050 … -6025`

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

See Also

Set Compensation Frequency

Read Peak & Average Power

Read Subsequent Power Arrays

5.3.19. READ SUBSEQUENT POWER ARRAYS (ETHERNET CONTROL)

:POWER_ARRAY_EP[n]?

Follows on from Read Initial Power Array which carries out a power measurement stores a series of discrete measurements in internal memory, grouped in packages of 160 measurements. The initial query returns the number of packages (p), the total number of measurements (m) and the first 160 measured values are returned with the initial query.

If p is greater than 1 (ie: more than 160 discrete measurement values were stored) then this Read Subsequent Power Arrays query should be used to iteratively return each subsequent package of 160 measurement values.

Once all packages have been obtained, the full array of discrete power measurement values can be used for calculated analysis of the measured input signal, including pulse width, crest factor, rise / fall time and duty cycle for example.

Applies To

PWR-xP Series - Peak & average power sensors

Parameters

Variable	Description
[n]	The package index from 1 to m. The total number of packets (m) and the initial package (with index 0) is returned by the Read Initial Power Array query.

Return String

[val_0] [val_1] [val_2] …[val_159]

Variable	Description
[val_*n*]	Series of discrete, absolute power measurements (dBm), each multiplied by 100 to give an integer value

Examples

Read Initial Power Array indicated that the power measurement is split over 5 packages, containing 804 discrete values, and returned the first 160 values (package 0). The remaining 4 packages of data must therefore be requested. Packages 1 to 3 contain 160 data points each and the final package contains the final 4 data points.

String to Send	String Returned
:POWER_ARRAY?	5 804 -6025 -6000 -5975 … -5025
:POWER_ARRAY_EP1?	-5050 -5075 -5075 … -6000
:POWER_ARRAY_EP2?	-6025 -6000 -5975 … -5025
:POWER_ARRAY_EP3?	-5050 -5075 -5075 … -6000
:POWER_ARRAY_EP4?	-6025 -6050 -6075 -6075

See Also

Read Peak & Average Power

Read Initial Power Array

6. Contact

Mini-Circuits

13 Neptune Avenue

Brooklyn, NY 11235

Phone: +1-718-934-4500

Email: sales@minicircuits.com

Web: www.minicircuits.com

Important Notice

Trademarks