Important Notice

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

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

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

Trademarks

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

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

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

Mini-Circuits
13 Neptune Avenue
Brooklyn, NY 11235, USA
Phone: +1-718-934-4500
Email: sales@minicircuits.com
Web: www.minicircuits.com
1 - Overview .............................................................................................................. 6
  1.1 - Control Options .............................................................................................. 6
2 - Mini-Circuits' Cascadable Multi-Channel Attenuator Concept ..................... 7
  2.1 - Multi-Channel Attenuator Rack Structure .................................................. 7
  2.2 - Cascading Multiple Racks ............................................................................ 8
3 - Mini-Circuits' Mesh Network Concept .......................................................... 10
  3.1 - Mesh Network Rack Structure ..................................................................... 11
  3.2 - Addressing Attenuator Channels .................................................................. 12
    3.2 (a) - Example Address Structure for a 4-Port Mesh Network ...................... 12
    3.2 (b) - Example Address Structure for a 6-Port Mesh Network ...................... 13
4 - SCPI Commands for Control of Attenuator Racks ........................................ 14
  4.1 - SCPI Command Structure .......................................................................... 14
  4.2 - Daisy-Chain Configuration Commands / Queries ..................................... 14
    4.2 (a) - Assign Addresses ............................................................................... 15
    4.2 (b) - Count Number of Slaves ..................................................................... 16
  4.3 - System Identification Queries .................................................................... 17
    4.3 (a) - Get Model Name ................................................................................ 17
    4.3 (b) - Get Serial Number ............................................................................. 18
    4.3 (c) - Get Firmware .................................................................................... 19
  4.4 - Attenuator Control ...................................................................................... 20
    4.4 (a) - Set Attenuation .................................................................................. 21
    4.4 (b) - Read Attenuation .............................................................................. 22
    4.4 (c) - Set Channel Start-Up Attenuation ..................................................... 23
    4.4 (d) - Get Channel Start-Up Attenuation ..................................................... 24
  4.5 - Ethernet Configuration Commands ............................................................ 25
    4.5 (a) - Set Static IP Address .......................................................................... 25
    4.5 (b) - Get Static IP Address ........................................................................ 26
    4.5 (c) - Set Static Subnet Mask ...................................................................... 27
    4.5 (d) - Get Static Subnet Mask .................................................................... 28
    4.5 (e) - Set Static Network Gateway .............................................................. 29
    4.5 (f) - Get Static Network Gateway .............................................................. 30
    4.5 (g) - Set HTTP Port .................................................................................. 31
    4.5 (h) - Get HTTP Port .................................................................................. 32
    4.5 (i) - Set Telnet Port ................................................................................... 33
    4.5 (j) - Get Telnet Port ................................................................................... 34
    4.5 (k) - Set Password Requirement ............................................................... 35
    4.5 (l) - Get Password Requirement ............................................................... 36
    4.5 (m) - Set Password .................................................................................... 37
    4.5 (n) - Get Password .................................................................................... 38
    4.5 (o) - Set DHCP Status ............................................................................... 39
    4.5 (p) - Get DHCP Status ............................................................................... 40
    4.5 (q) - Get MAC Address ............................................................................. 41
    4.5 (r) - Get Current Ethernet Configuration ................................................ 42
    4.5 (s) - Update Ethernet Settings .................................................................... 43
5 - Ethernet Control over IP Networks ................................................................. 44
  5.1 - Configuring Ethernet Settings ..................................................................... 44
  5.2 - Ethernet Communication ............................................................................ 45
    5.2 (a) - Sending SCPI Commands / Queries Using HTTP .............................. 45
5.2 (b) - Sending SCPI / Commands/Queries Using Telnet .......................................................... 46
5.3 - Device Discovery Using UDP ................................................................................................. 47

6 - USB Control in a Windows Environment ............................................................................. 49
6.1 - The DLL (Dynamic Link Library) Concept ........................................................................... 49
  6.1 (a) - ActiveX COM Object ....................................................................................................... 50
  6.1 (b) - Microsoft.NET Class Library ......................................................................................... 52
6.2 - Referencing the DLL (Dynamic Linked Library) ................................................................. 53
6.3 - Summary of DLL Functions ................................................................................................. 54
  6.3 (a) - DLL Functions for USB Control ..................................................................................... 54
  6.3 (b) - DLL Functions for Ethernet Configuration ..................................................................... 54
6.4 - DLL Functions for USB Control .......................................................................................... 55
  6.4 (a) - Connect by Serial Number ............................................................................................ 55
  6.4 (b) - Connect by Address ......................................................................................................... 56
  6.4 (c) - Disconnect ..................................................................................................................... 57
  6.4 (d) - Send SCPI Command ...................................................................................................... 58
6.5 - DLL Functions for Ethernet Configuration ........................................................................... 59
  6.5 (a) - Get Ethernet Configuration ............................................................................................ 59
  6.5 (b) - Get IP Address ................................................................................................................ 61
  6.5 (c) - Get MAC Address ........................................................................................................... 63
  6.5 (d) - Get Network Gateway ................................................................................................... 65
  6.5 (e) - Get Subnet Mask .............................................................................................................. 67
  6.5 (f) - Get TCP/IP Port ............................................................................................................... 69
  6.5 (g) - Get DHCP Status ........................................................................................................... 70
  6.5 (h) - Get Password Status ........................................................................................................ 71
  6.5 (i) - Get Password .................................................................................................................. 72
  6.5 (j) - Save IP Address ............................................................................................................... 73
  6.5 (k) - Save Network Gateway .................................................................................................. 74
  6.5 (l) - Save Subnet Mask ............................................................................................................ 75
  6.5 (m) - Save TCP/IP Port ............................................................................................................ 76
  6.5 (n) - Use DHCP ...................................................................................................................... 77
  6.5 (o) - Use Password ................................................................................................................. 78
  6.5 (p) - Set Password .................................................................................................................. 79

7 - USB Control Using Interrupt Codes ..................................................................................... 80
7.1 - USB Interrupt Code Concept ............................................................................................... 80
7.2 - Summary of Interrupt Commands ........................................................................................ 81
  7.2 (a) - Send SCPI Command ....................................................................................................... 81

8 - Programming Examples ........................................................................................................... 83
8.1 - Visual Basic (VB) Programming .......................................................................................... 83
  8.1 (a) - USB Connection Using the .NET DLL .......................................................................... 83
  8.1 (b) - Ethernet HTTP Connection .......................................................................................... 84
8.2 - Python Programming ............................................................................................................. 85
  8.2 (a) - Ethernet Connection Using Python’s urllib2 Library for HTTP ................................. 85
  8.2 (b) - USB Connection Using the ActiveX DLL (32-bit Python Distributions) .................... 86
  8.2 (c) - Work-Around for 64-bit Python Distributions with a USB Connection ....................... 87
8.3 - Perl Programming ................................................................................................................ 88
  8.3 (a) - Ethernet Connection Using Perl’s LWP Simple Interface for HTTP ......................... 88
  8.3 (b) - USB Connection Using the ActiveX DLL (32-bit Perl Distributions) ......................... 89
  8.3 (c) - Work-Around for 64-bit Perl Distributions with a USB Connection ......................... 90
8.4 - C# Programming .................................................................................................................. 91
8.4 (a) - Creating an Executable Using the .Net DLL in C# for USB Control ........................................ 91
1 - Overview

This programming manual is intended for customers wishing to create their own interface for Mini-Circuits’ USB & Ethernet controlled, multi-channel attenuator and mesh network racks. For instructions on using the supplied GUI program, or connecting the hardware, please see the User Guide at:

Mini-Circuits offers support over a variety of operating systems, programming environments and third party applications. Support for Windows® operating systems is provided through the Microsoft®.NET® and ActiveX® frameworks to allow the user to develop customized control applications. Support for Linux® operating systems is accomplished using USB interrupts and the standard libhid and libusb libraries.

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

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

The software package includes a GUI program, ActiveX and .NET DLL files, Linux support, project examples for third party software, and detailed user manuals. The latest package is available for download at:
http://www.minicircuits.com/support/software_download.html

For details on individual models, application notes, GUI installation instructions and user guides please see:
http://www.minicircuits.com/products/PortableTestEquipment.shtml

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 Options

Communication with the system can use any of the following approaches:
1. Using HTTP or Telnet communication over an Ethernet connection (see Ethernet Control over IP Networks), which is largely independent of the operating system
2. Using the provided ActiveX or .Net API objects (DLL files) on a Windows operating system (see USB Control in a Windows Environment)
3. Using USB interrupt codes on Unix based operating systems (see USB Control Using Interrupt Codes)

In all cases the full functionality of the system is accessible using a command set based on SCPI (see SCPI Commands for Control of Attenuator Racks).
2 - Mini-Circuits' Cascadable Multi-Channel Attenuator Concept

The cascadable multi-channel attenuator concept allows multiple attenuator channels within a rack to be controlled through a single USB or Ethernet interface from a single software application. For larger systems, multiple attenuator racks can be easily cascaded in a Master / Slave configuration via their serial data interfaces. The controller treats the separate attenuator racks as if they are a single system, with each attenuator channel individually addressable and controlled via the USB or Ethernet interface of the Master rack.

2.1 - Multi-Channel Attenuator Rack Structure

The programmable attenuators within a rack are grouped in blocks of 4 attenuator channels. Each block (named as RS4DAT on the block diagram) is accessible through its 2 digit address, from 01 to nn. The 4 channels within each "RS4DAT" block can be individually controlled using their channel numbers, from 1 to 4, so the command :03:CHAN:2:SETATT:12.75 for example sets attenuator channel 2 in block 3 to 12.75 dB (the path between ports A11 and B11 in figure 1).

The controller in a multi-channel attenuator rack has address 00 although this can be omitted when sending commands / queries to the controller in a standalone multi-channel attenuator rack, or when sending to the Master controller in a series of cascaded multi-channel attenuator racks.
2.2 - Cascading Multiple Racks

When multiple attenuator racks are cascaded via their serial interfaces the structure is as if all of the 4-channel attenuator blocks are directly connected in series from the controller in the Master rack. A typical block diagram for a cascaded system is shown in figure 2. The software control connection is via the USB or Ethernet connection of the Master rack. The USB and Ethernet connections of any cascaded Slave racks should not be used.

![Block Diagrams Showing Control Connections for 2 Cascaded ZTDAT-16-6G95A Attenuator Racks](image)

The internal addressing of the Master rack is exactly the same as if the rack was being used standalone, the controller takes address 00 and the 4-channel attenuator blocks take the sequential addresses from 01.

The addressing of any Slave racks follows on sequentially from the Master, with the controller of the Slave always taking the next available address, followed by any 4-channel attenuator blocks.

There is no need to designate any racks as Master or Slave. The rack connected by USB or Ethernet will automatically assume the Master role and any racks cascaded from it via the SPI interfaces will automatically become Slaves.

Extending the example of figure 2 with a third 16-channel attenuator rack would lead to an address structure as shown in the table of figure 3 below.
### Address Structure for 3 Cascaded ZTDAT-16-6G95A Attenuator Racks

All 48 attenuator channels within the cascaded chain of 3 ZTDAT-16-6G95A racks can be controlled individually by sending commands to the Master. The commands to set each of the 48 individual attenuators are summarized in figure 4.

<table>
<thead>
<tr>
<th>#</th>
<th>Set Channel A</th>
<th>Set Channel B</th>
<th>Set Channel C</th>
<th>Set Channel D</th>
</tr>
</thead>
</table>

**Fig 3** - Address Structure for 3 Cascaded ZTDAT-16-6G95A Attenuator Racks

**Fig 4** - Table of Addresses & Commands to Set the 48 Attenuator Channels of 3 Cascaded ZTDAT-16-6G95A Racks
3 - Mini-Circuits' Mesh Network Concept

Mini-Circuit's ZTMN Series mesh network test systems allow multiple devices to be interconnected, with an individually controllable programmable attenuator on each path. The system allows every device to communicate with every other device and the loss between any pair of devices to be varied without affecting the loss between any other pair of devices. A typical application of such a mesh is to simulate the effects of interconnected wireless devices moving in and out of range of each other.

![Diagram of a 4-port mesh network showing programmable attenuators on each path.](image)

*Fig 5 - Two Alternative Representations of a 4-Port Mesh Network, Showing the Programmable Attenuators on Each Path*
3.1 - Mesh Network Rack Structure

ZTMN Series mesh networks are formed by integrating passive splitter / combiners with Mini-Circuits’ 4-channel ”RS4DAT” programmable attenuator blocks. Since the splitter / combiners are passive, the control interface for the ZTMN mesh networks is exactly the same as the ZTDAT multi-channel attenuator series. The signal loss / attenuation between any pair of ports can be independently controlled by setting the attenuator on that path.

The map of which attenuator relates to which pair of ports can be derived by referring to the internal system block diagram for each ZTMN model. The example below shows a 4-port mesh network (ZTMN-0495AS), comprising a pair or 4-channel RS4DAT programmable attenuator blocks and 4 separate 3-way splitter / combiners. Note that only 3 channels from each programmable attenuator block is required for this configuration.

![Internal System Block Diagram of ZTMN-0495AS (4-Port Mesh Network) Showing Attenuator Channel Structure](image)

<table>
<thead>
<tr>
<th>4-Channel Attenuator</th>
<th>Attenuator Channels for Each Mesh Network Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS4DAT #1</td>
<td>Channel 1 (A) Port B ↔ Port C</td>
</tr>
<tr>
<td></td>
<td>Channel 2 (B) Port A ↔ Port B</td>
</tr>
<tr>
<td></td>
<td>Channel 3 (C) Port A ↔ Port D</td>
</tr>
<tr>
<td></td>
<td>Channel 4 (D) No connection</td>
</tr>
<tr>
<td>RS4DAT #2</td>
<td>Channel 1 (A) Port C ↔ Port D</td>
</tr>
<tr>
<td></td>
<td>Channel 2 (B) Port B ↔ Port D</td>
</tr>
<tr>
<td></td>
<td>Channel 3 (C) Port A ↔ Port C</td>
</tr>
<tr>
<td></td>
<td>Channel 4 (D) No connection</td>
</tr>
</tbody>
</table>

*Fig 6 - Internal System Block Diagram of ZTMN-0495AS (4-Port Mesh Network) Showing Attenuator Channel Structure*
3.2 - Addressing Attenuator Channels

The controller in the mesh network rack has address 00 although this can be omitted when sending commands / queries directly to the controller, for example to query the model name or serial number of the system.

Each of the 4-channel attenuator blocks (named RS4DAT internally) is accessible through its 2 digit address, from 01 to nn. The 4 channels within each block can be individually controlled using their channel numbers, from 1 (channel A) to 4 (channel D) As an example, the command 

:02:CHAN:1:SETATT:12.75

sets attenuator channel 1 in RS4DAT block 2 to 12.75 dB; this is the attenuator on the unique path between ports C and D in figure 6.

3.2 (a) - Example Address Structure for a 4-Port Mesh Network

Figure 7 summarises the address structure for the 4-port mesh network of figure 6 and the commands to set each individual attenuator channel. Note that the system comprises two 4-channel attenuator blocks (for 8 channels total) although only 6 channels are in use.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Controller</td>
</tr>
<tr>
<td>01</td>
<td>4-channel attenuator (RS4DAT) block 1</td>
</tr>
<tr>
<td>02</td>
<td>4-channel attenuator (RS4DAT) block 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Set Channel 1 (A)</th>
<th>Set Channel 2 (B)</th>
<th>Set Channel 3 (C)</th>
<th>Set Channel 4 (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>:01:CHAN:1:SETATT:x</td>
<td>:01:CHAN:2:SETATT:x</td>
<td>:01:CHAN:3:SETATT:x</td>
<td>N/A</td>
</tr>
<tr>
<td>02</td>
<td>:02:CHAN:1:SETATT:x</td>
<td>:02:CHAN:2:SETATT:x</td>
<td>:02:CHAN:3:SETATT:x</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Fig 7 - Address / Channel Structure for 4-Port Mesh Network Comprising Two 4-Channel Attenuator Blocks (6 Channels in Use)*
3.2 (b) - Example Address Structure for a 6-Port Mesh Network

![Diagram of a 6-Port Mesh Network](image)

*Fig 8- Representation of a 6-Port Mesh Network, Comprising Six 5-Way Splitters / Combiners and Fifteen Attenuator Channels*

Figure 9 summarises the address structure for a 6-port mesh network and the commands to set each individual attenuator channel. Note that the system comprises four 4-channel attenuator blocks (for 16 channels total) although only 15 channels are in use.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Controller</td>
</tr>
<tr>
<td>01</td>
<td>4-channel attenuator (RS4DAT) block 1</td>
</tr>
<tr>
<td>02</td>
<td>4-channel attenuator (RS4DAT) block 2</td>
</tr>
<tr>
<td>03</td>
<td>4-channel attenuator (RS4DAT) block 3</td>
</tr>
<tr>
<td>04</td>
<td>4-channel attenuator (RS4DAT) block 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Set Channel 1 (A)</th>
<th>Set Channel 2 (B)</th>
<th>Set Channel 3 (C)</th>
<th>Set Channel 4 (D)</th>
</tr>
</thead>
</table>

*Fig 9 - Address / Channel Structure for 6-Port Mesh Network Comprising Four 4-Channel Attenuator Blocks (15 Channels in Use)*
4 - SCPI Commands for Control of Attenuator Racks

The control method for the multi-channel attenuator systems is based on a series of SCPI commands. SCPI (Standard Commands for Programmable Instruments) is a common method for controlling instrumentation products.

4.1 - SCPI Command Structure

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

The command structure takes the below form:

```
[:address]:[command]
```

Where:

- **[address]**
  - The 2 digit address of the module to command or query
  - Address can be omitted if the command is to the Master (00)
  - Address "SL" can be used to send a command to all Slaves

- **[command]**
  - The command or query to send to the attenuator system

These commands and queries can be sent using a number of methods:

1. Using HTTP or Telnet communication over an Ethernet connection (see Ethernet Control over IP Networks), which is largely independent of the operating system
2. Using the provided ActiveX or .Net API objects (DLL files) on a Windows operating system (see USB Control in a Windows Environment)
3. Using USB interrupt codes on Unix based operating systems (see USB Control Using Interrupt Codes)

4.2 - Daisy-Chain Configuration Commands / Queries

These commands / queries can be used when multiple ZTDAT multi-channel attenuator racks are daisy-chained together via their Serial Out and Serial In connectors to form a single control system. No address component is required since these commands are only issued to the Masters controller.

These commands are only required when cascading multiple multi-channel attenuator racks so do not apply to ZTMN series mesh networks.

<table>
<thead>
<tr>
<th>Description</th>
<th>Command/Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>a Assign Address</td>
<td>:AssignAddresses</td>
</tr>
<tr>
<td>b Count Number of Slaves</td>
<td>:NumberOfSlaves?</td>
</tr>
</tbody>
</table>
4.2 (a) - Assign Addresses

Description

The Master will automatically detect and issue addresses to all connected Slave units as soon as the USB connection to the Master is initiated. If any changes are made to the SPI connections between Master and Slaves after this point then the AssignAddresses command should be used to reissue addresses to all connected Slave switch modules.

Note: addresses are always issued in the order that Slaves are connected to the Master through the SPI ports. Therefore, changing the order of connection of Slave units also changes their addresses.

This command is only required when cascading multiple multi-channel attenuator racks so does not apply to ZTMN series mesh networks.

Command Syntax

:AssignAddresses

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>Indication of whether the command was executed successfully</td>
</tr>
</tbody>
</table>

Examples

HTTP Implementation: http://10.10.10.10/:AssignAddresses

See Also

Count Number of Slaves
4.2 (b) - Count Number of Slaves

Description

Used for daisy-chained ZTDAT multi-channel attenuator racks to identify the number of RS$DAT 4-channel attenuator blocks within the complete system. The count returned is the total number of Slaves in the system, including the controller in any ZTDAT slave units and any RSDAT 4-channel attenuator blocks.

See Cascading Multiple Racks for details on the address structure. The example of figure 2 would return a response of 9 to the :NumberOfSlaves? query since there are 4 RS4DAT attenuator blocks in each of the daisy-chained systems, plus the controller in the Slave ZTDAT system.

The System Identification Queries can subsequently be used to determine which module is connected at each address.

This command is only required when cascading multiple multi-channel attenuator racks so does not apply to ZTMN series mesh networks.

Command Syntax

:NumberOfSlaves?

Return String

[count]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[count]</td>
<td>The total number of Slaves connected into the Master</td>
</tr>
</tbody>
</table>

Examples

HTTP Implementation:  
http://10.10.10.10/:NumberOfSlaves?

See Also

Assign Addresses
4.3 - System Identification Queries

These queries provide a means of identifying the attenuator rack at a specific address. If the address is omitted then the queries will apply to the Master.

<table>
<thead>
<tr>
<th>Description</th>
<th>Command/Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>a Get Model Name</td>
<td>:[address]:MN?</td>
</tr>
<tr>
<td>b Get Serial Number</td>
<td>:[address]:SN?</td>
</tr>
<tr>
<td>c Get Firmware</td>
<td>:[address]:FIRMWARE?</td>
</tr>
</tbody>
</table>

4.3 (a) - Get Model Name

Description

Returns the Mini-Circuits part number of the addressed multi-channel attenuator rack. Individual 4-channel attenuator blocks within the rack will also respond to the query if addressed, with an internal part number.

Command Syntax

: [address] :MN?

Return String

: [address]: [model]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[model]</td>
<td>Model name of the multi-channel attenuator rack</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MN?</td>
<td>00:ZTDAT-16-6G95A</td>
</tr>
<tr>
<td>:00:MN?</td>
<td>00:ZTDAT-16-6G95A</td>
</tr>
<tr>
<td>:01:MN?</td>
<td>01:RS4DAT-6G-95</td>
</tr>
<tr>
<td>:05:MN?</td>
<td>05:ZTDAT-16-6G95A</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:MN?

See Also

Get Serial Number
4.3 (b) - Get Serial Number

Description

Returns the serial number of the addressed multi-channel attenuator rack. Individual 4-channel attenuator blocks within the rack will also respond to the query if addressed, with an internal serial number.

Command Syntax

: [address] : SN?

Return String

:[address]:[serial]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[serial]</td>
<td>Serial number of the multi-channel attenuator rack</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SN?</td>
<td>00:11612010001</td>
</tr>
<tr>
<td>:00:SN?</td>
<td>00:11612010001</td>
</tr>
<tr>
<td>:05:SN?</td>
<td>01:11612010002</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:SN?

See Also

Get Model Name
4.3 (c) - Get Firmware

**Description**

Returns the firmware version of the addressed multi-channel attenuator rack. Individual 4-channel attenuator blocks within the rack will also respond to the query if addressed, with an internal firmware version.

**Command Syntax**

```
:[address]:FIRMWARE?
```

**Return String**

```
:[address]:[firmware]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[firmware]</td>
<td>Firmware version name</td>
</tr>
</tbody>
</table>

**Examples**

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:FIRMWARE?</td>
<td>00:A1</td>
</tr>
<tr>
<td>:00:FIRMWARE?</td>
<td>00:A1</td>
</tr>
<tr>
<td>:01:FIRMWARE?</td>
<td>01:A1</td>
</tr>
</tbody>
</table>

**HTTP Implementation:**

http://10.10.10.10/:FIRMWARE?
4.4 - Attenuator Control

These commands and queries allow control of a specific attenuator channel within the system. The commands are issued in the below format:

`: [address]: CHAN: [channel]: [command]: [value]`

- **[address]**
  - The 2 digit address of the 4-channel attenuator block that contains the channel to be controlled
  - Address "SL" can be used to send a command to all Slaves
- **[channel]**
  - The channel number within the block of 4 attenuators
- **[command]**
  - The command / query to send
- **[value]**
  - The value to set (if applicable)

<table>
<thead>
<tr>
<th>Description</th>
<th>Command/Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>a Set Attenuation</td>
<td>:[address]: CHAN: [channels]: SETATT: [att]</td>
</tr>
<tr>
<td>b Read Attenuition</td>
<td>:[address]: CHAN: [channel]: ATT?</td>
</tr>
<tr>
<td>c Set Channel Start-Up Attenuation</td>
<td>:[address]: CHAN: [channels]: STARTUPATT: VALUE: [att]</td>
</tr>
<tr>
<td>d Get Channel Start-Up Attenuation</td>
<td>:[address]: CHAN: [channel]: STARTUPATT: VALUE?</td>
</tr>
</tbody>
</table>
4.4 (a) - Set Attenuation

Description

Sets the attenuation of a single channel or combination of channels. Use address “SL” to send the command to all 4-channel attenuator blocks.

Command Syntax

: [address]:CHAN:[channels]:SETATT:[Att]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[channels]</td>
<td>The attenuator channel number within the 4-channel attenuator block from 1 (channel A) to 4 (channel D). Multiple channels can be sent by listing each channel number separated by a colon</td>
</tr>
<tr>
<td>[Att]</td>
<td>The attenuation to set</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0 - Failed</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1 - Success</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>Channels</th>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS4DAT 1, channel 1</td>
<td>:01:CHAN:1:SETATT:10.25</td>
<td>1 - Success</td>
</tr>
<tr>
<td>RS4DAT 1, channels 1-4</td>
<td>:01:CHAN:1:2:3:4:SETATT:10.25</td>
<td>1 - Success</td>
</tr>
<tr>
<td>RS4DAT 2, channel 1</td>
<td>:02:CHAN:1:SETATT:0</td>
<td>1 - Success</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:01:CHAN:1:SETATT:10.25

See Also

Read Attenuation
4.4 (b) - Read Attenuation

Description

Returns the attenuation of a single channel

Command Syntax

`: [address] : CHAN: [channel] : ATT?`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[channel]</td>
<td>The attenuator channel number within the 4-channel attenuator block from 1 (channel A) to 4 (channel D)</td>
</tr>
</tbody>
</table>

Return String

[att]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[att]</td>
<td>The attenuation (dB) of the requested channel</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:01:CHAN:1:ATT?</td>
<td>10.25</td>
</tr>
<tr>
<td>:01:CHAN:2:ATT?</td>
<td>20.50</td>
</tr>
<tr>
<td>:01:CHAN:3:ATT?</td>
<td>30.75</td>
</tr>
<tr>
<td>:01:CHAN:4:ATT?</td>
<td>40.0</td>
</tr>
<tr>
<td>:02:CHAN:1:ATT?</td>
<td>0.0</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:01:CHAN:1:ATT?

See Also

Set Attenuation
4.4 (c) - Set Channel Start-Up Attenuation

Description

Sets the start-up attenuation value for a single attenuator channel (the value to be set when the system is first powered on. Use address “SL” to send the command to all 4-channel attenuator blocks.

Command Syntax

: [address]:CHAN: [channels]:STARTUPATT:VALUE: [att]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[channels]</td>
<td>The attenuator channel number within the 4-channel attenuator block from 1 (channel A) to 4 (channel D). Multiple channels can be sent by listing each channel number separated by a colon</td>
</tr>
<tr>
<td>[att]</td>
<td>The initial attenuation level to be loaded on start-up</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>Channels</th>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS4DAT 1, channel 1</td>
<td>:01:CHAN:1:STARTUPATT:VALUE:0</td>
<td>1</td>
</tr>
<tr>
<td>RS4DAT 1, channels 1-4</td>
<td>:01:CHAN:1:2:3:4:STARTUPATT:VALUE:0</td>
<td>1</td>
</tr>
<tr>
<td>All channels</td>
<td>:SL:CHAN:1:2:3:4:STARTUPATT:VALUE:0</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation: http://10.10.10.10/:01:CHAN:1:STARTUPATT:VALUE:0

See Also

Get Channel Start-Up Attenuation
4.4 (d) - Get Channel Start-Up Attenuation

Description

Returns the start-up attenuation value for a single attenuator channel (the value to be set when the system is first powered on).

Command Syntax

`: [address] : CHAN : [channel] : STARTUPATT : VALUE ?`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[channel]</td>
<td>The attenuator channel number within the 4-channel attenuator block from 1 (channel A) to 4 (channel D)</td>
</tr>
</tbody>
</table>

Return String

[att]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[att]</td>
<td>The initial attenuation value (dB) of the requested channel</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:01:CHAN:1:STARTUPATT:VALUE?</td>
<td>12.75</td>
</tr>
<tr>
<td>:01:CHAN:4:STARTUPATT:VALUE?</td>
<td>12.75</td>
</tr>
</tbody>
</table>

HTTP Implementation:

`http://10.10.10.10/:01:CHAN:1:STARTUPATT:VALUE?`

See Also

Set Channel Start-Up Attenuation
4.5 - Ethernet Configuration Commands

These functions provide a method of configuring the system’s Ethernet IP settings, while connected via Ethernet or USB.

4.5 (a) - Set Static IP Address

Description

Sets the IP address to be used for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

```
:ETHERNET:CONFIG:IP: [ip]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ip]</td>
<td>The static IP address to be used; must be valid and available on the network</td>
</tr>
</tbody>
</table>

Return String

```
[status]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:IP:192.100.1.1</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:IP:192.100.1.1
```

See Also

- Get Static IP Address
- Set Static Subnet Mask
- Set Static Network Gateway
- Update Ethernet Settings
4.5 (b) - Get Static IP Address

Description

Returns the IP address to be used for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

Command Syntax

:ETHERNET:CONFIG:IP?

Return String

[ip]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ip]</td>
<td>The static IP address to be used</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:IP?</td>
<td>192.100.1.1</td>
</tr>
</tbody>
</table>

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:IP?

See Also

Set Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration
4.5 (c) - Set Static Subnet Mask

Description

Sets the subnet mask to be used for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

:ETHERNET:CONFIG:SM: [mask]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mask]</td>
<td>The subnet mask for communication on the network</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:SM:255.255.255.0</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:SM:255.255.255.0

See Also

Set Static IP Address
Get Static Subnet Mask
Set Static Network Gateway
Update Ethernet Settings
4.5 (d) - Get Static Subnet Mask

Description

Returns the subnet mask to be used for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

Requirements

Please contact testsolutions@minicircuits.com.

Return String

[mask]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mask]</td>
<td>The subnet mask for communication on the network</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:SM?</td>
<td>255.255.255.0</td>
</tr>
</tbody>
</table>

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:SM?

See Also

Get Static IP Address
Set Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration
4.5 (e) - Set Static Network Gateway

Description

Sets the IP address of the network gateway to be used for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

`:ETHERNET:CONFIG:NG:[gateway]`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[gateway]</td>
<td>IP address of the network gateway</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:ETHERNET:CONFIG:NG:192.100.1.0</code></td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:NG:192.168.100.1.0
```

See Also

- Set Static IP Address
- Set Static Subnet Mask
- Get Static Network Gateway
- Update Ethernet Settings
4.5 (f) - Get Static Network Gateway

Description

Returns the IP address of the network gateway to be used for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

Command Syntax

:ETHERNET:CONFIG:NG?

Return String

[gateway]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[gateway]</td>
<td>IP address of the network gateway</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:NG?</td>
<td>192.168.1.0</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:NG?

See Also

- Get Static IP Address
- Get Static Subnet Mask
- Set Static Network Gateway
- Get Current Ethernet Configuration
4.5 (g) - Set HTTP Port

Description

Sets the IP port to be used for HTTP communication. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

:ETHERNET:CONFIG:HTPORT: [port]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[port]</td>
<td>IP port to be used for HTTP communication. The port will need to be included in all HTTP commands if any other than the default port 80 is selected.</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:HTPORT:8080</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:HTPORT:8080

See Also

Get HTTP Port  
Set Telnet Port  
Update Ethernet Settings
4.5 (h) - Get HTTP Port

Description

Gets the IP port to be used for HTTP communication.

Command Syntax

:ETHERNET:CONFIG:HTPORT?

Return String

[port]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[port]</td>
<td>IP port to be used for HTTP communication</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:HTPORT?</td>
<td>8080</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:HTPORT?

See Also

Set HTTP Port
Get Telnet Port
4.5 (i) - Set Telnet Port

Description

Sets the IP port to be used for Telnet communication. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

:ETHERNET:CONFIG:TELNETPORT: [port]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[port]</td>
<td>IP port to be used for Telnet communication. The port will need to be included when initiating a Telnet session if other than the default port 23 is selected.</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:TELNETPORT:21</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:TELNETPORT:21

See Also

Set HTTP Port
Get Telnet Port
Update Ethernet Settings
4.5 (j) - Get Telnet Port

Description

Gets the IP port to be used for Telnet communication.

Command Syntax

```plaintext
:ETHERNET:CONFIG:TELNETPORT?
```

Return String

```
[port]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[port]</td>
<td>IP port to be used for Telnet communication</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:TELNETPORT?</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:TELNETPORT?
```

See Also

- Get HTTP Port
- Set Telnet Port
4.5 (k) - Set Password Requirement

Description

Sets whether or not a password is required for Ethernet communication. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

:ETHERNET:CONFIG:PWDENABLED: \[enabled\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[enabled]</td>
<td>0</td>
<td>Password not required for Ethernet communication</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Password required for Ethernet communication</td>
</tr>
</tbody>
</table>

Return String

\[status\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:PWDENABLED:1</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:PWDENABLED:1

See Also

Get Password Requirement
Set Password
Get Password
Update Ethernet Settings
4.5 (l) - Get Password Requirement

Description

 Indicates whether or not a password is required for Ethernet communication.

Command Syntax

:ETHERNET:CONFIG:PWDENABLED?

Return String

(enabled)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>0</td>
<td>Password not required for Ethernet communication</td>
</tr>
<tr>
<td>enabled</td>
<td>1</td>
<td>Password required for Ethernet communication</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:PWDENABLED?</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:PWDENABLED?

See Also

Set Password Requirement
Set Password
Get Password
4.5 (m) - Set Password

Description

Sets the password for Ethernet communication. The password will only be required for communication with the device when password security is enabled. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

```plaintext
:ETHERNET:CONFIG:PWD: [pwd]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pwd]</td>
<td>Password to set for Ethernet communication (not case sensitive)</td>
</tr>
</tbody>
</table>

Return String

```plaintext
[status]
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:PWD:PASS-123</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:PWD:PASS-123

See Also

Set Password Requirement
Get Password Requirement
Get Password
Update Ethernet Settings
4.5 (n) - Get Password

Description

Returns the password for Ethernet communication. The password will only be required for communication with the device when password security is enabled.

Command Syntax

:ETHERNET:CONFIG:PWD?

Return String

[pwd]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pwd]</td>
<td>Password for Ethernet communication (not case sensitive)</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:PWD?</td>
<td>PASS-123</td>
</tr>
</tbody>
</table>

HTTP Implementation:  

http://10.10.10.10/:ETHERNET:CONFIG:PWD?

See Also

Set Password Requirement  
Get Password Requirement  
Set Password
4.5 (o) - Set DHCP Status

Description

Enables or disables DHCP (Dynamic Host Control Protocol). When enabled the system will request a valid IP address from the network's DHCP server. When disabled, the system's static IP settings will be used. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Command Syntax

:ETHERNET:CONFIG:DHCPENABLED: [enabled]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[enabled]</td>
<td>0</td>
<td>DHCP disabled (static IP settings will be used)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>DHCP enabled (IP address will be requested from DHCP server on the network)</td>
</tr>
</tbody>
</table>

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:DHCPENABLED:1</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:DHCPENABLED:1

See Also

Set Static IP Address
Get DHCP Status
Update Ethernet Settings
4.5 (p) - Get DHCP Status

Description

Indicates whether or not DHCP (Dynamic Host Control Protocol) is enabled. When enabled the system will request a valid IP address from the network's DHCP server. When disabled, the system's static IP settings will be used.

Command Syntax

:ETHERNET:CONFIG:DHCPENABLED?

Return String

[enabled]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[enabled]</td>
<td>0</td>
<td>DHCP disabled (static IP settings will be used)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>DHCP enabled (IP address will be requested from DHCP server on the network)</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:DHCPENABLED?</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:DHCPENABLED?

See Also

Set Static IP Address
Set DHCP Status
Get Current Ethernet Configuration
4.5 (q) - Get MAC Address

Description

Returns the MAC (Media Access Control) address of system attenuator (a physical hardware address).

Command Syntax

:ETHERNET:CONFIG:MAC?

Return String

[mac]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mac]</td>
<td>MAC address of the attenuator</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:MAC?</td>
<td>D0-73-7F-82-D8-01</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:MAC?

See Also

Get Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration
4.5 (r) - Get Current Ethernet Configuration

Description

Returns the Ethernet configuration (IP address, subnet mask and network gateway) that is currently active for the device. If DHCP is enabled this will be the settings issued dynamically by the network's DHCP server. If DHCP is disabled this will be the user configured static IP settings.

Command Syntax

:ETHERNET:CONFIG:LISTEN?

Return String

[ip];[mask];[gateway]

Variable | Description
--- | ---
[ip] | Active IP address of the device
[mask] | Subnet mask for the network
[gateway] | IP address of the network gateway

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:LISTEN?</td>
<td>192.100.1.1;255.255.255.0;192.100.1.0</td>
</tr>
</tbody>
</table>

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:LISTEN?

See Also

Get Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Update Ethernet Settings
4.5 (s) - Update Ethernet Settings

Description

Resets the Ethernet controller so that any recently applied changes to the Ethernet configuration can be loaded. Any subsequent commands / queries to the system will need to be issued using the new Ethernet configuration.

Note: If a connection cannot be established after the INIT command has been issued it may indicate an invalid configuration was created (for example a static IP address which clashes with another device on the network). The Ethernet settings can always be overwritten by connecting to the system using the USB connection.

Command Syntax

:ETHERNET:CONFIG:INIT

Return String

[status]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[status]</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>String to Send</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ETHERNET:CONFIG:INIT</td>
<td>1</td>
</tr>
</tbody>
</table>

HTTP Implementation:

http://10.10.10.10/:ETHERNET:CONFIG:INIT

See Also

Get Current Ethernet Configuration
5 - Ethernet Control over IP Networks

Mini-Circuits’ multi-channel attenuator racks have an RJ45 connector option for remote control over Ethernet TCP/IP networks. HTTP (Get/Post commands) and Telnet communication are supported. UDP transmission is also supported for discovering available attenuator systems on the network.

The multi-channel attenuator rack connected by Ethernet automatically becomes the Master, with any other modules cascaded via the SPI connections defined as Slaves.

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

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

Notes:
1. The TCP/IP port must be included in every HTTP command to the attenuator system unless the default port 80 is used
2. Port 23 is reserved for Telnet communication

5.1 - Configuring Ethernet Settings

The attenuator system must be connected via the USB interface in order to configure the Ethernet settings. Following initial configuration, the system can be controlled via the Ethernet interface with no further need for a USB connection. The API DLLs provide a full set of functions for configuring the Ethernet settings.
5.2 - Ethernet Communication

Communication over Ethernet can be accomplished using HTTP Get/Post commands or Telnet communication to send the SCPI commands outlined in SCPI Commands for Control of Attenuator Racks. These communication protocols are both commonly supported and simple to implement in most programming languages. Any Internet browser can be used as a console/tester for HTTP control by typing the commands/queries directly into the address bar.

5.2 (a) - Sending SCPI Commands / Queries Using HTTP

The basic format of the HTTP command to send to the multi-channel attenuator system is:

http://ADDRESS:PORT/PWD;COMMAND

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

Example 1:


Explanation:
- The attenuator system has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to “123”
- The command is to set attenuator channel 1 in block 1

Example 2:

http://10.10.10.10/::01:ATT?

Explanation:
- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to query the attenuation state of the 4 channels in block 1

The system will return the result of the command/query as a string of ASCII characters.
5.2 (b) - Sending SCPI / Commands/Queries Using Telnet

Communication is started by creating a Telnet connection to the system’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.

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

1) Set up Telnet connection to a multi-channel attenuator system with IP address 10.0.6.46:

   ![Telnet connection](image1)

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

   ![Line feed successful](image2)

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

   ![Password successful](image3)

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

   ![Commands and queries](image4)
5.3 - Device Discovery Using UDP

In addition to HTTP and Telnet, Mini-Circuits’ multi-channel attenuator systems also provide limited support of the UDP protocol for the purpose of “device discovery.” This allows a user to request the IP address and configuration of all Mini-Circuits multi-channel attenuator systems 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 system with the USB interface (see Configuring Ethernet Settings).

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

UDP Ports

Mini-Circuits’ attenuator systems 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 test system’s IP address is already known it is not necessary to use UDP.

Transmission

The command MCL_MULTI_CHAN_CONTROLLER? should be broadcast to the local network using UDP protocol on port 4950.

Receipt

All systems 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_MULTI_CHAN_CONTROLLER?**

Received Data:

- Model Name: ZTDAT-16-6G95A
  - Serial Number: 11302120001
  - 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: ZTDAT-16-6G95A
  - Serial Number: 11302120002
  - IP Address=192.168.9.102 Port: 80
  - Subnet Mask=255.255.0.0
  - Network Gateway=192.168.9.0
  - Mac Address=D0-73-7F-82-D8-02

- Model Name: ZTDAT-16-6G95A
  - Serial Number: 11302120003
  - IP Address=192.168.9.103 Port: 80
  - Subnet Mask=255.255.0.0
  - Network Gateway=192.168.9.0
  - Mac Address=D0-73-7F-82-D8-03
6 - USB Control in a Windows Environment

6.1 - The DLL (Dynamic Link Library) Concept

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

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

Mini-Circuits' software package provides DLL objects designed to allow your own application to interface with the functions of the cascadable solid state switches.

![DLL Interface Concept](Fig 10: DLL Interface Concept)

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

1. **ActiveX com object**
   Design to be used in any programming environment that supports third party ActiveX COM (Component Object Model) compliant applications.
   The ActiveX file should be registered using RegSvr32 (see following sections for details).

2. **Microsoft.NET Class Library**
   A logical unit of functionality that runs under the control of the Microsoft.NET system.
6.1 (a) - ActiveX COM Object

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

Supported Programming Environments

Mini-Circuits’ multi-channel attenuator systems have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support ActiveX functionality. Please contact Mini-Circuits for support.

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

Installation

1. Copy the DLL file to the correct directory:
   For 32-bit Windows operating systems this is C:\WINDOWS\System32
   For 64-bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. Open the Command Prompt:
   a. For Windows XP®:
      i. Select “All Programs” and then “Accessories” from the Start Menu
      ii. Click on “Command Prompt” to open
   b. For later versions of the Windows operating system you will need to have Administrator privileges in order to run the Command Prompt in “Elevated” mode:
      i. Open the Start Menu/Start Screen and type “Command Prompt”
      ii. Right-click on the shortcut for the Command Prompt
      iii. Select “Run as Administrator”
      iv. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3. Use regsvr32 to register the DLL:
   For 32-bit Windows operating systems type (see Fig 13):
   \WINDOWS\System32\Regsvr32 \WINDOWS\System32\MCL_MultiChannelAtt.dll
   For 64-bit Windows operating systems type (see Fig 14):
   \WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\MCL_MultiChannelAtt.dll
4. Hit enter to confirm and a message box will appear to advise of successful registration.
Fig 11: Opening the Command Prompt in Windows XP

Fig 12: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

Fig 13: Registering the DLL in a 32-bit environment

Fig 14: Registering the DLL in a 64-bit environment
6.1 (b) - Microsoft.NET Class Library

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

Supported Programming Environments

Mini-Circuits’ multi-channel attenuator systems have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support Microsoft.NET functionality. Please contact Mini-Circuits for support.

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

Installation

1. Copy the DLL file to the correct directory
   a. For 32 bit Windows operating systems this is C:\WINDOWS\System32
   b. For 64 bit Windows operating systems this is C:\WINDOWS\SysWOW64

2. No registration is required
6.2 - Referencing the DLL (Dynamic Linked Library)

In order to use the DLL functionality, some programming environments will require the user to set a reference to the relevant DLL file. Once this is done, the user just needs to declare a new instance of the USB Control class (defined within the DLL) for each system to be controlled. The class is assigned to a variable which is used to call the DLL functions as needed. In the following examples, the variable names MyPTE1 and MyPTE2 have been used to represent 2 connected attenuator systems.

Example Declarations using the ActiveX DLL (MCL_MultiChannelAtt.dll)

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td>Public MyPTE1 As New MCL_MultiChannelAtt_USB_Control</td>
</tr>
<tr>
<td></td>
<td>' Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>Public MyPTE2 As New MCL_MultiChannelAtt_USB_Control</td>
</tr>
<tr>
<td></td>
<td>' Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Visual C++</td>
<td>MCL_MultiChannelAtt_USB_Control^MyPTE1 = gcnew</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_USB_Control^MyPTE1 = gcnew</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_USB_Control^MyPTE2 = gcnew</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Visual C#</td>
<td>public MCL_MultiChannelAtt_USB_Control MyPTE1 = new</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_USB_Control MyPTE1 = new</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>public MCL_MultiChannelAtt_USB_Control MyPTE2 = new</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_USB_Control MyPTE2 = new</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Matlab</td>
<td>MyPTE1 = actxserver(MCL_MultiChannelAtt_USB_Control')</td>
</tr>
<tr>
<td></td>
<td>MyPTE2 = actxserver(MCL_MultiChannelAtt_USB_Control')</td>
</tr>
</tbody>
</table>

Example Declarations using the .NET DLL (MCL_MultiChannelAtt_64.dll)

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td>Public MyPTE1 As New MCL_MultiChannelAtt_64_USB_MultiChannelAtt</td>
</tr>
<tr>
<td></td>
<td>' Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>Public MyPTE2 As New MCL_MultiChannelAtt_64_USB_MultiChannelAtt</td>
</tr>
<tr>
<td></td>
<td>' Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Visual C++</td>
<td>MCL_MultiChannelAtt_64_USB_MultiChannelAtt^MyPTE1 = gcnew</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_64_USB_MultiChannelAtt^MyPTE1 = gcnew</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_64_USB_MultiChannelAtt^MyPTE2 = gcnew</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Visual C#</td>
<td>public MCL_MultiChannelAtt_64_USB_MultiChannelAtt MyPTE1 = new</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_64_USB_MultiChannelAtt MyPTE1 = new</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE1</td>
</tr>
<tr>
<td></td>
<td>public MCL_MultiChannelAtt_64_USB_MultiChannelAtt MyPTE2 = new</td>
</tr>
<tr>
<td></td>
<td>MCL_MultiChannelAtt_64_USB_MultiChannelAtt MyPTE2 = new</td>
</tr>
<tr>
<td></td>
<td>// Declare new control object, assign to MyPTE2</td>
</tr>
<tr>
<td>Matlab</td>
<td>MCL_SW = NET.addAssembly('C:\Windows\SysWOW64\MCL_MultiChannelAtt_64.dll')</td>
</tr>
<tr>
<td></td>
<td>myPTE1 = MCL_MultiChannelAtt_64_USB_MultiChannelAtt</td>
</tr>
<tr>
<td></td>
<td>% Invoke new control object, MyPTE1</td>
</tr>
<tr>
<td></td>
<td>myPTE2 = MCL_MultiChannelAtt_64_USB_MultiChannelAtt</td>
</tr>
<tr>
<td></td>
<td>% Invoke new control object, MyPTE2</td>
</tr>
</tbody>
</table>
6.3 - Summary of DLL Functions

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

6.3 (a) - DLL Functions for USB Control

   a) Short Connect (Optional String SN)
   b) Short ConnectByAddress (Optional Short Address)
   c) Void Disconnect()
   d) Short Send_SCPI (String SndSTR, String RetSTR)

6.3 (b) - DLL Functions for Ethernet Configuration

   b) Short GetEthernet_IPAddress (Int b1, Int b2, Int b3, Int b4)
   c) Short GetEthernet_MACAddress (Int MAC1, Int MAC2, Int MAC3, Int MAC4, Int MAC5, Int MAC6)
   d) Short GetEthernet_NetworkGateway (Int b1, Int b2, Int b3, Int b4)
   e) Short GetEthernet_SubNetMask (Int b1, Int b2, Int b3, Int b4)
   f) Short GetEthernet_TCP/IPPort (Int port)
   g) Short GetEthernet_UseDHCP ()
   h) Short GetEthernet_UsePWD ()
   i) Short GetEthernet_PWD (string Pwd)
   j) Short SaveEthernet_IPAddress (Int b1, Int b2, Int b3, Int b4)
   k) Short SaveEthernet_NetworkGateway (Int b1, Int b2, Int b3, Int b4)
   l) Short SaveEthernet_SubnetMask (Int b1, Int b2, Int b3, Int b4)
   m) Short SaveEthernet_TCP/IPPort (Int port)
   n) Short SaveEthernet_UseDHCP (Int UseDHCP)
   o) Short SaveEthernet_UsePWD (Int UsePwd)
   p) Short SaveEthernet_PWD (String Pwd)
6.4 - DLL Functions for USB Control

6.4 (a) - Connect by Serial Number

Declaration

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

Description

Initializes the USB connection. If multiple attenuator racks are connected to the same host computer by USB then the serial number should be included, otherwise this can be omitted. The system should be disconnected on completion of the program using the `Disconnect` function.

The multi-channel attenuator rack connected by USB automatically becomes the Master, with any other modules cascaded via the SPI connections defined as Slaves.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SN</td>
<td>Optional. The serial number of the test system. Can be omitted if only one attenuator rack is connected by USB.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>No connection was possible</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Connection successfully established</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Connection already established (Connect has been called more than once). The system will continue to operate normally.</td>
</tr>
</tbody>
</table>

Examples

```markdown
Visual Basic
  status = MyPTE1.Connect(SN)

Visual C++
  status = MyPTE1->Connect(SN);

Visual C#
  status = MyPTE1.Connect(SN);

Matlab
  status = MyPTE1.Connect(SN)
```

See Also

- Connect by Address
- Disconnect
6.4 (b) - Connect by Address

Declaration

Short ConnectByAddress(Optional Short Address)

Description

Initialize the USB connection to a multi-channel attenuator rack by referring to a user-defined USB address. The address is an integer number from 1 to 255 which can be assigned using the Set_Address function (the factory default is 255). The system should be disconnected on completion of the program using the Disconnect function.

The attenuator rack connected by USB automatically becomes the Master, with any other modules cascaded via the SPI connections defined as Slaves.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Address</td>
<td>Optional. The address of the system. Can be omitted if only one attenuator rack is connected by USB.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>No connection was possible</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Connection successfully established</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Connection already established (Connect has been called more than once)</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

```
status = MyPTE1.ConnectByAddress(5)
```

Visual C++

```
status = MyPTE1->ConnectByAddress(5); 
```

Visual C#

```
status = MyPTE1.ConnectByAddress(5);
```

Matlab

```
status = MyPTE1.connectByAddress(5)
```

See Also

Connect by Serial Number
Disconnect
6.4 (c) - Disconnect

Declaration

`Void Disconnect()`

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td><code>MyPTE1.Disconnect()</code></td>
<td></td>
</tr>
<tr>
<td>Visual C++</td>
<td><code>MyPTE1-&gt;Disconnect();</code></td>
<td></td>
</tr>
<tr>
<td>Visual C#</td>
<td><code>MyPTE1.Disconnect();</code></td>
<td></td>
</tr>
<tr>
<td>Matlab</td>
<td><code>MyPTE1.Disconnect</code></td>
<td></td>
</tr>
</tbody>
</table>

See Also

- Connect by Serial Number
- Connect by Address
6.4 (d) - Send SCPI Command

Declaration

```
Short Send_SCPI(String SndSTR, String RetSTR)
```

Description

This function sends a SCPI command to the connected multi-channel attenuator system and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the cascadable switch system.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SndSTR</td>
<td>Required. The SCPI command to send.</td>
</tr>
<tr>
<td>String</td>
<td>RetSTR</td>
<td>Required. User defined string which will be updated with the value returned from the test system.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

```
Status = MyPTE1.Send_SCPI("00:MN?", RetStr)
' Return the model name of the Master attenuator rack
```

Visual C++

```
Status = MyPTE1->Send_SCPI("00:MN?", RetStr);
// Return the model name of the Master attenuator rack
```

Visual C#

```
Status = MyPTE1.Send_SCPI("00:MN?", RetStr);
// Return the model name of the Master attenuator rack
```

Matlab

```
Status = MyPTE1.Send_SCPI("00:MN?", RetStr)
% Return the model name of the Master attenuator rack
```

See Also

SCPI Commands for Control of Attenuator Racks
6.5 - DLL Functions for Ethernet Configuration

6.5 (a) - Get Ethernet Configuration

Declaration

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

Description

This function returns the current IP configuration of the connected multi-channel attenuator system in a series of user defined variables. The settings checked are IP address, subnet mask and network gateway.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>Required. Integer variable which will be updated with the first (highest order) octet of the IP address.</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>Required. Integer variable which will be updated with the second octet of the IP address.</td>
</tr>
<tr>
<td>Int</td>
<td>IP3</td>
<td>Required. Integer variable which will be updated with the third octet of the IP address.</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>Required. Integer variable which will be updated with the last (lowest order) octet of the IP address.</td>
</tr>
<tr>
<td>Int</td>
<td>Mask1</td>
<td>Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask.</td>
</tr>
<tr>
<td>Int</td>
<td>Mask2</td>
<td>Required. Integer variable which will be updated with the second octet of the subnet mask.</td>
</tr>
<tr>
<td>Int</td>
<td>Mask3</td>
<td>Required. Integer variable which will be updated with the third octet of the subnet mask.</td>
</tr>
<tr>
<td>Int</td>
<td>Mask4</td>
<td>Required. Integer variable which will be updated with the last (lowest order) octet of the subnet mask.</td>
</tr>
<tr>
<td>Int</td>
<td>Gateway1</td>
<td>Required. Integer variable which will be updated with the first (highest order) octet of the network gateway.</td>
</tr>
<tr>
<td>Int</td>
<td>Gateway2</td>
<td>Required. Integer variable which will be updated with the second octet of the network gateway.</td>
</tr>
<tr>
<td>Int</td>
<td>Gateway3</td>
<td>Required. Integer variable which will be updated with the third octet of the network gateway.</td>
</tr>
<tr>
<td>Int</td>
<td>Gateway4</td>
<td>Required. Integer variable which will be updated with the last (lowest order) octet of the network gateway.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example
Visual Basic

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

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

End If

Visual C++

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

Visual C#

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

Matlab

[status, IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1, GW2, GW3, GW4] = MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1, GW2, GW3, GW4)
If status > 0 then
{
    MsgBox ("IP address: ", IP1, ",", IP2, ",", IP3, ",", IP4)
    MsgBox ("Subnet Mask: ", M1, ",", M2, ",", M3, ",", M4)
    MsgBox ("Gateway: ", GW1, ",", GW2, ",", GW3, ",", GW4)
}

See Also

Get MAC Address
Get TCP/IP Port
6.5 (b) - Get IP Address

Declaration

`Short GetEthernet_IPAddress(Int b1, Int b2, Int b3, Int b4)`

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP3</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>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”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>
Example

**Visual Basic**

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

**Visual C++**

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

**Visual C#**

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

**Matlab**

```matlab
If status > 0 then
    MsgBox ("IP address: ", IP1, ",", IP2, ",", IP3, ",", IP4)
End If
```

See Also

- Get Ethernet Configuration
- Get TCP/IP Port
- Save IP Address
- Save TCP/IP Port
6.5 (c) - Get MAC Address

Declaration

*Short GetEthernet_MACAddress*(Int MAC1, Int MAC2, Int MAC3, Int MAC4, Int MAC5, Int MAC6)

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>MAC1</td>
<td>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</td>
</tr>
<tr>
<td>Int</td>
<td>MAC2</td>
<td>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</td>
</tr>
<tr>
<td>Int</td>
<td>MAC3</td>
<td>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</td>
</tr>
<tr>
<td>Int</td>
<td>MAC4</td>
<td>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</td>
</tr>
<tr>
<td>Int</td>
<td>MAC5</td>
<td>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</td>
</tr>
<tr>
<td>Int</td>
<td>MAC6</td>
<td>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</td>
</tr>
</tbody>
</table>
## Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

## Example

### Visual Basic

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

### Visual C++

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

### Visual C#

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

### Matlab

```matlab
[status, M1, M2, M3, M4, M5, M6] = MyPTE1.GetEthernet_MACAddress(M1, M2, M3, M4, M5, M6)
If status > 0 then
    MsgBox("MAC address: ", M1, ".", M2, ".", M3, ".", M4, ".", M5, ".", M6)
```

## See Also

- Get Ethernet Configuration
6.5 (d) - Get Network Gateway

Declaration

```
Short GetEthernet_NetworkGateway(Int b1, Int b2, Int b3, Int b4)
```

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP3</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>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”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>
Example

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

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

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

Matlab
If status > 0 then
{
    MsgBox("Gateway: ", IP1, ".", IP2, ".", IP3, ".", IP4)
}

See Also

Get Ethernet Configuration
Save Network Gateway
6.5 (e) - Get Subnet Mask

Declaration

```
Short GetEthernet_SubNetMask(Int b1, Int b2, Int b3, Int b4)
```

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>b1</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>b2</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>b3</td>
<td>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”).</td>
</tr>
<tr>
<td>Int</td>
<td>b4</td>
<td>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”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>
Example

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

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

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

Matlab
[status, b1, b2, b3, b4] = MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4)
If status > 0 then
    MsgBox("Subnet mask: ", b1, ".", b2, ".", b3, ".", b4)

See Also

Get Ethernet Configuration
Save Subnet Mask
6.5 (f) - Get TCP/IP Port

Declaration

\[
\text{Short GetEthernet_TCPIPPort(Int } \text{ port)}
\]

Description

This function returns the TCP/IP port used by the test system 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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>port</td>
<td>Required. Integer variable which will be updated with the TCP/IP port.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

\[
\text{If MyPTPE1.GetEthernet_SubNetMask(port) > 0 Then}
\]
\[
\text{\quad MsgBox ("Port: " & port)}
\]
\[
\text{End If}
\]

Visual C++

\[
\text{if (MyPTPE1->GetEthernet_SubNetMask(port) > 0)}
\]
\[
\text{\quad MessageBox::Show("Port: " + port);}
\]

Visual C#

\[
\text{if (MyPTPE1.GetEthernet_SubNetMask(port) > 0)}
\]
\[
\text{\quad MessageBox.Show("Port: " + port);}\]

Matlab

\[
\text{[status, port] = MyPTPE1.GetEthernet_SubNetMask(port)}
\]
\[
\text{if status > 0 then}
\]
\[
\text{\quad MsgBox ("Port: ", port)}
\]

See Also

- Get Ethernet Configuration
- Save TCP/IP Port
6.5 (g) - Get DHCP Status

Declaration

```c
Short GetEthernet_UseDHCP()
```

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>DHCP not in use (IP settings are static and manually configured)</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>DHCP in use (IP settings are assigned automatically by the network)</td>
</tr>
</tbody>
</table>

Example

```c
Visual Basic
DHCPstatus = MyPTE1.GetEthernet_UseDHCP()

Visual C++
DHCPstatus = MyPTE1->GetEthernet_UseDHCP();

Visual C#
DHCPstatus = MyPTE1.GetEthernet_UseDHCP();

Matlab
[DHCPstatus] = MyPTE1.GetEthernet_UseDHCP
```

See Also

Get Ethernet Configuration
Use DHCP
6.5 (h) - Get Password Status

Declaration

Short GetEthernet_UsePWD()

Description

This function indicates whether the multi-channel attenuator system is currently configured to require a password for HTTP/Telnet communication.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Password not required</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Password required</td>
</tr>
</tbody>
</table>

Example

Visual Basic
PWDstatus = MyPTE1.GetEthernet_UsePWD()

Visual C++
PWDstatus = MyPTE1->GetEthernet_UsePWD();

Visual C#
PWDstatus = MyPTE1.GetEthernet_UsePWD();

Matlab
PWDstatus = MyPTE1.GetEthernet_UsePWD

[PWDstatus] = MyPTE1.GetEthernet_UsePWD

See Also

Get Password
Use Password
Set Password
6.5 (i) - Get Password

Declaration

```
Short GetEthernet_PWD(String Pwd)
```

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Pwd</td>
<td>Required. String variable which will be updated with the password.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

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

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

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

Matlab
[status, pwd] = MyPTE1.GetEthernet_PWD(pwd)
If status > 0 then
    {MsgBox ("Password: ", pwd)
}
```

See Also

Get Password Status
Use Password
Set Password
6.5 (j) - Save IP Address

Declaration

Short SaveEthernet_IPAddress(Int b1, Int b2, Int b3, Int b4)

Description

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

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>Required. First (highest order) octet of the IP address to set (for example “192” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>Required. Second octet of the IP address to set (for example “168” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP3</td>
<td>Required. Third octet of the IP address to set (for example “1” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>Required. Last (lowest order) octet of the IP address to set (for example “0” for the IP address “192.168.1.0”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get IP Address
6.5 (k) - Save Network Gateway

Declaration

Short SaveEthernet_NetworkGateway(Int b1, Int b2, Int b3, Int b4)

Description

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

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>Required. First (highest order) octet of the network gateway IP address (for example “192” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>Required. Second octet of the network gateway IP address (for example “168” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>Required. Third octet of the network gateway IP address (for example “1” for the IP address “192.168.1.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>Required. Last (lowest order) octet of the network gateway IP address (for example “0” for the IP address “192.168.1.0”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get Network Gateway
6.5 (l) - Save Subnet Mask

Declaration

Short SaveEthernet_SubnetMask (Int b1, Int b2, Int b3, Int b4)

Description

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

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>IP1</td>
<td>Required. First (highest order) octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP2</td>
<td>Required. Second octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP3</td>
<td>Required. Third octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).</td>
</tr>
<tr>
<td>Int</td>
<td>IP4</td>
<td>Required. Last (lowest order) octet of the subnet mask (for example “0” for the subnet mask “255.255.255.0”).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get Subnet Mask
6.5 (m) - Save TCP/IP Port

Declaration

`Short SaveEthernet_TCPIPPort(Int port)`

Description

This function sets the TCP/IP port used by the system 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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>port</td>
<td>Required. Numeric value of the TCP/IP port.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

`status = MyPTE1.SaveEthernet_TCPIPPort(70)`

Visual C++

`status = MyPTE1->SaveEthernet_TCPIPPort(70);`

Visual C#

`status = MyPTE1.SaveEthernet_TCPIPPort(70);`

Matlab

`[status] = MyPTE1.SaveEthernet_TCPIPPort(70)`

See Also

Get TCP/IP Port
6.5 (n) - Use DHCP

Declaration

`Short SaveEthernet_UseDHCP(Int UseDHCP)`

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>UseDHCP</td>
<td>Required. Integer value to set the DHCP mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - DHCP disabled (static IP settings used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - DHCP enabled (IP setting assigned by network)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic
```
status = MyPTE1.SaveEthernet_UseDHCP(1)
```

Visual C++
```
status = MyPTE1->SaveEthernet_UseDHCP(1);
```

Visual C#
```
status = MyPTE1.SaveEthernet_UseDHCP(1);
```

Matlab
```
[status] = MyPTE1.SaveEthernet_UseDHCP(1)
```

See Also

Get DHCP Status
6.5 (o) - Use Password

Declaration

Short SaveEthernet_UsePWD (Int UsePwd)

Description

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

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>UseDHCP</td>
<td>Required. Integer value to set the password mode: 0 – Password not required 1 – Password required</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

Visual Basic

status = MyPTE1.SaveEthernet_UsePWD(1)

Visual C++

status = MyPTE1->SaveEthernet_UsePWD(1);

Visual C#

status = MyPTE1.SaveEthernet_UsePWD(1);

Matlab

[status] = MyPTE1.SaveEthernet_UsePWD(1)

See Also

Get Password Status
Get Password
Set Password
6.5 (p) - Set Password

Declaration

Short SaveEthernet_PWD(String Pwd)

Description

This function sets the password used by the system for HTTP/Telnet communication. The password will not affect switch operation unless Use Password is also enabled.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Pwd</td>
<td>Required. The password to set (20 characters maximum).</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Example

**Visual Basic**

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

**Visual C++**

```cpp
status = MyPTE1->SaveEthernet_PWD("123");
```

**Visual C#**

```csharp
status = MyPTE1.SaveEthernet_PWD("123");
```

**Matlab**

```matlab
[status] = MyPTE1.SaveEthernet_PWD("123")
```

See Also

- Get Password Status
- Get Password
- Use Password
7 - USB Control Using Interrupt Codes

In environments that do not support ActiveX objects or the .NET frameworks, Mini-Circuits' API DLL files cannot be used; this includes programming environments running on PCs with a UNIX based operating system. In these cases, the method of control is to use USB interrupt codes.

7.1 - USB Interrupt Code Concept

To open a USB connection to a multi-channel attenuator system, the Vendor ID and Product ID are required:
- Mini-Circuits Vendor ID: 0x20CE
- Product ID: 0x22

Communication with the system 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 system.

The switch module connected by USB automatically becomes the Master, with any other modules cascaded via the SPI connections defined as Slaves.
7.2 - Summary of Interrupt Commands

7.2 (a) - Send SCPI Command

Description

This function sends a SCPI command to the attenuator system and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the cascadable switch system.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 or *</td>
<td>Interrupt code for Send SCPI Command</td>
</tr>
<tr>
<td>1 - 63</td>
<td>SCPI Transmit String</td>
<td>The SCPI command to send represented as a series of ASCII character codes, one character code per byte</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 or *</td>
<td>Interrupt code for Send SCPI Command</td>
</tr>
<tr>
<td>1 to (n-1)</td>
<td>SCPI Return String</td>
<td>The SCPI return string, one character per byte, represented as ASCII character codes</td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>Zero value byte to indicate the end of the SCPI return string</td>
</tr>
<tr>
<td>(n+1) to 63</td>
<td>Not significant</td>
<td>“Don’t care” bytes, can be any value</td>
</tr>
</tbody>
</table>
Example (Get Model Name of Master)

The SCPI command to request the model name of the Master is :00:MN? (see Get Model Name)

The ASCII character codes representing the 7 characters in this command should be sent in bytes 1 to 4 of the transmit array as follows:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Interrupt code for Send SCPI Command</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td>ASCII character code for :</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>ASCII character code for 0</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>ASCII character code for 0</td>
</tr>
<tr>
<td>4</td>
<td>49</td>
<td>ASCII character code for :</td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ASCII character code for M</td>
</tr>
<tr>
<td>3</td>
<td>78</td>
<td>ASCII character code for N</td>
</tr>
<tr>
<td>7</td>
<td>63</td>
<td>ASCII character code for ?</td>
</tr>
</tbody>
</table>

The returned array for ZTDAT-16-6G95A would be as follows:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Interrupt code for Send SCPI Command</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
<td>ASCII character code for Z</td>
</tr>
<tr>
<td>2</td>
<td>84</td>
<td>ASCII character code for T</td>
</tr>
<tr>
<td>3</td>
<td>68</td>
<td>ASCII character code for D</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>ASCII character code for A</td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td>ASCII character code for T</td>
</tr>
<tr>
<td>6</td>
<td>45</td>
<td>ASCII character code for -</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>ASCII character code for 1</td>
</tr>
<tr>
<td>8</td>
<td>54</td>
<td>ASCII character code for 6</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>ASCII character code for -</td>
</tr>
<tr>
<td>10</td>
<td>54</td>
<td>ASCII character code for 6</td>
</tr>
<tr>
<td>11</td>
<td>71</td>
<td>ASCII character code for G</td>
</tr>
<tr>
<td>12</td>
<td>57</td>
<td>ASCII character code for 9</td>
</tr>
<tr>
<td>13</td>
<td>53</td>
<td>ASCII character code for 5</td>
</tr>
<tr>
<td>14</td>
<td>65</td>
<td>ASCII character code for A</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>Zero value byte to indicate end of string</td>
</tr>
</tbody>
</table>

See Also

SCPI Commands for Control of Attenuator Racks
8 - Programming Examples

These examples are intended to demonstrate the basics of programming with Mini-Circuits' multi-channel attenuator rack and mesh network test systems. If support is required for a specific programming example which isn't covered below then please contact Mini-Circuits through testsolutions@minicircuits.com.

8.1 - Visual Basic (VB) Programming

8.1 (a) - USB Connection Using the .NET DLL

The .NET DLL provides the simplest API for USB control in a Windows environment. The DLL can be used on any combination of 32-bit or 64-bit operating systems and programming environments, as long as the environment supports the .NET framework. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```vbnet
Dim ztm As New MCL_MultiChannelAtt_64.USB_MultiChannelAtt ' New instance of the USB class

Dim a_value As String = "12.75" ' Attenuation value (dB) to set
Dim a_block As Integer
Dim a_channel As Integer
Dim a_return As String = ""

' Loop through six 4-channel attenuator blocks (addressed 01 to 06)
For a_block = 1 To 6
    ' Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    For a_channel = 1 To 4
        ' Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        ztm.Send_SCPI("0" & a_block & ":CHAN:" & a_channel & ":SETATT:" & a_value, a_return)

        ' Query and print the attenuator value
        ztm.Send_SCPI("0" & a_block & ":CHAN:" & a_channel & ":ATT?", a_return)
        Console.WriteLine("Block ", a_block & " channel ", a_channel & " set to ", a_return)
    Next
Next

ztm.Disconnect() ' Disconnect from the DLL on completion
```
8.1 (b) - Ethernet HTTP Connection

Microsoft. NET's WebRequest class can be used to send HTTP commands to the multi-channel attenuator system for Ethernet control from Visual Basic. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```vbnet
Imports System.IO
Imports System.Net

Private Function Send_HTTP_Get(Command_To_Send As String) As String
    ' Declare a function to send an HTTP command and return the response
    Dim webStream As Stream
    Dim ZT_IP_Address As String = "192.100.1.1" ' The IP address of the attenuator system
    Dim HTTP_Response As String = "" ' A string to record the return value
    Dim urlToSend As HttpWebRequest
    Dim urlResponse As HttpWebResponse
    Dim urlToSend As String = "http://192.100.1.1:/" & Command_To_Send
    urlToSend.Method = "GET"
    urlResponse = urlToSend.GetResponse()
    webStream = urlResponse.GetResponseStream()
    Dim webStreamReader As New StreamReader(webStream)
    While webStreamReader.Peek >= 0
        HTTP_Response = webStreamReader.ReadToEnd()
    End While
    Return Trim(HTTP_Response)
End Function

' Use the above function to send the HTTP commands required to control the attenuator system
' ****************************************
Dim a_value As String = "12.75" ' Attenuation value (dB) to set
Dim a_block As Integer
Dim a_channel As Integer
Dim a_return As String = ""

' Loop through six 4-channel attenuator blocks (addressed 01 to 06)
For a_block = 1 To 6
    ' Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    For a_channel = 1 To 4
        ' Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        a_return = Send_HTTP_Get("0" & a_block & ":CHAN:" & a_channel & ":SETATT:" & a_value)
        ' Query and print the attenuator value
        a_return = Send_HTTP_Get("0" & a_block & ":CHAN:" & a_channel & ":ATT?"
        Console.WriteLine("Block " & a_block & " channel " & a_channel & " set to " & a_return)
    Next
Next
```
8.2 - Python Programming

8.2 (a) - Ethernet Connection Using Python's urllib2 Library for HTTP

Python's urllib2 library can be used to send HTTP commands to the multi-channel attenuator systems when programming with Python. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```python
# Define a function to send an HTTP command and get the result
def Get_HTTP_Result(CmdToSend):
    # Specify the IP address
    CmdToSend = "http://192.100.1.1/:") + CmdToSend

    # Send the HTTP command and try to read the result
    try:
        HTTP_Result = urllib2.urlopen(CmdToSend)
        PTE_Return = HTTP_Result.read()
    except:
        print "Error, no response from device; check IP address and connections."
        PTE_Return = "No Response!"
        sys.exit()  # Exit the script

    # Return the response
    return PTE_Return

# Send commands / queries to the attenuator
value = 12.75  # The attenuation to set

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for a_block in range(1, 7):
    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for a_channel in range(1, 5):
        # Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        att = Get_HTTP_Result("0" + a_block + ":CHAN:" + a_channel + ":SETATT:" + value)

        # Query and print the attenuator value
        att = Get_HTTP_Result("0" + a_block + ":CHAN:" + a_channel + ":ATT?")
        print "Block", str(a_block), "channel", str(a_channel), "set to", str(att)
```

---

8.2 (b) - Using Python's urllib2 Library for FTP

Python's urllib2 library can also be used to send FTP commands to multi-channel attenuator systems when programming with Python. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.
8.2 (b) - USB Connection Using the ActiveX DLL (32-bit Python Distributions)

The majority of 32-bit Pythons distributions for Windows operating systems support ActiveX, meaning Mini-Circuits’ ActiveX DLL can be used to control the multi-channel attenuator system in these environments. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```python
import win32com.client  # Reference PyWin32
import pythoncom

ztm = win32com.client.Dispatch("MCL_MultiChannelAtt.USB_Control")  # Reference the DLL
Conn_Status = ztm.Connect()  # Connect to the DLL
value = 12.75  # The attenuation to set

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for a_block in range(1, 7):
    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for a_channel in range(1, 5):
        # Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        att = ztm.Send_SCPI("0"+str(a_block)+";CHAN:"+str(a_channel)+";SETATT:"+str(value), att_return);

        # Query and print the attenuator value
        att = ztm.Send_SCPI("0"+str(a_block)+";CHAN:"+str(a_channel)+";ATT?", att_return);
        print "Block", str(a_block), "channel", str(a_channel), "set to", str(att_return)

ztm.Disconnect  # Disconnect from the DLL on completion
```
8.2 (c) - Work-Around for 64-bit Python Distributions with a USB Connection

The majority of 64-bit Perl distributions do not provide support for either ActiveX or .Net so in these cases Mini-Circuits’ DLLs cannot be used directly. The work-around when a USB connection is required is to create a separate executable program in another programming environment which can sit in the middle. The function of the executable is to use the .Net DLL to connect to the multi-channel attenuator system, send a single user specified command, return the response to the user, and disconnect from the DLL. This executable can then be easily called from Perl script to send the required commands to the system, without Perl having to directly interface with the DLL.

Mini-Circuits can supply on request an executable to interface with the DLL. See Creating an Executable Using the .Net DLL in C# for USB Control for the example source code for such an executable (developed using C#). The below script demonstrates use of this executable in Perl script to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```python
import time
import os
import subprocess
from os import system
from subprocess import Popen, PIPE

sn = "-s 11604280010"  # Serial number of the multi-channel attenuator system
value = "75.0";  # Attenuation value to set

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for a_block in range(1, 7):
    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for a_channel in range(1, 5):
        # Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        pipe = subprocess.Popen("ZTM.exe"+sn+"0"+a_block+":CHAN:"+a_channel+":SETATT:"+value,
                        stdout=subprocess.PIPE)
        pipe.wait
        att_return = pipe.stdout.read()
        # Query and print the attenuation value
        pipe = subprocess.Popen("ZTM.exe"+sn+"0"+a_block+":CHAN:"+a_channel+":ATT?",
                        stdout=subprocess.PIPE)
        pipe.wait
        att_return = pipe.stdout.read()
        print "Block", str(a_block), "channel", str(a_channel), "set to", str(att_return)
```

8.3 - Perl Programming

8.3 (a) - Ethernet Connection Using Perl's LWP Simple Interface for HTTP

Perl's LWP Simple interface can be used to send HTTP commands to the multi-channel attenuator systems when programming with Perl. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```perl
#!/usr/bin/perl
use strict;
use warnings;
use LWP::Simple;  # Use the LWP::Simple interface for HTTP

my $ip_address = "192.100.1.1";  # IP address of the multi-channel attenuator system
my $value = "75.0";  # Attenuation value to set
my $return_value = "0";  # Variable to store the responses from the system

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for ( $a_block = 1; $a_block < 7; $a_block = $a_block + 1 ){
    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for ( $a_channel = 1; $a_channel < 5; $a_channel = $a_channel + 1 ){
        # Send set attenuation command (eg: http://192.100.1.1/:01:CHAN:1:SETATT:12.75)
        $return_value = get("http://$ip_address/:0$a_block:CHAN:$a_channel:SETATT:$value");

        # Query and print the attenuator value
        $return_value = get("http://$ip_address/:0$a_block:CHAN:$a_channel:ATT?");
        print "Block $a_block channel $a_channel set to $return_value\n";
    }
}
```
8.3 (b) - USB Connection Using the ActiveX DLL (32-bit Perl Distributions)

The majority of 32-bit Perl distributions for Windows operating systems support ActiveX, meaning Mini-Circuits' ActiveX DLL can be used to control the multi-channel attenuator system in these environments. The below code example demonstrates a process to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```perl
use feature ':5.10';
use Win32::OLE;
use Win32::OLE::Const 'Microsoft ActiveX Data Objects';

my $ztm = Win32::OLE->new('MCL_MultiChannelAtt.USB_Control');
my $value = "75.0";       # Attenuation value to set
my $return_value = "0";  # Variable to store the responses from the system
$ztm-&gt;Connect();  # Connect to the DLL

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for ( $a_block = 1; $a_block &lt; 7; $a_block = $a_block + 1 ){

    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for( $a_channel = 1; $a_channel &lt; 5; $a_channel = $a_channel + 1 ){

        # Send set attenuation command (eg: :01:CHAN:1:SETATT:12.75)
        $ztm-&gt;Send_SCPI(":\$a_block:CHAN:\$a_channel:SETATT:12.75", $return_value);

        # Query and print the attenuator value
        $ztm-&gt;Send_SCPI(":\$a_block:CHAN:\$a_channel:ATT?", $return_value);
        print "Block $a_block channel $a_channel set to $return_value\n";
    }
}

$ztm-&gt;Disconnect;  # Disconnect from the DLL on completion
```
8.3 (c) - Work-Around for 64-bit Perl Distributions with a USB Connection

The majority of 64-bit Perl distributions do not provide support for either ActiveX or .Net so in these cases Mini-Circuits’ DLLs cannot be used directly. The work-around when a USB connection is required is to create a separate executable program in another programming environment which can sit in the middle. The function of the executable is to use the .Net DLL to connect to the multi-channel attenuator system, send a single user specified command, return the response to the user, and disconnect from the DLL. This executable can then be easily called from Perl script to send the required commands to the system, without Perl having to directly interface with the DLL.

Mini-Circuits can supply on request an executable to interface with the DLL. See Creating an Executable Using the .Net DLL in C# for USB Control for the example source code for such an executable (developed using C#). The below script demonstrates use of this executable in Perl script to set the first 24 attenuators in a multi-channel attenuator rack structure. The attenuators are structured in 6 blocks of 4-channels.

```perl
#!/usr/bin/perl
use strict;
use warnings;

my $serial_number = 11604280010;  # The ZTM Series serial number
my $value = "75.0";               # Attenuation value to set
my $return_value = "0";          # Variable to store the responses from the system

my $exe = "ZTM.exe";  # The .exe providing an interface to the multi-channel attenuator DLL
my @cmd;

# Loop through six 4-channel attenuator blocks (addressed 01 to 06)
for ( $a_block = 1; $a_block < 7; $a_block = $a_block + 1 ){
    # Loop through the 4 channels (1 to 4) in each 4-channel attenuator block
    for ( $a_channel = 1; $a_channel < 5; $a_channel = $a_channel + 1 ){
        # Send set attenuation command (eg: 01:CHAN:1:SETATT:12.75)
        @cmd = ($exe, "-s $serial_number:0$a_block:CHAN:$a_channel:SETATT:$value");
        my $return_value = qx{@cmd};

        # Query and print the attenuator value
        @cmd = ($exe, "-s $serial_number:0$a_block:CHAN:$a_channel:ATT?");
        my $return_value = qx{@cmd};
        print "Block $a_block channel $a_channel set to $return_value\n"
    }
}
```
8.4 - C# Programming

8.4 (a) - Creating an Executable Using the .Net DLL in C# for USB Control

The below example is a simple executable program that connects to the .Net DLL, sends a user specified SCPI command to the test system, returns the response, then disconnects from the DLL and terminates. It requires the .Net DLL to be installed on the host operating system and the ZTM or RCM Series test system to be connected to the PC via USB. It can be used as a workaround where a USB connection is required but the programming environment does not provide native support for ActiveX or .NET (for example with some 64-bit distributions of Python or Perl).

```csharp
namespace ZTM
{
    class Program
    {
        static int Main(string[] args)
        {
            int x = 0;
            string SN = null;
            string SCPI = null;
            string RESULT = null;
            int Add = 0;
            MCL_MultiChannelAtt.USB_Control ZT;  // Reference the DLL
            if (args.Length == 0) return 0;
            ZT = new MCL_MultiChannelAtt.USB_Control();  // Declare a class
            SCPI = args[2];
            if (args[0].ToString().Contains("-help")) // Print a help file
            {
                Console.WriteLine("Help ZTM.exe");
                Console.WriteLine("-s SN command :Send SCPI command to S/N");
                Console.WriteLine("-a add SCPI :Send SCPI command to Address");
                Console.WriteLine("-");
            }
            if (args[0].ToString().Contains("-s")) // User wants to connect by S/N
            {
                SN = args[1];
                x = ZT.Connect(ref SN);  // Call DLL connect function
                x = ZT.Send_SCPI(ref SCPI, ref RESULT);  // Send SCPI command
                Console.WriteLine(RESULT);  // Return the result
            }
            if (args[0].ToString().Contains("-a")) // User wants to connect by address
            {
                Add = Int16.Parse(args[1]);
                x = ZT.ConnectByAddress(ref Add);
                x = ZT.Send_SCPI(ref SCPI, ref RESULT);
                Console.WriteLine(RESULT);
            }
            ZT.Disconnect();  // Call DLL disconnect function to finish
            return x;
        }
    }
}
```

This executable can be called from a command line prompt or within a script. The following command line calls demonstrate use of the executable (compiled as ZTM.exe), connecting by serial number or address, to set and read attenuation:

- `ZTM.exe -s 11601250027 :01:CHAN:1:SETATT:12.75` (serial number 11601250027)
- `ZTM.exe -a 255 :01:CHAN:1:ATT?` (USB address 255)