Test Solutions - Programming Manual

Multi-Channel Attenuators & Mesh Networks

ZTDAT Series Multi-Channel Programmable Attenuator Racks
ZT Series Custom Multi-Channel Attenuator Racks
ZTMN Series Mesh Network Racks

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

Important Notice

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

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.

The full software and documentation package including a GUI program, DLL files, user guide and programming examples is available for download from the Mini-Circuits website at:
https://www.minicircuits.com/softwaredownload/solidstate.html

For details and specifications on the individual models, please see:
https://www.minicircuits.com/WebStore/RF-Solid-State-Compact-Switch.html

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

1.1 - Control Methods

Communication with the system can use any of the following approaches:

1. Using SSH, HTTP or Telnet communication via an Ethernet TCP / IP connection (see Ethernet Control API), which is largely independent of the operating system
   Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.
2. Using the provided API DLL files (ActiveX or .Net objects) on Microsoft Windows operating systems (see USB Control API for Microsoft Windows)
3. Using USB interrupt codes for direct programming on Linux operating systems (see USB Control via Direct Programming (Linux))

Setting states and querying the system is achieved using a command set based on SCPI (see SCPI Commands for Control of Attenuator Racks), sent via one of the above methods.

2 - Mini-Circuits' Daisy-Chain Control Concept

The daisy-chain 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 or 8 attenuator channels. The configuration is identified on the block diagram with 4-channel attenuator blocks named "RS4DAT" and 8-channel blocks named "RS8DAT"). Each RS4DAT or RS8DAT block is accessible through its 2 digit address, from 01 to nn. The 4 or 8 channels within each block can be individually controlled using their channel numbers, from 1 to 8, so the command `:03:CHAN:2:SETATT:12.75` for example sets attenuator channel 2 in RS4DAT block 3 to 12.75 dB (the path between ports A11 and B11 in figure 1).

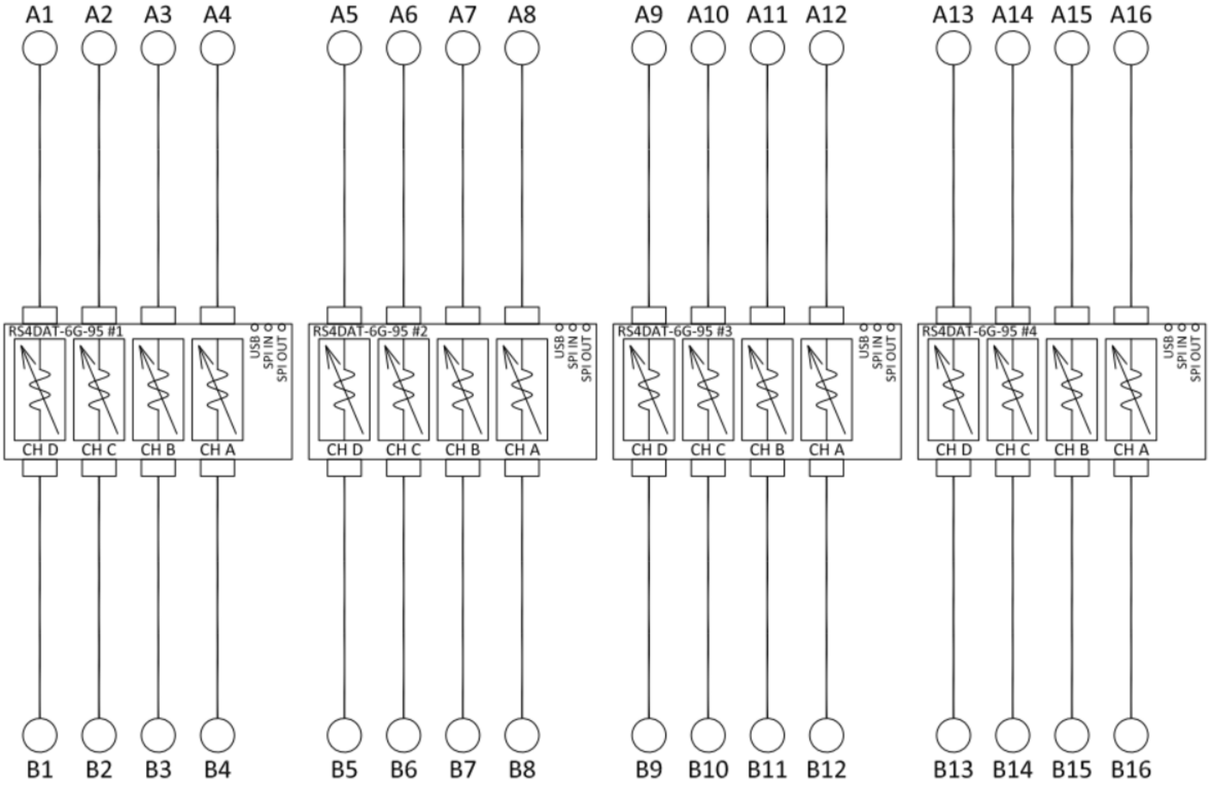


Fig 1 - Internal Block Diagram of ZTDAT-16-6G95A (16 channel) Showing Attenuator Channel Structure

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 daisy-chained multi-channel attenuator racks.

2.2 - Daisy-Chaining Multiple Racks

When multiple attenuator racks are cascaded via their serial interfaces the structure is as if all of the RS4DAT / RS8DAT 4-channel or 8-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.

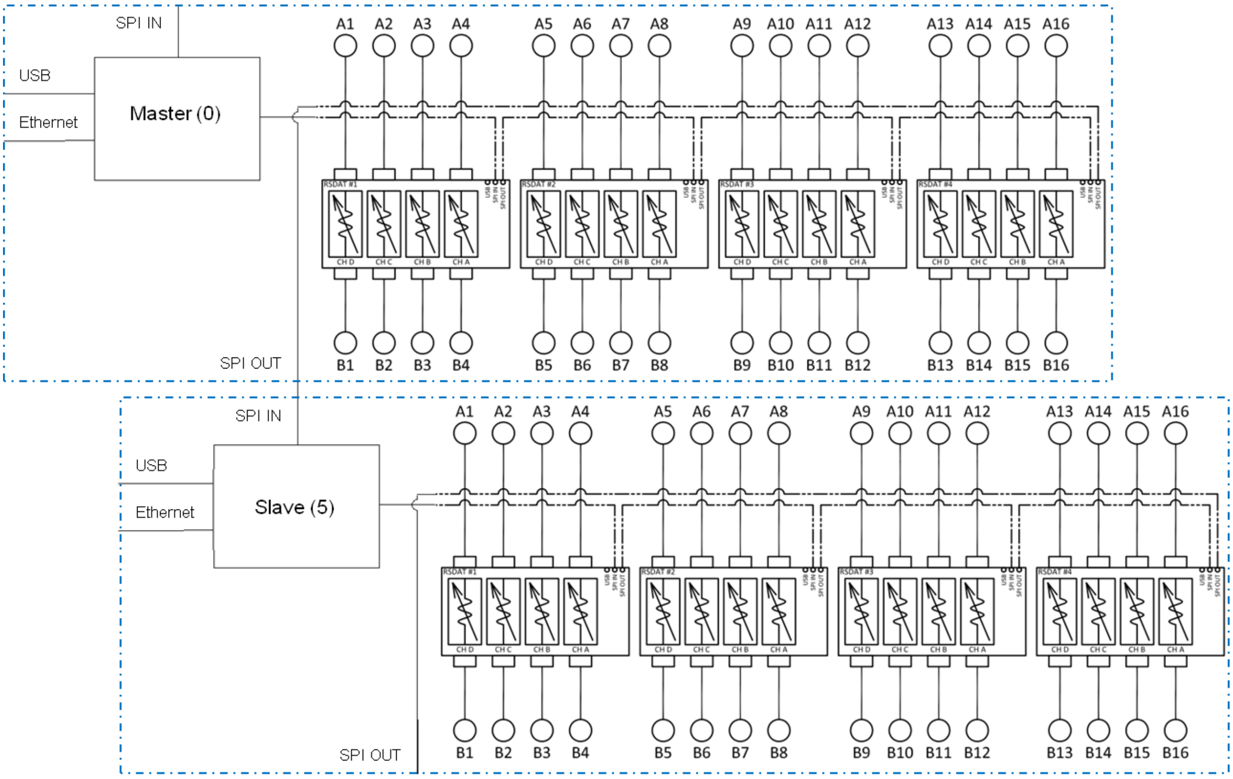


Fig 2 - Block Diagrams Showing Control Connections for 2 Cascaded ZTDAT-16-6G95A Attenuator Racks

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 serial 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	Attenuator Rack	Description
00	Master	Controller
01		4-channel attenuator (RS4DAT) block 1
02		4-channel attenuator (RS4DAT) block 2
03		4-channel attenuator (RS4DAT) block 3
04		4-channel attenuator (RS4DAT) block 4
05	Slave 1	Controller
06		4-channel attenuator (RS4DAT) block 1
07		4-channel attenuator (RS4DAT) block 2
08		4-channel attenuator (RS4DAT) block 3
09		4-channel attenuator (RS4DAT) block 4
10	Slave 2	Controller
11		4-channel attenuator (RS4DAT) block 1
12		4-channel attenuator (RS4DAT) block 2
13		4-channel attenuator (RS4DAT) block 3
14		4-channel attenuator (RS4DAT) block 4

Fig 3 - 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.

#	Set Channel A	Set Channel B	Set Channel C	Set Channel D
01	:01:CHAN:1:SETATT:x	:01:CHAN:2:SETATT:x	:01:CHAN:3:SETATT:x	:01:CHAN:4:SETATT:x
02	:02:CHAN:1:SETATT:x	:02:CHAN:2:SETATT:x	:02:CHAN:3:SETATT:x	:02:CHAN:4:SETATT:x
03	:03:CHAN:1:SETATT:x	:03:CHAN:2:SETATT:x	:03:CHAN:3:SETATT:x	:03:CHAN:4:SETATT:x
04	:04:CHAN:1:SETATT:x	:04:CHAN:2:SETATT:x	:04:CHAN:3:SETATT:x	:04:CHAN:4:SETATT:x
06	:06:CHAN:1:SETATT:x	:06:CHAN:2:SETATT:x	:06:CHAN:3:SETATT:x	:06:CHAN:4:SETATT:x
07	:07:CHAN:1:SETATT:x	:07:CHAN:2:SETATT:x	:07:CHAN:3:SETATT:x	:07:CHAN:4:SETATT:x
08	:08:CHAN:1:SETATT:x	:08:CHAN:2:SETATT:x	:08:CHAN:3:SETATT:x	:08:CHAN:4:SETATT:x
09	:09:CHAN:1:SETATT:x	:09:CHAN:2:SETATT:x	:09:CHAN:3:SETATT:x	:09:CHAN:4:SETATT:x
11	:11:CHAN:1:SETATT:x	:11:CHAN:2:SETATT:x	:11:CHAN:3:SETATT:x	:11:CHAN:4:SETATT:x
12	:12:CHAN:1:SETATT:x	:12:CHAN:2:SETATT:x	:12:CHAN:3:SETATT:x	:12:CHAN:4:SETATT:x
13	:13:CHAN:1:SETATT:x	:13:CHAN:2:SETATT:x	:13:CHAN:3:SETATT:x	:13:CHAN:4:SETATT:x
14	:14:CHAN:1:SETATT:x	:14:CHAN:2:SETATT:x	:14:CHAN:3:SETATT:x	:14:CHAN:4:SETATT:x

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.

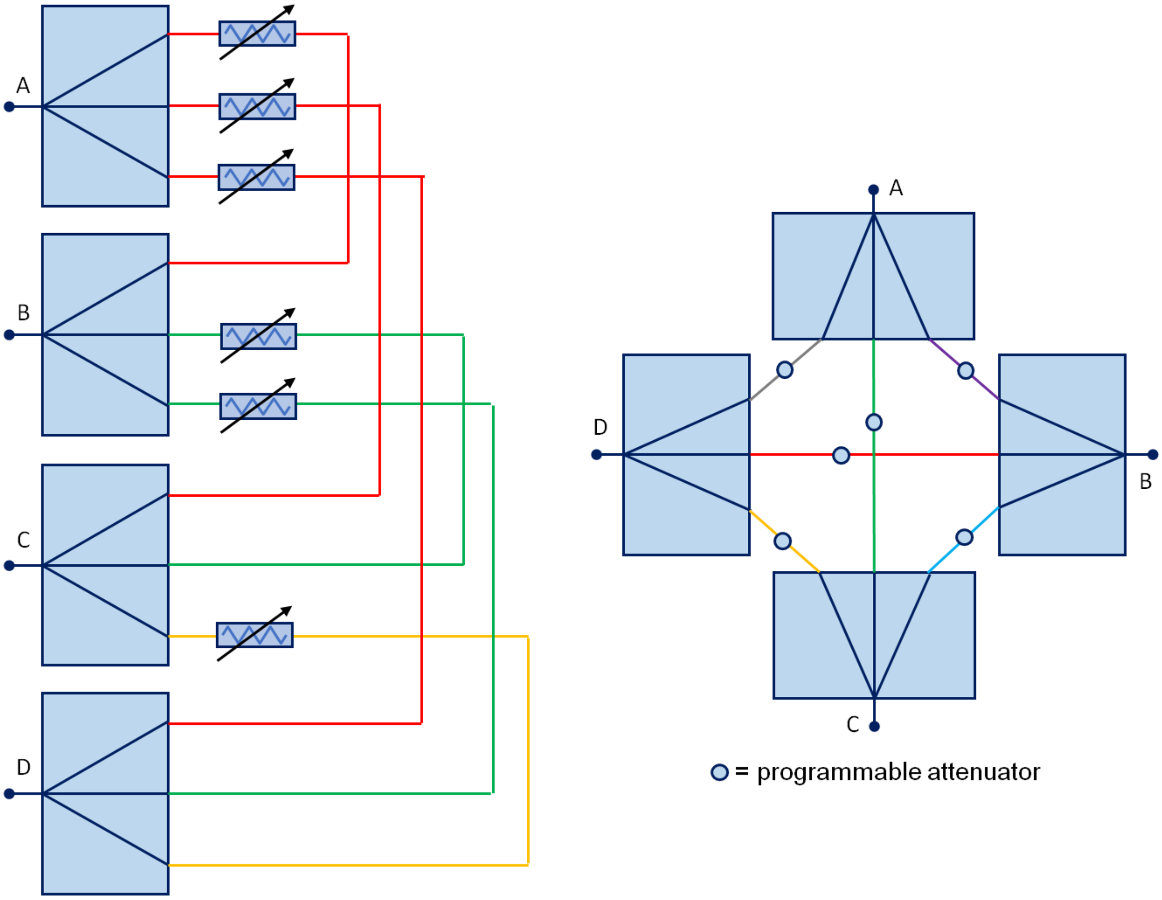
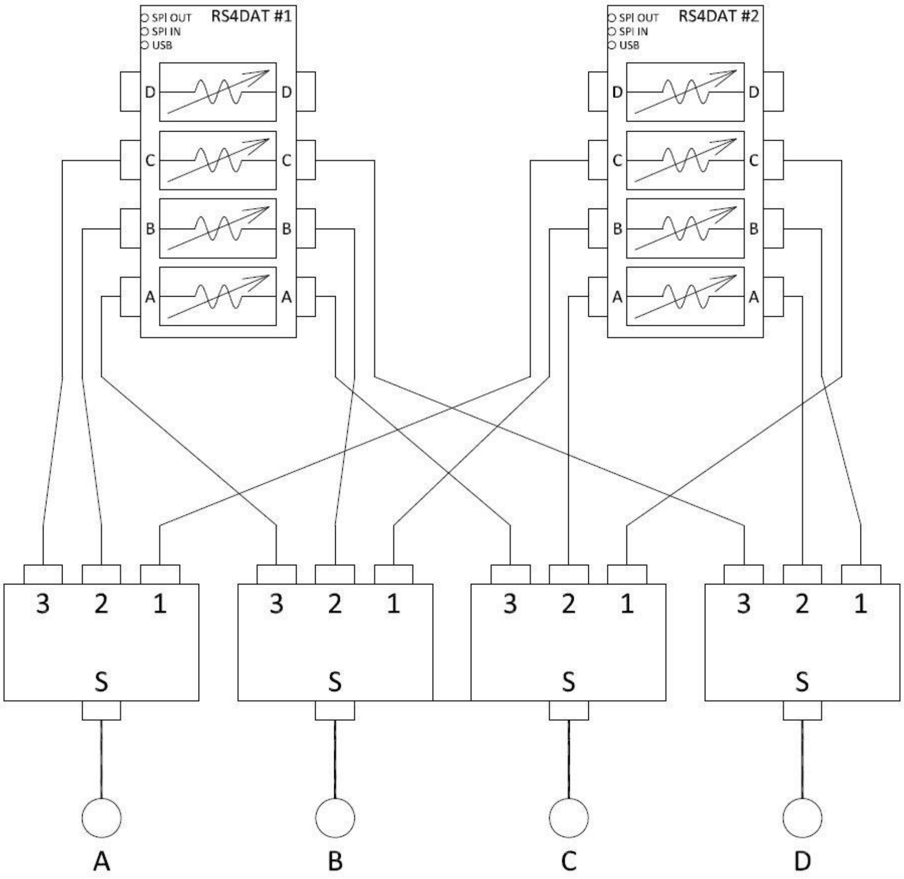


Fig 5 - Two Alternative Representations of a 4-Port Mesh Network, Showing the Programmable Attenuators on Each Path

3.1 - Mesh Network Rack Structure

ZTMN Series mesh networks are formed by integrating passive splitter / combiners with Mini-Circuits' 4-channel "RS4DAT" or 8-channel “RS8DAT” 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 of 4-channel RS4DAT programmable attenuator blocks and 4 separate 3-way splitter / combiners. Note that only 3 channels from each programmable attenuator block are required for this configuration.



4-Channel Attenuator	Attenuator Channels for Each Mesh Network Path			
	Channel 1 (A)	Channel 2 (B)	Channel 3 (C)	Channel 4 (D)
RS4DAT #1	Port B <-> Port C	Port A <-> Port B	Port A <-> Port D	No connection
RS4DAT #2	Port C <-> Port D	Port B <-> Port D	Port A <-> Port C	No connection

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 RS4DAT 4-channel attenuator blocks 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 of 2 RS4DAT 4-channel attenuator blocks (for 8 channels total) although only 6 channels are in use.

Address	Description
00	Controller
01	4-channel attenuator (RS4DAT) block 1
02	4-channel attenuator (RS4DAT) block 2

#	Set Channel 1 (A)	Set Channel 2 (B)	Set Channel 3 (C)	Set Channel 4 (D)
01	`:01:CHAN:1:SETATT:x`	`:01:CHAN:2:SETATT:x`	`:01:CHAN:3:SETATT:x`	`N/A`
02	`:02:CHAN:1:SETATT:x`	`:02:CHAN:2:SETATT:x`	`:02:CHAN:3:SETATT:x`	`N/A`

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

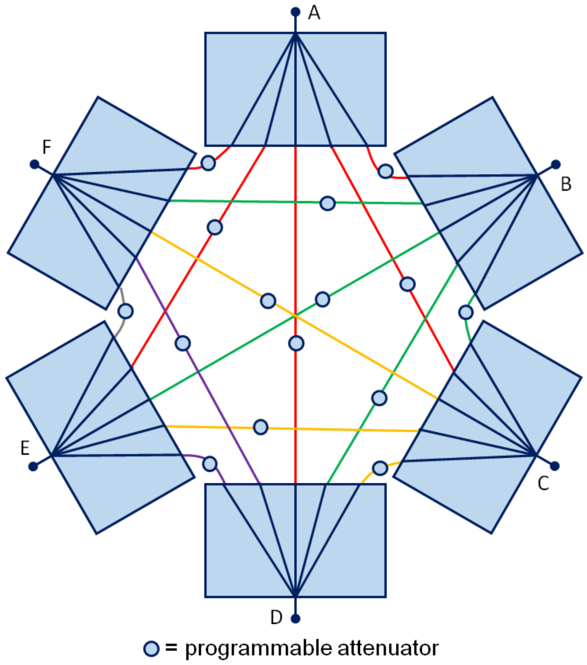


Fig 8- Representation of a 6-Port Mesh Network, Comprising Six 5-Way Splitter / Combiners and Fifteen Attenuator Channels

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 RS4DAT 4-channel attenuator blocks (for 16 channels total) although only 15 channels are in use.

Address	Description
00	Controller
01	4-channel attenuator (RS4DAT) block 1
02	4-channel attenuator (RS4DAT) block 2
03	4-channel attenuator (RS4DAT) block 3
04	4-channel attenuator (RS4DAT) block 4

#	Set Channel 1 (A)	Set Channel 2 (B)	Set Channel 3 (C)	Set Channel 4 (D)
01	`:01:CHAN:1:SETATT:x`	`:01:CHAN:2:SETATT:x`	`:01:CHAN:3:SETATT:x`	`:01:CHAN:4:SETATT:x`
02	`:02:CHAN:1:SETATT:x`	`:02:CHAN:2:SETATT:x`	`:02:CHAN:3:SETATT:x`	`:02:CHAN:4:SETATT:x`
03	`:03:CHAN:1:SETATT:x`	`:03:CHAN:2:SETATT:x`	`:03:CHAN:3:SETATT:x`	`:03:CHAN:4:SETATT:x`
04	`:04:CHAN:1:SETATT:x`	`:04:CHAN:2:SETATT:x`	`:04:CHAN:3:SETATT:x`	N/A

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 recommended method for setting states and querying the system is a series of commands based on SCPI (Standard Commands for Programmable Instruments). These commands can be sent using any of the APIs detailed in this manual.

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

4.1 - 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 Master controller.

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

	Description	Command/Query
a	Assign Address	**:AssignAddresses**
b	Count Number of Slaves	**:NumberOfSlaves?**

4.1 (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]
```

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

Examples

HTTP Implementation: `http://10.10.10.10/:AssignAddresses`

See Also

Count Number of Slaves

4.1 (b) - Count Number of Slaves

Description

Used for daisy-chained ZTDAT multi-channel attenuator racks to identify the number of RS4DAT 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]
```

Variable	Description
[count]	The total number of Slaves connected into the Master

Examples

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

See Also

Assign Addresses

4.2 - 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.

	Description	Command/Query
a	`Get Model Name`	`:[address]:MN?`
b	`Get Serial Number`	`:[address]:SN?`
c	`Get Firmware`	`:[address]:FIRMWARE?`
d	`Set System Name`	`:[address]:SYSNAME:[name]`
e	`Get System Name`	`:[address]:SYSNAME?`

4.2 (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]
```

Variable	Description
`[model]`	Model name of the multi-channel attenuator rack

Examples

String to Send	String Returned
`:MN?`	`ZTDAT-16-6G95A`
`:00:MN?`	`:00:ZTDAT-16-6G95A`
`:01:MN?`	`:01:RS4DAT-6G-95`
`:05:MN?`	`:05:ZTDAT-16-6G95A`

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

See Also

Get Serial Number
Get Firmware
Set System Name
Get System Name

4.2 (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]
```

Variable	Description
[serial]	Serial number of the multi-channel attenuator rack

Examples

String to Send	String Returned
:SN?	11612010001
:00:SN?	:00:11612010001
:05:SN?	:01:11612010002

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

See Also

Get Model Name
Get Firmware
Set System Name
Get System Name

4.2 (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]
```

Variable	Description
[firmware]	Firmware version name

Examples

String to Send	String Returned
:FIRMWARE?	A1
:00:FIRMWARE?	:00:A1
:01:FIRMWARE?	:01:A1

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

See Also

Get Model Name
Get Serial Number
Set System Name
Get System Name

4.2 (d) - Set System Name

Description

Sets a custom name / label (up to 50 characters) of the system for easy identification.

Description

Firmware B7 or later

Command Syntax

```
:[address]:SYSNAME:[name]
```

Return String

```
:[address]:[status]
```

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

Examples

String to Send	String Returned
:SYSNAME:Wi-Fi Test Rack	1

HTTP Implementation: `http://10.10.10.10/:SYSNAME:Wi-Fi Test Rack`

See Also

Get Model Name
Get Serial Number
Get Firmware
Get System Name

4.2 (e) - Get System Name

Description

Returns a custom name / label of the system for easy identification.

Description

Firmware B7 or later

Command Syntax

```
:[address]:SYSNAME?
```

Return String

```
:[address]:[name]
```

Variable	Description
[name]	The custom name / label defined for the system

Examples

String to Send	String Returned
:SYSNAME?	Attenuator System
:01:SYSNAME?	:01:RS4DAT-6G-95

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

See Also

Get Model Name
Get Serial Number
Get Firmware
Set System Name

4.3 - 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 RS4DAT 4-channel or RS8DAT 8-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)

	Description	Command/Query
a	Set Attenuation	:[address]:CHAN:[channels]:SETATT:[att]
b	Read Attenuation	:[address]:CHAN:[channel]:ATT?
c	Set Start-Up Attenuation Mode	:[address]:STARTUPATT:INDICATOR:[mode]
d	Get Start-Up Attenuation Mode	:[address]:STARTUPATT:INDICATOR?
e	Store Last Attenuation Value	:SL:LASTATT:STORE:INITIATE
f	Set Channel Start-Up Attenuation	:[address]:CHAN:[channels]:STARTUPATT:VALUE:[att]
g	Get Channel Start-Up Attenuation	:[address]:CHAN:[channel]:STARTUPATT:VALUE?
h	Set Attenuator Label	:[address]:CHAN:[channel]:LABEL:[label_text]
i	Read Attenuator Label	:[address]:CHAN:[channel]:LABEL?

4.3 (a) - Set Attenuation

Description

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

Command Syntax

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

Variable	Description
`[channels]`	The attenuator channel number within the RS4DAT / RS8DAT attenuator block from 1 (channel A) to 8 (channel H). Multiple channels can be sent by listing each channel number separated by a colon
`[Att]`	The attenuation to set

Return String

```
:[address]:[status]
```

Variable	Value	Description
`[status]`	0	Command failed
	1	Command completed successfully
	2	Requested attenuation value exceeds maximum - Maximum attenuation set

Examples

Channels	String to Send	String Returned
RS4DAT 1, channel 1	`:01:CHAN:1:SETATT:10.25`	`:01:1`
RS4DAT 1, channels 1-4	`:01:CHAN:1:2:3:4:SETATT:10.25`	`:01:1`
All channels	`:SL:CHAN:1:2:3:4:SETATT:10.25`	`:SL:CHAN:1:2:3:4:SETATT:10.25`
RS4DAT 2, channel 1	`:02:CHAN:1:SETATT:0`	`:02:1`

HTTP Implementation: `http://10.10.10.10/:01:CHAN:1:SETATT:10.25`

See Also

Read Attenuation

4.3 (b) - Read Attenuation

Description

Returns the attenuation of a single channel

Command Syntax

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

Variable	Description
`[channel]`	The attenuator channel number within the RS4DAT / RS8DAT attenuator block from 1 (channel A) to 8 (channel H)

Return String

```
:[address]:[att]
```

Variable	Description
`[att]`	The attenuation (dB) of the requested channel

Examples

String to Send	String Returned
`:01:CHAN:1:ATT?`	`:01:10.25`
`:01:CHAN:2:ATT?`	`:01:20.50`
`:01:CHAN:3:ATT?`	`:01:30.75`
`:01:CHAN:4:ATT?`	`:01:40.0`
`:02:CHAN:1:ATT?`	`:02:0.0`

HTTP Implementation: `http://10.10.10.10/:01:CHAN:1:ATT?`

See Also

Set Attenuation

4.3 (c) - Set Start-Up Attenuation Mode

Description

Sets the start-up mode to be used by all channels within the RS4DAT / RS8DAT attenuator block, this specifies how the initial attenuation value will be chosen when the system is powered up.

Command Syntax

```
:[address]:STARTUPATT:INDICATOR:[Mode]
```

Variable	Value	Description
[Mode]	L	Last Attenuation - The attenuation will be set to the same level as when the device was last powered off
	F	Fixed Attenuation - The attenuation will be set to a user-defined value
	N	Default - The attenuator will assume the factory default state (maximum attenuation)

Return String

```
:[address]:[Status]
```

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

Examples

String to Send	String Returned
:01:STARTUPATT:INDICATOR:L	:01:1
:02:STARTUPATT:INDICATOR:F	:02:1
:SL:STARTUPATT:INDICATOR:N	:SL:STARTUPATT:INDICATOR:N

HTTP Implementation: `http://10.10.10.10/:01:STARTUPATT:INDICATOR:F`

See Also

Get Start-Up Attenuation Mode
Store Last Attenuation Value
Set Channel Start-Up Attenuation
Get Channel Start-Up Attenuation

4.3 (d) - Get Start-Up Attenuation Mode

Description

Returns the start-up mode currently in use for a single RS4DAT / RS8DAT attenuator block; this specifies how the initial attenuation value will be chosen when the system is powered up.

Command Syntax

```
:[address]:STARTUPATT:INDICATOR?
```

Return String

```
:[address]:[Mode]
```

Variable	Value	Description
[Mode]	L	Last Attenuation - The attenuation will be set to the same level as when the device was last powered off
	F	Fixed Attenuation - The attenuation will be set to a user-defined value
	N	Default - The attenuator will assume the factory default state (maximum attenuation)

Examples

String to Send	String Returned
:01:STARTUPATT:INDICATOR?	:01:L
:02:STARTUPATT:INDICATOR?	:02:F
:SL:STARTUPATT:INDICATOR?	:SL:STARTUPATT:INDICATOR?

HTTP Implementation: `http://10.10.10.10/:01:STARTUPATT:INDICATOR?`

See Also

Set Start-Up Attenuation Mode
Store Last Attenuation Value
Set Channel Start-Up Attenuation
Get Channel Start-Up Attenuation

4.3 (e) - Store Last Attenuation Value

Description

Saves the final attenuation values to internal memory; necessary before powering off the device when it has been configured to load the last known attenuation on next power up. If this is not the last command before powering off in this mode, the attenuators will re-start with the maximum attenuation value when it is next powered on.

Command Syntax

```
:SL:LASTATT:STORE:INITIATE
```

Return String

```
:SL:LASTATT:STORE:INITIATE
```

Examples

String to Send	String Returned
:SL:LASTATT:STORE:INITIATE	:SL:LASTATT:STORE:INITIATE

HTTP Implementation: `http://10.10.10.10/:SL:LASTATT:STORE:INITIATE`

See Also

Set Start-Up Attenuation Mode
Get Start-Up Attenuation Mode
Set Channel Start-Up Attenuation
Get Channel Start-Up Attenuation

4.3 (f) - 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 RS4DAT / RS8DAT attenuator blocks. Applies when the attenuator block has been configured to start-up with a fixed attenuation value.

Command Syntax

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

Variable	Description
[channel]	The attenuator channel number within the RS4DAT / RS8DAT attenuator block from 1 (channel A) to 8 (channel H).
[att]	The initial attenuation level to be loaded on start-up

Return String

```
:[address]:[status]
```

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

Examples

Channels	String to Send	String Returned
RS4DAT 1, channel 1	:01:CHAN:1:STARTUPATT:VALUE:0	:01:1
RS4DAT 1, channels 1-4	:01:CHAN:1:2:3:4:STARTUPATT:VALUE:0	:01:1
All channels	:SL:CHAN:1:2:3:4:STARTUPATT:VALUE:0	SL:CHAN:1:2:3:4:STARTUPATT:VALUE:0

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

See Also

Set Start-Up Attenuation Mode
Get Start-Up Attenuation Mode
Get Channel Start-Up Attenuation

4.3 (g) - 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). Applies when the RS4DAT / RS8DAT attenuator block has been configured to start-up with a fixed attenuation value.

Command Syntax

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

Variable	Description
[channel]	The attenuator channel number within the RS4DAT / RS8DAT attenuator block from 1 (channel A) to 8 (channel H)

Return String

```
:[address]:[att]
```

Variable	Description
[att]	The intial attenuation value (dB) of the requested channel

Examples

String to Send	String Returned
:01:CHAN:1:STARTUPATT:VALUE?	:01:12.75
:01:CHAN:4:STARTUPATT:VALUE?	:01:12.75

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

See Also

Set Start-Up Attenuation Mode
Get Start-Up Attenuation Mode
Set Channel Start-Up Attenuation

4.3 (h) - Set Attenuator Label

Description

Sets a custom text label for a specific attenuator channel.

Command Syntax

```
:[address]:CHAN:[channel]:LABEL:[label_text]
```

Variable	Description
[channels]	The attenuator channel number within the RS4DAT / RS8DAT attenuator block from 1 (channel A) to 8 (channel H).
[label_text]	The text label to apply to this attenuator channel

Return String

```
:[address]:[status]
```

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

Examples

Channels	String to Send	String Returned
RS4DAT 1, channel 1	:01:CHAN:1:LABEL:LTE Test	:01:1
RS4DAT 2, channel 4	:02:CHAN:4:LABEL:Wi-Fi Test	:02:1

HTTP Implementation: `http://10.10.10.10/:01:CHAN:1:LABEL:LTE Test`

See Also

4.3 (i) - Read Attenuator Label

Description

Returns the custom text label for a specific attenuator channel.

Command Syntax

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

Variable	Description
`[channel]`	The attenuator channel number within the 4-channel attenuator block from 1 (channel A) to 8 (channel H)

Return String

```
:[address]:[label_text]
```

Variable	Description
`[label_text]`	The text label for this attenuator channel

Examples

String to Send	String Returned
`:01:CHAN:1:LABEL?`	`:01:LTE Test`
`:02:CHAN:4:LABEL?`	`:02:Wi-Fi Test`

HTTP Implementation: `http://10.10.10.10/:01:CHAN:1:LABEL?`

See Also

Set Attenuator Label
Read Attenuator Label

4.4 - Attenuation Sweeping / Fading

These commands and queries allow the system to be configured to run an automated sweep / fading sequence with any combination of attenuator channels.

```
:[command]:[value]
    [command]
```
- The command / query to send

```
    [value]
```
- The value to set (if applicable)

	Description	Command/Query
a	Sweep - Set Direction	SWEEP:DIRECTION:[Direction]
b	Sweep - Get Direction	SWEEP:DIRECTION?
c	Sweep - Set Dwell Units	SWEEP:DWELL_UNIT:[Units]
d	Sweep - Set Dwell Time	SWEEP:DWELL:[Time]
e	Sweep - Get Dwell Time	SWEEP:DWELL?
f	Sweep - Set Number of Channels	SWEEP:NOOFCHANNELS:[Number]
g	Sweep - Get Number of Channels	SWEEP:NOOFCHANNELS?
h	Sweep - Index Channel	SWEEP:CHANNEL_INDEX:[Index]
i	Sweep - Set Channel Address	SWEEP:CHANNEL_ADDRESS:[Address]
j	Sweep - Get Channel Address	SWEEP:CHANNEL_ADDRESS?
k	Sweep - Set Start Attenuation	SWEEP:START:[Att]
l	Sweep - Get Start Attenuation	SWEEP:START?
m	Sweep - Set Stop Attenuation	SWEEP:STOP:[Att]
n	Sweep - Get Stop Attenuation	SWEEP:STOP?
o	Sweep - Set Step Size	SWEEP:STEPSIZE:[Att]
p	Sweep - Get Step Size	SWEEP:STEPSIZE?
q	Sweep - Turn On / Off	SWEEP:MASTERMODE:[on_off]

Once an attenuation sequence is programmed and enabled, it is managed by the attenuator's internal microprocessor; this supports very fast sequences with minimum dwell times in the order of 600 µs. It is not possible to query any attenuator parameters whilst the sequence is active so any subsequent command / query to the device will disable the sequence.

An alternative implementation method is to control the sequence and timing from your program, only sending "set attenuation" commands to the attenuator at the appropriate times. The advantage of this approach is that the program is able to query and keep track of the current attenuation state. The disadvantage is that the communication delays inherent in USB / Ethernet communication dictate a minimum dwell time in the order of milliseconds with this approach, rather than microseconds.

An example sequence of commands to configure a sweep sequence is shown below:

```
:SWEEP:DIRECTION:0            // Sweep from start value to stop value
:SWEEP:DWELL_UNIT:U           // Set the dwell time in microseconds
:SWEEP:DWELL:800              // Set a dwell time of 800 µs
:SWEEP:CHAN:1:START:0         // Set start value for channel 1 (0 dB)
:SWEEP:CHAN:1:STOP:65         // Set stop value for channel 1 (65 dB)
:SWEEP:CHAN:1:STEPSIZE:0.25   // Set step size for chan1 (0.25 dB)
:SWEEP:NOOFCHANNELS:3         // Sweep on 3 attenuator channels

// Set the first channel to include
:SWEEP:CHANNEL_INDEX:0        // Index the first channel slot
:SWEEP:CHANNEL_ADDRESS:01D    // Include channel 01D in the sweep

// Set the second channel to include
:SWEEP:CHANNEL_INDEX:0        // Index the second channel slot
:SWEEP:CHANNEL_ADDRESS:02A    // Include channel 02A in the sweep

// Set the third channel to include
:SWEEP:CHANNEL_INDEX:0        // Index the third channel slot
:SWEEP:CHANNEL_ADDRESS:02B    // Include channel 02B in the sweep

:SWEEP:MASTERMODE:ON                // Enable the sweep

// Any subsequent command / query sent will stop the sequence
```

4.4 (a) - Sweep Mode - Set Sweep Direction

Description

Sets the direction in which the attenuation level will sweep.

Command Syntax

```
:SWEEP:DIRECTION:[Direction]
```

Variable	Value	Description
[Direction]	0	Forward - Sweep from "Start" to "Stop" value
	1	Backwards - Sweep from "Stop" to "Start" value
	2	Bi-directionally - Sweep in the forward and then reverse directions

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:DIRECTION:0	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:DIRECTION:0`

See Also

Sweep Mode - Get Sweep Direction

4.4 (b) - Sweep Mode - Get Sweep Direction

Description

Returns the direction in which the attenuation level will sweep.

Command Syntax

```
:SWEEP:DIRECTION?
```

Return String

```
[Direction]
```

Variable	Value	Description
[Direction]	0	Forward - Sweep from "Start" to "Stop" value
	1	Backwards - Sweep from "Stop" to "Start" value
	2	Bi-directionally - Sweep in the forward and then reverse directions

Examples

String to Send	String Returned
:SWEEP:DIRECTION?	0

HTTP Implementation: `http://10.10.10.10/:SWEEP:DIRECTION?`

See Also

Sweep Mode - Set Sweep Direction

4.4 (c) - Sweep Mode - Set Dwell Time Units

Description

Sets the units to be used for the sweep dwell time.

Command Syntax

```
:SWEEP:DWELL_UNIT:[Units]
```

Variable		Description
[Units]	U	Dwell time in microseconds (µs)
	M	Dwell time in milliseconds (ms)
	S	Dwell time in seconds (s)

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:DWELL_UNIT:U	1
:SWEEP:DWELL_UNIT:M	1
:SWEEP:DWELL_UNIT:S	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:DWELL_UNIT:U`

See Also

Sweep Mode - Set Point Dwell Time
Sweep Mode - Get Dwell Time

4.4 (d) - Sweep Mode - Set Dwell Time

Description

Sets the dwell time to be used for the sweep. The dwell time units are defined separately.

Command Syntax

```
:SWEEP:DWELL:[Time]
```

Variable	Description
[Time]	The dwell time of the indexed point

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:DWELL:650	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:DWELL:650`

See Also

Sweep Mode - Set Dwell Time Units
Sweep Mode - Get Dwell Time

4.4 (e) - Sweep Mode - Get Dwell Time

Description

Gets the dwell time (including the units) of the attenuation sweep.

Command Syntax

```
:SWEEP:DWELL?
```

Return String

```
[Dwell] [Units]
```

Variable	Value	Description
[Dwell]		The dwell time of the sweep
[Units]	uSec	Dwell time in microseconds (µs)
	mSec	Dwell time in milliseconds (ms)
	Sec	Dwell time in seconds (s)

Examples

String to Send	String Returned
:SWEEP:DWELL?	625 uSec
:SWEEP:DWELL?	50 mSec
:SWEEP:DWELL?	2 Sec

HTTP Implementation: `http://10.10.10.10/:SWEEP:DWELL?`

See Also

Sweep Mode - Set Dwell Time Units
Sweep Mode - Set Dwell Time

4.4 (f) - Sweep Mode - Set Number of Channels

Description

Sets the number of channels on which the sweep should run.

Command Syntax

```
:SWEEP:NOOFCHANNELS:[Number]
```

Variable	Description
[Number]	The integer number of channels to be included in the sweep

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:NOOFCHANNELS:3	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:NOOFCHANNELS:11`

See Also

Sweep Mode - Get Number of Channels
Sweep Mode - Index Channel
Sweep Mode - Set Channel Address
Sweep Mode - Get Channel Address

4.4 (g) - Sweep Mode - Get Number of Channels

Description

Returns the number of channels on which the sweep is set to run.

Command Syntax

```
:SWEEP:NOOFCHANNELS?
```

Return String

```
[Number]
```

Variable	Description
[Number]	The integer number of channels to be included in the sweep

Examples

String to Send	String Returned
:SWEEP:NOOFCHANNELS?	3

HTTP Implementation: `http://10.10.10.10/:SWEEP:NOOFCHANNELS?`

See Also

Sweep Mode - Set Number of Channels
Sweep Mode - Index Channel
Sweep Mode - Set Channel Address
Sweep Mode - Get Channel Address

4.4 (h) - Sweep Mode - Index Channel

Description

Indexes one of the channel slots in the sweep so that a specific channel can be assigned.

Command Syntax

```
:SWEEP:CHANNEL_INDEX:[Index]
```

Variable	Description
[Index]	The index number of the channel slot to define (from 0 to [Number of Channels] - 1)

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:CHANNEL_INDEX:0	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:CHANNEL_INDEX:0`

See Also

Sweep Mode - Set Number of Channels
Sweep Mode - Get Number of Channels
Sweep Mode - Set Channel Address
Sweep Mode - Get Channel Address

4.4 (i) - Sweep Mode - Set Channel Address

Description

Assign a specific attenuator channel to be included in the sweep. Note one of the available slots within the sweep must first be indexed.

Command Syntax

```
:SWEEP:CHANNEL_ADDRESS:[Address]
```

Variable	Description
[Address]	The alpha-numeric representation of an attenuator channel within the system. For example "02C" for channel 3 (C) within the RS4DAT / RS8DAT attenuator block at address 2.

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:CHANNEL_ADDRESS:02C	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:CHANNEL_ADDRESS:02C`

See Also

Sweep Mode - Set Number of Channels
Sweep Mode - Get Number of Channels
Sweep Mode - Index Channel
Sweep Mode - Get Channel Address

4.4 (j) - Sweep Mode - Get Channel Address

Description

Returns the attenuator channel to be included in the sweep at the current indexed slot.

Command Syntax

```
:SWEEP:CHANNEL_ADDRESS?
```

Return String

```
[Address]
```

Variable	Description
[Address]	The alpha-numeric representation of an attenuator channel within the system. For example “02C” for channel 3 (C) within the RS4DAT / RS8DAT attenuator block at address 2.

Examples

String to Send	String Returned
:SWEEP:CHANNEL_ADDRESS?	02C

HTTP Implementation: `http://10.10.10.10/:SWEEP:CHANNEL_ADDRESS?`

See Also

Sweep Mode - Set Number of Channels
Sweep Mode - Get Number of Channels
Sweep Mode - Index Channel
Sweep Mode - Set Channel Address

4.4 (k) - Sweep Mode - Set Start Attenuation

Description

Sets the first attenuation level to be loaded during the sweep.

Command Syntax

```
:SWEEP:START:[Att]
```

Variable	Description
[Att]	The initial attenuation level to set

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:START:0	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:START:0`

See Also

Sweep Mode - Get Start Attenuation
Sweep Mode - Set Stop Attenuation
Sweep Mode - Set Step Size

4.4 (l) - Sweep Mode - Get Start Attenuation

Description

Returns the first attenuation level to be loaded during the sweep.

Command Syntax

```
:SWEEP:START?
```

Return String

```
[Attenuation]
```

Variable	Description
[Attenuation]	The initial attenuation level to be set during the sweep

Examples

String to Send	String Returned
:SWEEP:START?	0.0

HTTP Implementation: `http://10.10.10.10/:SWEEP:START?`

See Also

Sweep Mode - Set Start Attenuation
Sweep Mode - Get Stop Attenuation
Sweep Mode - Get Step Size

4.4 (m) - Sweep Mode - Set Stop Attenuation

Description

Sets the final attenuation level to be loaded during the sweep.

Command Syntax

```
:SWEEP:STOP:[Att]
```

Variable	Description
[Att]	The final attenuation level to set

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:STOP:65.75	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:STOP:65.75`

See Also

Sweep Mode - Get Stop Attenuation
Sweep Mode - Set Start Attenuation
Sweep Mode - Set Step Size

4.4 (n) - Sweep Mode - Get Stop Attenuation

Description

Returns the final attenuation level to be loaded during the sweep.

Command Syntax

```
:SWEEP:STOP?
```

Return String

```
[Attenuation]
```

Variable	Description
[Attenuation]	The final attenuation level to be set during the sweep

Examples

String to Send	String Returned
:SWEEP:STOP?	65.75

HTTP Implementation: `http://10.10.10.10/:SWEEP:STOP?`

See Also

Sweep Mode - Set Stop Attenuation
Sweep Mode - Get Start Attenuation
Sweep Mode - Get Step Size

4.4 (o) - Sweep Mode - Set Step Size

Description

Sets the attenuation step size that will be used to increment the attenuation from the start to stop levels (or decrement from stop to start if the sweep is running in the reverse direction).

Command Syntax

```
:SWEEP:STEPSIZE:[Att]
```

Variable	Description
[Att]	The attenuation step size

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:STEPSIZE:0.5	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:STEPSIZE:0.5`

See Also

Sweep Mode - Get Step Size
Sweep Mode - Set Start Attenuation
Sweep Mode - Set Stop Attenuation

4.4 (p) - Sweep Mode - Get Step Size

Description

Returns the attenuation step size that will be used to increment the attenuation from the start to stop levels (or decrement from stop to start if the sweep is running in the reverse direction).

Command Syntax

```
:SWEEP:STEPSIZE?
```

Return String

```
[Attenuation]
```

Variable	Description
[Attenuation]	The attenuation step size

Examples

String to Send	String Returned
:SWEEP:STEPSIZE?	0.50

HTTP Implementation: `http://10.10.10.10/:SWEEP:STEPSIZE?`

See Also

Sweep Mode - Set Step Size
Sweep Mode - Get Start Attenuation
Sweep Mode - Get Stop Attenuation

4.4 (q) - Sweep Mode - Turn On / Off

Description

Enables or disable the attenuation sweep sequence according to the parameters set.

Notes:

- Once an attenuation sequence is programmed and enabled, it is managed by the system's internal microprocessor; this supports very fast sequences with minimum dwell times in the order of 600 µs. It is not possible to query any attenuator parameters whilst the sequence is active so any subsequent command / query to the device will disable the sequence.
- An alternative implementation method is to control the sequence and timing from your program, only sending "set attenuation" commands to the attenuator at the appropriate times. The advantage of this approach is that the program is able to query and keep track of the current attenuation state. The disadvantage is that the communication delays inherent in USB / Ethernet communication dictate a minimum dwell time in the order of milliseconds with this approach, rather than microseconds.

Command Syntax

```
:SWEEP:MASTERMODE:[On_Off]
```

Variable	Value	Description
[On_Off]	ON	Enable the sweep sequence according to the previously configured parameters
	OFF	Disable the sweep sequence

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:SWEEP:MASTERMODE:ON	1
:SWEEP:MASTERMODE:OFF	1

HTTP Implementation: `http://10.10.10.10/:SWEEP:MASTERMODE:ON`

4.5 - Attenuation Hopping

These commands and queries allow the system to be configured to run an automated sweep / fading sequence with any combination of attenuator channels.

```
:[command]:[value]
      [command]
```
- The command / query to send

```
      [value]
```
- The value to set (if applicable)

	Description	Command/Query
a	Hop - Set Number of Points	:HOP:POINTS:[NoOfPoints]
b	Hop - Get Number of Points	:HOP:POINTS?
c	Hop - Set Sequence Direction	:HOP:DIRECTION:[Direction]
d	Hop - Get Sequence Direction	:HOP:DIRECTION?
e	Hop - Set Indexed Point	:HOP:POINT:[PointNo]
f	Hop - Get Indexed Point	:HOP:POINT?
g	Hop - Set Point Dwell Units	:HOP:DWELL_UNIT:[Units]
i	Hop - Set Point Dwell Time	:HOP:DWELL:[Time]
j	Hop - Get Point Dwell Time	:HOP:DWELL?
k	Hop - Set Point Attenuation	:HOP:ATT:[Att]
l	Hop - Get Point Attenuation	:HOP:ATT?
m	Hop - Set Number of Channels	:HOP:NOOFCHANNELS:[Number]
n	Hop - Get Number of Channels	:HOP:NOOFCHANNELS?
o	Hop - Index Channel	:HOP:CHANNEL_INDEX[Index]
p	Hop - Set Channel Address	:HOP:CHANNEL_ADDRESS:[Address]
q	Hop - Get Channel Address	:HOP:CHANNEL_ADDRESS?
r	Hop - Turn On / Off	:HOP:MASTERMODE:[on_off]

Once an attenuation sequence is programmed and enabled, it is managed by the system's internal microprocessor; this supports very fast sequences with minimum dwell times in the order of 600 µs. It is not possible to query any attenuator parameters whilst the sequence is active so any subsequent command / query to the device will disable the sequence.

An alternative implementation method is to control the sequence and timing from your program, only sending "set attenuation" commands to the attenuator at the appropriate times. The advantage of this approach is that the program is able to query and keep track of the current attenuation state. The disadvantage is that the communication delays inherent in USB / Ethernet communication dictate a minimum dwell time in the order of milliseconds with this approach, rather than microseconds.

An example sequence of commands to configure a hop sequence is shown below:

```
:HOP:POINTS:10          // Create a hop sequence with 10 points
:HOP:DIRECTION:0        // Hop from first to last value in the list

// Set the first hop point
:HOP:POINT:0            // Index the first point in the sequence
:HOP:DWELL_UNIT:U       // Set the dwell time in microseconds
:HOP:DWELL:800          // Set a dwell time of 800 µs
:HOP:ATT:10             // Set the first point to 10 dB
:HOP:NOOCHANNELS:2      // Include 2 channels for the first point

:HOP:CHANNEL_INDEX:0    // Index the first channel slot
:HOP:CHANNEL_ADDRESS:01D      // Include channel 01D in this slot

:HOP:CHANNEL_INDEX:1    // Index the second channel slot
:HOP:CHANNEL_ADDRESS:02A      // Include channel 02A in this slot

// Set the second hop point
:HOP:POINT:1            // Index the first point in the sequence
:HOP:DWELL_UNIT:U       // Set the dwell time in microseconds
:HOP:DWELL:1600         // Set a dwell time of 1600 µs
:HOP:ATT:15             // Set the second point to 15 dB
:HOP:NOOCHANNELS:2      // Include 2 channels for the first point

:HOP:CHANNEL_INDEX:0    // Index the first channel slot
:HOP:CHANNEL_ADDRESS:01D      // Include channel 01D in this slot

:HOP:CHANNEL_INDEX:1    // Index the second channel slot
:HOP:CHANNEL_ADDRESS:02A      // Include channel 02A in this slot

// Set the third / final hop point
:HOP:POINT:2            // Index the first point in the sequence
:HOP:DWELL_UNIT:U       // Set the dwell time in microseconds
:HOP:DWELL:800          // Set a dwell time of 800 µs
:HOP:ATT:20             // Set the final point to 20 dB
:HOP:NOOCHANNELS:2      // Include 2 channels for the first point

:HOP:CHANNEL_INDEX:0    // Index the first channel slot
:HOP:CHANNEL_ADDRESS:01D      // Include channel 01D in this slot

:HOP:CHANNEL_INDEX:1    // Index the second channel slot
:HOP:CHANNEL_ADDRESS:02A      // Include channel 02A in this slot


:HOP:MASTERMODE:ON      // Enable the hop sequence

// Any subsequent command / query sent will stop the sequence
```

4.5 (a) - Hop Mode - Set Number of Points

Description

Sets the number of points to be used in the attenuation hop sequence (from 1 to 100).

Command Syntax

```
:HOP:POINTS:[NoOfPoints]
```

Variable	Value	Description
[NoOfPoints]	1-100	The number of points to set in the hop sequence

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:POINTS:10	1

HTTP Implementation: `http://10.10.10.10/:HOP:POINTS:10`

See Also

Hop Mode - Get Number of Points
Hop Mode - Set Sequence Direction
Hop Mode - Get Sequence Direction

4.5 (b) - Hop Mode - Get Number of Points

Description

Returns the number of points to be used in the attenuation hop sequence.

Command Syntax

```
:HOP:POINTS?
```

Return String

```
[NoOfPoints]
```

Variable	Value	Description
[NoOfPoints]	1-1000	The number of points in the hop sequence

Examples

String to Send	String Returned
:HOP:POINTS?	10

HTTP Implementation: `http://10.10.10.10/:HOP:POINTS?`

See Also

Hop Mode - Set Number of Points
Hop Mode - Set Sequence Direction
Hop Mode - Get Sequence Direction

4.5 (c) - Hop Mode - Set Sequence Direction

Description

Sets the direction in which the attenuator will progress through the list of attenuation hops.

Command Syntax

```
:HOP:DIRECTION:[Direction]
```

Variable	Value	Description
[Direction]	0	Forward - The list of attenuation hops will be loaded from index 1 to index n
	1	Backwards - The list of attenuation hops will be loaded from index n to index 1
	2	Bi-directionally - The list of attenuation hops will be loaded in the forward and then reverse directions

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:DIRECTION:0	1

HTTP Implementation: `http://10.10.10.10/:HOP:DIRECTION:0`

See Also

Hop Mode - Set Number of Points
Hop Mode - Get Number of Points
Hop Mode - Get Sequence Direction

4.5 (d) - Hop Mode - Get Sequence Direction

Description

Returns the direction in which the attenuator will progress through the list of attenuation hops.

Command Syntax

```
:HOP:DIRECTION?
```

Return String

```
[Direction]
```

Variable	Value	Description
[Direction]	0	Forward - The list of attenuation hops will be loaded from index 1 to index n
	1	Backwards - The list of attenuation hops will be loaded from index n to index 1
	2	Bi-directionally - The list of attenuation hops will be loaded in the forward and then reverse directions

Examples

String to Send	String Returned
:HOP:DIRECTION?	0

HTTP Implementation: `http://10.10.10.10/:HOP:DIRECTION?`

See Also

Hop Mode - Set Number of Points
Hop Mode - Get Number of Points
Hop Mode - Set Sequence Direction

4.5 (e) - Hop Mode - Set Indexed Point

Description

Defines which point in the hop sequence is currently indexed, this allows the parameters for that point to be configured (attenuation value, dwell time and included channels).

Command Syntax

```
:HOP:POINT:[PointNo]
```

Variable	Description
[PointNo]	The index number of the point in the hop sequence, from 0 to (n-1), where n is the number of points in the sequence

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:POINT:3	1

HTTP Implementation: `http://10.10.10.10/:HOP:POINT:3`

See Also

Hop Mode - Get Indexed Point

4.5 (f) - Hop Mode - Get Indexed Point

Description

Returns the number of the indexed attenuation point within the hop sequence.

Command Syntax

```
:HOP:POINT?
```

Return String

```
[PointNo]
```

Variable	Description
[PointNo]	The index number of the point in the hop sequence, from 0 to (n-1), where n is the number of points in the sequence

Examples

String to Send	String Returned
:HOP:POINT?	3

HTTP Implementation: `http://10.10.10.10/:HOP:POINT?`

See Also

Hop Mode - Set Indexed Point

4.5 (g) - Hop Mode - Set Point Dwell Time Units

Description

Sets the units to be used for the dwell time of the indexed point in the hop sequence.

Command Syntax

```
:HOP:DWELL_UNIT:[Units]
```

Variable		Description
[Units]	U	Dwell time in microseconds (µs)
	M	Dwell time in milliseconds (ms)
	S	Dwell time in seconds (s)

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:DWELL_UNIT:U	1
:HOP:DWELL_UNIT:M	1
:HOP:DWELL_UNIT:S	1

HTTP Implementation: `http://10.10.10.10/:HOP:DWELL_UNIT:U`

See Also

Hop Mode - Set Point Dwell Time
Hop Mode - Get Point Dwell Time

4.5 (h) - Hop Mode - Set Point Dwell Time

Description

Sets the dwell time of the indexed point in the hop sequence. The dwell time units are defined separately.

Command Syntax

```
:HOP:DWELL:[Time]
```

Variable	Description
[Time]	The dwell time of the indexed point

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:DWELL:650	1

HTTP Implementation: `http://10.10.10.10/:HOP:DWELL:650`

See Also

Hop Mode - Set Point Dwell Time Units
Hop Mode - Get Point Dwell Time

4.5 (i) - Hop Mode - Get Point Dwell Time

Description

Gets the dwell time (including the units) of the indexed point in the hop sequence.

Command Syntax

```
:HOP:DWELL?
```

Return String

```
[Dwell]
```

Variable	Description
[Dwell]	The dwell time of the indexed point

Examples

String to Send	String Returned
:HOP:DWELL?	625 uSec
:HOP:DWELL?	50 mSec
:HOP:DWELL?	2 Sec

HTTP Implementation: `http://10.10.10.10/:HOP:DWELL?`

See Also

Hop Mode - Set Point Dwell Time Units
Hop Mode - Set Point Dwell Time

4.5 (j) - Hop Mode - Set Point Attenuation

Description

Sets the attenuation to be used for the indexed hop point.

Command Syntax

```
:HOP:ATT:[Att]
```

Variable	Description
[Att]	The attenuation of the indexed hop point

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:ATT:75.5	1

HTTP Implementation: http://10.10.10.10/:HOP:ATT:75.5

See Also

Hop Mode - Get Point Attenuation

4.5 (k) - Hop Mode - Get Point Attenuation

Description

Returns the attenuation to be used for the indexed hop point.

Command Syntax

```
:HOP:ATT?
```

Return String

```
[Attenuation]
```

Variable	Description
[Attenuation]	The attenuation of the indexed hop point

Examples

String to Send	String Returned
:HOP:ATT?	75.50

HTTP Implementation: `http://10.10.10.10/:HOP:ATT?`

See Also

Hop Mode - Set Point Attenuation

4.5 (l) - Hop Mode - Set Number of Channels

Description

Sets the number of channels to be set for the hop point currently indexed.

Command Syntax

```
:HOP:NOOFCHANNELS:[Number]
```

Variable	Description
[Number]	The integer number of channels to be set at this indexed hop point

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:NOOFCHANNELS:3	1

HTTP Implementation: `http://10.10.10.10/:HOP:NOOFCHANNELS:11`

See Also

Hop Mode - Get Number of Channels
Hop Mode - Index Channel
Hop Mode - Set Channel Address
Hop Mode - Get Channel Address

4.5 (m) - Hop Mode - Get Number of Channels

Description

Returns the number of channels to be set for the hop point currently indexed

Command Syntax

```
:HOP:NOOFCHANNELS?
```

Return String

```
[Number]
```

Variable	Description
[Number]	The integer number of channels to be set at this indexed hop point

Examples

String to Send	String Returned
:HOP:NOOFCHANNELS?	3

HTTP Implementation: `http://10.10.10.10/:HOP:NOOFCHANNELS?`

See Also

Hop Mode - Set Number of Channels
Hop Mode - Index Channel
Hop Mode - Set Channel Address
Hop Mode - Get Channel Address

4.5 (n) - Hop Mode - Index Channel

Description

Indexes one of the channel slots for the current indexed hop point so that a specific channel can be assigned.

Command Syntax

```
:HOP:CHANNEL_INDEX:[Index]
```

Variable	Description
[Index]	The index number of the channel slot to define (from 0 to [Number of Channels] - 1)

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:CHANNEL_INDEX:0	1

HTTP Implementation: `http://10.10.10.10/:HOP:CHANNEL_INDEX:0`

See Also

Hop Mode - Set Number of Channels
Hop Mode - Get Number of Channels
Hop Mode - Set Channel Address
Hop Mode - Get Channel Address

4.5 (o) - Hop Mode - Set Channel Address

Description

Assign a specific attenuator channel to be included in the current indexed hop point. Note one of the available slots within the hop point must first be indexed.

Command Syntax

```
:HOP:CHANNEL_ADDRESS:[Address]
```

Variable	Description
`[Address]`	The alpha-numeric representation of an attenuator channel within the system. For example "02C" for channel 3 (C) within the RS4DAT / RS8DAT attenuator block at address 2.

Return String

```
[Status]
```

Variable	Value	Description
`[Status]`	0	Command failed
	1	Command completed successfully

Examples

String to Send	String Returned
`:HOP:CHANNEL_ADDRESS:02C`	1

HTTP Implementation: `http://10.10.10.10/:HOP:CHANNEL_ADDRESS:02C`

See Also

Hop Mode - Set Number of Channels
Hop Mode - Get Number of Channels
Hop Mode - Index Channel
Hop Mode - Get Channel Address

4.5 (p) - Hop Mode - Get Channel Address

Description

Returns the attenuator channel to be included in the current indexed hop point.

Command Syntax

```
:HOP:CHANNEL_ADDRESS?
```

Return String

```
[Address]
```

Variable	Description
[Address]	The alpha-numeric representation of an attenuator channel within the system. For example “02C” for channel 3 (C) within the RS4DAT / RS8DAT attenuator block at address 2.

Examples

String to Send	String Returned
:HOP:CHANNEL_ADDRESS?	02C

HTTP Implementation: `http://10.10.10.10/:HOP:CHANNEL_ADDRESS?`

See Also

Hop Mode - Set Number of Channels
Hop Mode - Get Number of Channels
Hop Mode - Index Channel
Hop Mode - Set Channel Address

4.5 (q) - Hop Mode - Turn On / Off

Description

Enables or disables the hop sequence according to the previously configured parameters.

Notes:

- Once an attenuation sequence is programmed and enabled, it is managed by the system's internal microprocessor; this supports very fast sequences with minimum dwell times in the order of 600 µs. It is not possible to query any attenuator parameters whilst the sequence is active so any subsequent command / query to the device will disable the sequence.
- An alternative implementation method is to control the sequence and timing from your program, only sending "set attenuation" commands to the attenuator at the appropriate times. The advantage of this approach is that the program is able to query and keep track of the current attenuation state. The disadvantage is that the communication delays inherent in USB / Ethernet communication dictate a minimum dwell time in the order of milliseconds with this approach, rather than microseconds.

Command Syntax

```
:HOP:MASTERMODE:[On_Off]
```

Variable	Value	Description
[On_Off]	ON	Enable the hop sequence using the previously configured list of attenuation hops
	OFF	Disable the hop sequence

Return String

```
[Status]
```

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

Examples

String to Send	String Returned
:HOP:MASTERMODE:ON	1
:HOP:MASTERMODE:OFF	1

HTTP Implementation: `http://10.10.10.10/:HOP:MASTERMODE:ON`

4.6 - Ethernet Configuration Commands

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

4.6 (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]
```

Variable	Description
[ip]	The static IP address to be used; must be valid and available on the network

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:IP:192.100.1.1	1

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

4.6 (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]
```

Variable	Description
[ip]	The static IP address to be used

Examples

String to Send	String Returned
:ETHERNET:CONFIG:IP?	192.100.1.1

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

4.6 (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]
```

Variable	Description
[mask]	The subnet mask for communication on the network

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SM:255.255.255.0	1

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

4.6 (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]

Variable	Description
[mask]	The subnet mask for communication on the network

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SM?	255.255.255.0

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

4.6 (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]
```

Variable	Description
[gateway]	IP address of the network gateway

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:NG:192.100.1.0	1

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

4.6 (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]
```

Variable	Description
[gateway]	IP address of the network gateway

Examples

String to Send	String Returned
:ETHERNET:CONFIG:NG?	192.168.1.0

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

4.6 (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]
```

Variable	Description
[port]	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.

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:HTPORT:8080	1

HTTP Implementation:

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

See Also

Get HTTP Port
Set SSH Port
Set Telnet Port
Update Ethernet Settings

4.6 (h) - Get HTTP Port

Description

Gets the IP port to be used for HTTP communication.

Command Syntax

```
:ETHERNET:CONFIG:HTPORT?
```

Return String

```
[port]
```

Variable	Description
[port]	IP port to be used for HTTP communication

Examples

String to Send	String Returned
:ETHERNET:CONFIG:HTPORT?	8080

HTTP Implementation:

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

See Also

Set HTTP Port
Get SSH Port
Get Telnet Port

4.6 (i) - Set SSH Port

Description

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

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

```
:ETHERNET:CONFIG:SSHPORT:[port]
```

Variable	Description
[port]	IP port to be used for SSH communication. The default is port 22

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SSHPORT:21	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:SSHPORT:21
```

See Also

Get SSH Port
Set HTTP Port
Set Telnet Port
Update Ethernet Settings

4.6 (j) - Get SSH Port

Description

Gets the IP port to be used for SSH communication.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

```
:ETHERNET:CONFIG:SSHPORT?
```

Return String

```
[port]
```

Variable	Description
[port]	IP port to be used for Telnet communication

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SSHPORT?	1

HTTP Implementation:

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

See Also

Set SSH Port
Get HTTP Port
Get Telnet Port

4.6 (k) - 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]
```

Variable	Description
[port]	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.

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:TELNETPORT:21	1

HTTP Implementation:

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

See Also

Set HTTP Port
Set SSH Port
Get Telnet Port
Update Ethernet Settings

4.6 (l) - Get Telnet Port

Description

Gets the IP port to be used for Telnet communication.

Command Syntax

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

Return String

```
[port]
```

Variable	Description
[port]	IP port to be used for Telnet communication

Examples

String to Send	String Returned
:ETHERNET:CONFIG:TELNETPORT?	1

HTTP Implementation:

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

See Also

Get HTTP Port
Get SSH Port
Set Telnet Port

4.6 (m) - Set Password Requirement

Description

Sets whether or not a password is required for HTTP and Telnet communication. The password is always required for SSH communication.

Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

```
:ETHERNET:CONFIG:PWDENABLED:[enabled]
```

Variable	Value	Description
[enabled]	0	Password not required for Ethernet communication
	1	Password required for Ethernet communication

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:PWDENABLED:1	1

HTTP Implementation:

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

See Also

Get Password Requirement
Set Password
Get Password
Update Ethernet Settings

4.6 (n) - Get Password Requirement

Description

Indicates whether or not a password is required for HTTP and Telnet communication. A password is always required for SSH communication.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

```
:ETHERNET:CONFIG:PWDENABLED?
```

Return String

```
[enabled]
```

Variable	Value	Description
[enabled]	0	Password not required for Ethernet communication
	1	Password required for Ethernet communication

Examples

String to Send	String Returned
:ETHERNET:CONFIG:PWDENABLED?	1

HTTP Implementation:

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

See Also

Set Password Requirement
Set Password
Get Password

4.6 (o) - Set Password

Description

Sets the password for Ethernet communication. The password is always required for SSH communication but only for HTTP and Telnet when password security is enabled. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

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

Variable	Description
[pwd]	Password to set for Ethernet communication (not case sensitive)

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:PWD:PASS-123	1

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

4.6 (p) - Get Password

Description

Returns the password for Ethernet communication. The password is always required for SSH communication but only for HTTP and Telnet when password security is enabled.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Command Syntax

```
:ETHERNET:CONFIG:PWD?
```

Return String

```
[pwd]
```

Variable	Description
[pwd]	Password for Ethernet communication (not case sensitive)

Examples

String to Send	String Returned
:ETHERNET:CONFIG:PWD?	PASS-123

HTTP Implementation:

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

See Also

Set Password Requirement
Get Password Requirement
Set Password

4.6 (q) - 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]
```

Variable	Value	Description
[enabled]	0	DHCP disabled (static IP settings will be used)
	1	DHCP enabled (IP address will be requested from DHCP server on the network)

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:DHCPENABLED:1	1

HTTP Implementation:

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

See Also

Set Static IP Address
Get DHCP Status
Update Ethernet Settings

4.6 (r) - 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]
```

Variable	Value	Description
[enabled]	0	DHCP disabled (static IP settings will be used)
	1	DHCP enabled (IP address will be requested from DHCP server on the network)

Examples

String to Send	String Returned
:ETHERNET:CONFIG:DHCPENABLED?	1

HTTP Implementation:

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

See Also

Set Static IP Address
Set DHCP Status
Get Current Ethernet Configuration

4.6 (s) - 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]
```

Variable	Description
[mac]	MAC address of the attenuator

Examples

String to Send	String Returned
:ETHERNET:CONFIG:MAC?	D0-73-7F-82-D8-01

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

4.6 (t) - 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

String to Send	String Returned
:ETHERNET:CONFIG:LISTEN?	192.100.1.1;255.255.255.0;192.100.1.0

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

4.6 (u) - 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]
```

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

Examples

String to Send	String Returned
:ETHERNET:CONFIG:INIT	1

HTTP Implementation:

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

See Also

Get Current Ethernet Configuration

5 - Ethernet Control API

Control of the system via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed above via HTTP or Telnet. SSH is also available as an option on some models for secure communication (please contact testsolutions@minicircuits.com for details).

In addition, UDP is supported for discovering available systems on the network.

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

5.1 - Configuring Ethernet Settings

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 (used for HTTP communication), default is port 80
    - SSH port, default is port 22
    - Telnet port, default is port 23
    - 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 (used for HTTP communication), default is port 80
    - SSH port, default is port 22
    - Telnet port, default is port 23
    - Password (up to 20 characters; default is no password)

The Ethernet connection details can be configured using the commands summarized in Ethernet Configuration Commands, whilst connected by USB or Ethernet.

5.2 - HTTP Communication

The basic format of the HTTP command is:

http://ADDRESS:PORT/PWD;COMMAND

Where

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

Example 1:

http://192.168.100.100:800/PWD=123;:01:CHAN:1:SETATT:10.25

- 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?

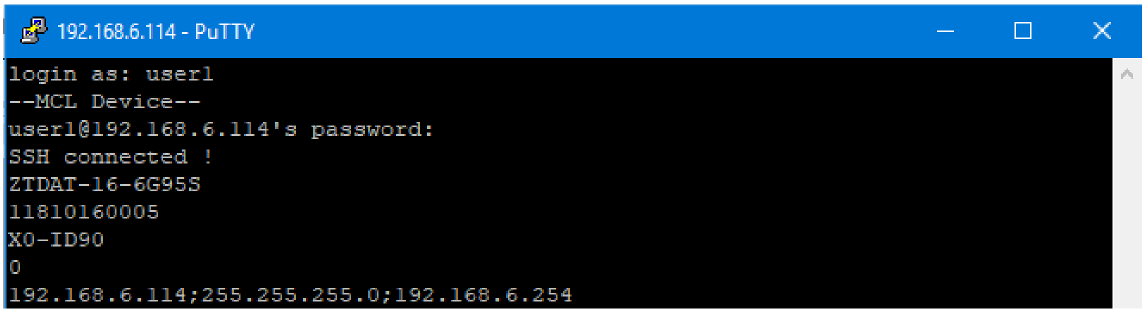
- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to query the attenuation state of the 4 channels in block 1

5.3 - SSH Communication

Note: SSH communication is not supported as standard on all models, please contact testsolutions@minicircuits.com for details.

SSH allows secure communication with the system, using the configured SSH port (default is port 22) and password. The user name is `ssh_user`.

SSH is widely supported and can be implemented in most programming environments. Alternatively, a client such as PuTTY can be used as a console to quickly establish an SSH connection and control the system.



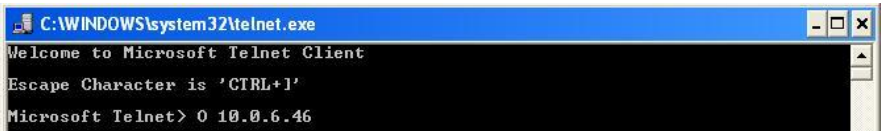
```
login as: user1
--MCL Device--
user1@192.168.6.114's password:
SSH connected !
ZTDAT-16-6G95S
11810160005
X0-ID90
0
192.168.6.114;255.255.255.0;192.168.6.254
```

5.4 - Telnet Communication

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.

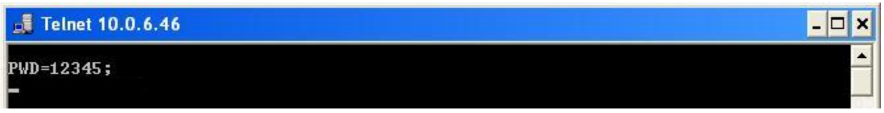
Each command must be terminated with the carriage return and line-feed characters (\r\n). Responses will be similarly terminated. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below:

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



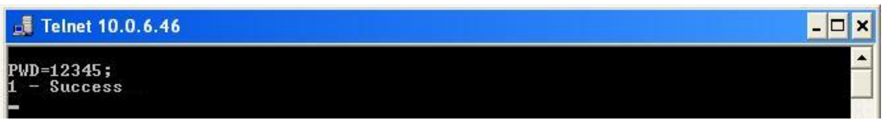
```
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+]'
Microsoft Telnet> O 10.0.6.46
```

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



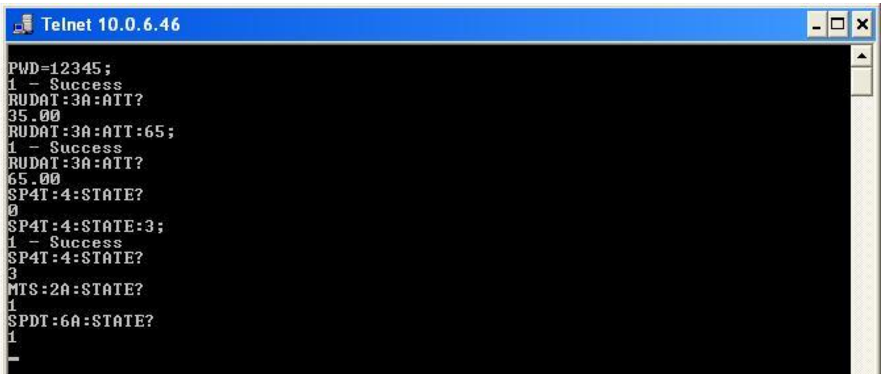
```
PWD=12345;
_
```

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:

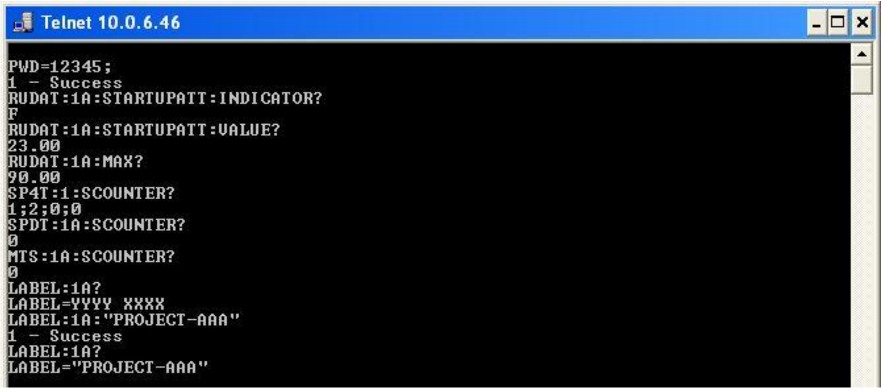


```
PWD=12345;
1 - Success
_
```

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



```
PWD=12345;
1 - Success
RUDAT:3A:ATT?
35.00
RUDAT:3A:ATT:65;
1 - Success
RUDAT:3A:ATT?
65.00
SP4T:4:STATE?
0
SP4T:4:STATE:3;
1 - Success
SP4T:4:STATE?
3
MTS:2A:STATE?
1
SPDT:6A:STATE?
1
_
```



```
PWD=12345;
1 - Success
RUDAT:1A:STARTUPATT:INDICATOR?
F
RUDAT:1A:STARTUPATT:VALUE?
23.00
RUDAT:1A:MAX?
90.00
SP4T:1:SCOUNTER?
1;2;0;0
SPDT:1A:SCOUNTER?
0
MTS:1A:SCOUNTER?
0
LABEL:1A?
LABEL=YYYY XXXX
LABEL:1A:"PROJECT-AAA"
1 - Success
LABEL:1A?
LABEL="PROJECT-AAA"
```

5.5 - Device Discovery Using UDP

Limited support of UDP is provided for the purpose of "device discovery." This allows a user to request the IP address and configuration of all Mini-Circuits multi-channel attenuator systems connected on the network; full control of those units is then accomplished using SSH, 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
```

6 - USB Control API for Microsoft Windows

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

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

The latest version of each DLL file can be downloaded from the Mini-Circuits website at:
https://www.minicircuits.com/softwaredownload/multiatt.html

6.1 - DLL API Options

6.1 (a) - .NET Framework 4.5 DLL (Recommended)

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

Filename: mcl_MultiChannelAtt_NET45.dll

Requirements

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

Installation

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

6.1 (b) - .NET Framework 2.0 DLL (Legacy Support)

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

Filename: mcl_MultiChannelAtt_64.dll

Requirements

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

Installation

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

6.1 (c) - ActiveX COM Object DLL (Legacy Support)

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

Filename: mcl_MultiChannelAtt.dll

Requirements

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

Installation

1) Download the latest DLL file from the Mini-Circuits website
2) Copy the .dll file to the correct directory:
   a. 32-bit PC: `C:\WINDOWS\System32`
   b. 64-bit PC: `C:\WINDOWS\SysWOW64`
3) 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 Windows versions the Command Prompt will need to be run in "Elevated" mode (as an administrator):
      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. Enter the administrator credentials if requested
4) Register the DLL using regsvr32:
   a. 32-bit PC: `Regsvr32 mcl_MultiChanAtt.dll`
   b. 64-bit PC (be sure to specify the path for regsvr32 and the DLL):
      `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mcl_MultiChanAtt.dll`
5) Hit enter to confirm, a message box will appear to advise of successful registration

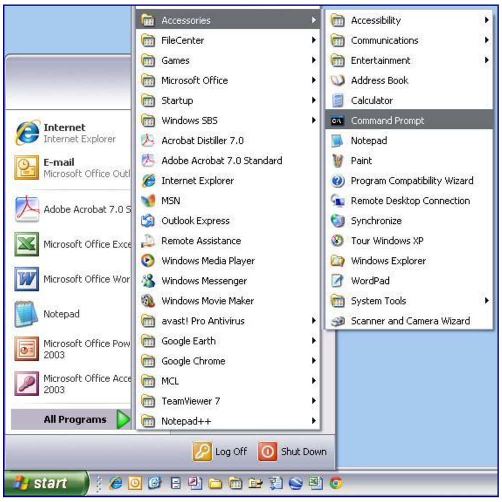


Fig 10: Opening the Command Prompt in Windows XP

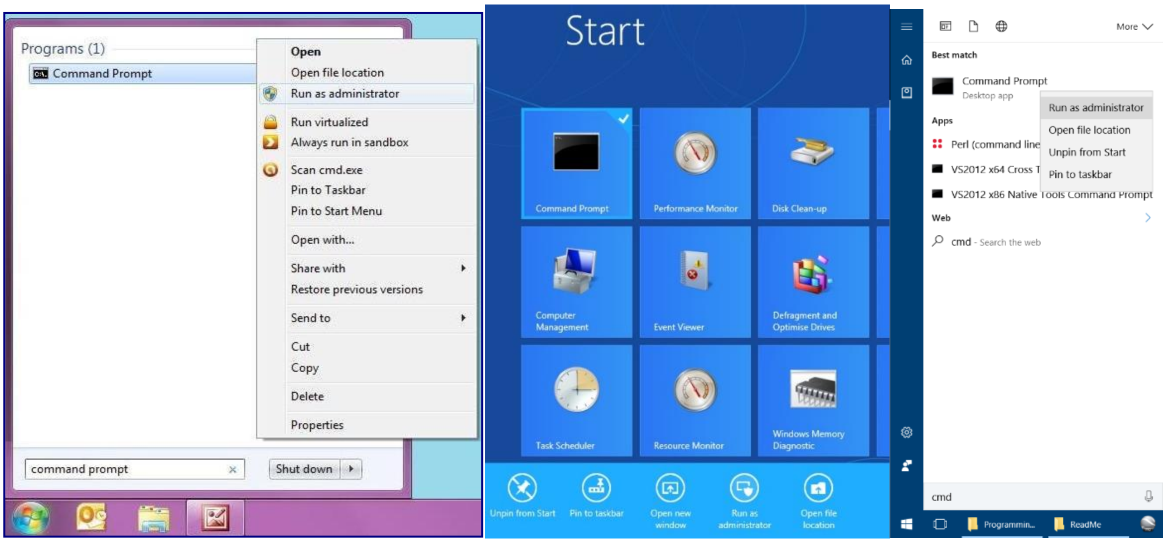
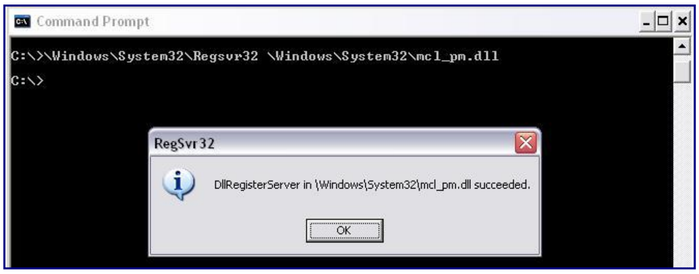


Fig 11: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)



Command Prompt

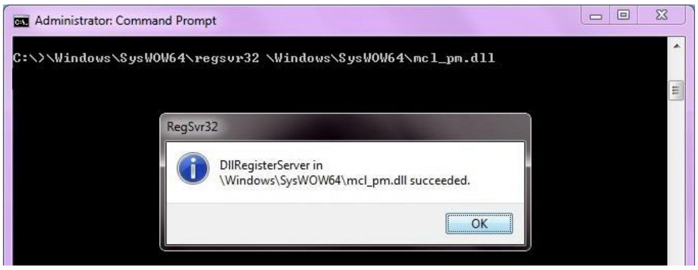
```
C:\>\Windows\System32\Regsvr32 \Windows\System32\mcl_pm.dll
C:\>
```

RegSvr32

DllRegisterServer in \Windows\System32\mcl_pm.dll succeeded.

OK

Fig 12: Registering the DLL in a 32-bit environment



Administrator: Command Prompt

```
C:\>\Windows\SysWOW64\regsvr32 \Windows\SysWOW64\mcl_pm.dll
```

RegSvr32

DllRegisterServer in \Windows\SysWOW64\mcl_pm.dll succeeded.

OK

Fig 13: Registering the DLL in a 64-bit environment

6.2 - Referencing the DLL

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

Example Declarations Using the .NET 4.5 DLL (mcl_MultiChannelAtt_NET45.dll)

(For operation with the .Net 2.0 DLL, replace "mcl_MultiChannelAtt_NET45" with "mcl_MultiChannelAtt")

Python	`import clr     # Import the pythonnet CLR library`
	`clr.AddReference('mcl_MultiChannelAtt_NET45')`
	`from mcl_MultiChannelAtt_NET45 import USB_Control`
	`MyPTE1 = USB_Control()`
	`MyPTE2 = USB_Control()`

Visual Basic	`Public MyPTE1 As New mcl_MultiChannelAtt_NET45.USB_Control`
	`Public MyPTE2 As New mcl_MultiChannelAtt_NET45.USB_Control`

Visual C++	`mcl_MultiChannelAtt_NET45::USB_Control ^MyPTE1 = gcnew`
	`mcl_MultiChannelAtt_NET45::USB_Control();`
	`mcl_MultiChannelAtt_NET45::USB_Control ^MyPTE2 = gcnew`
	`mcl_MultiChannelAtt_NET45::USB_Control();`

Visual C#	`mcl_MultiChannelAtt_NET45.USB_Control MyPTE1 = new`
	`mcl_MultiChannelAtt_NET45.USB_Control();`
	`mcl_MultiChannelAtt_NET45.USB_Control MyPTE2 = new`
	`mcl_MultiChannelAtt_NET45.USB_Control();`

MatLab	`MCL_SW = NET.addAssembly('C:\Windows\SysWOW64\mcl_MultiChannelAtt_NET45.dll')`
	`MyPTE1 = mcl_MultiChannelAtt_NET45.USB_Control`
	`MyPTE2 = mcl_MultiChannelAtt_NET45.USB_Control`

Example Declarations using the ActiveX DLL (mcl_MultiChannelAtt.dll)

Visual Basic	`Public MyPTE1 As New MCL_MultiChannelAtt.USB_Control`
	`Public MyPTE2 As New MCL_MultiChannelAtt.USB_Control`

Visual C++	`MCL_MultiChannelAtt::USB_Control ^MyPTE1 = gcnew`
	`MCL_MultiChannelAtt::USB_Control();`
	`MCL_MultiChannelAtt::USB_Control ^MyPTE2 = gcnew`
	`MCL_MultiChannelAtt::USB_Control();`

Visual C#	`public MCL_MultiChannelAtt.USB_Control MyPTE1 = new`
	`MCL_MultiChannelAtt.USB_Control();`
	`public MCL_MultiChannelAtt.USB_Control MyPTE2 = new`
	`MCL_MultiChannelAtt.USB_Control();`

MatLab	`MyPTE1 = actxserver(MCL_MultiChannelAtt.USB_Control')`
	`MyPTE2 = actxserver(MCL_MultiChannelAtt.USB_Control')`

6.3 - Note on DLL Use in Python / MatLab

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

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

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

Definition	**int Read_SN(ByRef string SN)**

Visual C#	status = MyPTE1.Read_SN(ref(SN));
	if(status > 0)
	{
	        MessageBox.Show("The connected device is " + SN);
	}

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

MatLab	[status, SN] = MyPTE1.Read_SN('')
	if status > 0
	        h = msgbox('The connected device is ', SN)
	end

6.4 - DLL Function Definitions

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

6.4 (a) - DLL Functions for USB Control

a) int Get_Available_SN_List(Ref string SN_List)
b) int Get_Available_Address_List(Ref string Add_List)
c) Short Connect (Optional Ref String SN)
d) Short ConnectByAddress (Optional Short Address)
e) Void Disconnect ()
f) Short Send_SCPI (String SndSTR , Ref String RetSTR)

6.4 (b) - DLL Functions for Ethernet Configuration

a) Short GetEthernet_CurrentConfig (Ref Int IP1, Ref Int IP2, Ref Int IP3, Ref Int IP4,
Ref Int Mask1, Ref Int Mask2, Ref Int Mask3, Ref Int Mask4,
Ref Int Gateway1, Ref Int Gateway2, Ref Int Gateway3, Ref Int Gateway4)
b) Short GetEthernet_IPAddress (Ref Int b1, Ref Int b2, Ref Int b3, Ref Int b4)
c) Short GetEthernet_MACAddress (Ref Int MAC1 , Ref Int MAC2, Ref Int MAC3, Ref Int MAC4,
Ref Int MAC5, Ref Int MAC6)
d) Short GetEthernet_NetworkGateway (Ref Int b1, Ref Int b2, Ref Int b3, Ref Int b4)
e) Short GetEthernet_SubNetMask (Ref Int b1, Ref Int b2, Ref Int b3, Ref Int b4)
f) Short GetEthernet_TCPIPPort (Ref Int port)
g) Short GetEthernet_UseDHCP ()
h) Short GetEthernet_UsePWD ()
i) Short GetEthernet_PWD (Ref 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_TCPIPPort (Int port)
n) Short SaveEthernet_UseDHCP (Int UseDHCP)
o) Short SaveEthernet_UsePWD (Int UsePwd)
p) Short SaveEthernet_PWD (String Pwd)

6.5 - DLL Function Explanations - USB Control

6.5 (a) - Get List of Connected Serial Numbers

Declaration

```
int Get_Available_SN_List(Ref String SN_List)
```

Description

Returns a list of serial numbers for all connected attenuator racks.

Parameters

Data Type	Variable	Description
String	SN_List	Variable passed by reference, to be updated with a list of all connected serial numbers, separated by a single space, for example “11301210001 11301210002 11301210003”.

Return Values

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

Examples

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get List of Available Addresses

6.5 (b) - Get List of Available Addresses

Declaration

```
int Get_Available_Address_List(Ref String Add_List)
```

Description

Returns a list of USB addresses for all connected attenuator racks.

Parameters

Data Type	Variable	Description
String	Add_List	Variable passed by reference, to be updated with a list of all connected addresses separated by a single space character, for example, “5 101 254 255”

Return Values

Data Type	Value	Description
int	0	Command failed
int	Non zero	The number of devices connected

Examples

Python

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

Visual Basic

```
If MyPTE1.Get_Available_Add_List(SN_List) > 0 Then
       MsgBox ("Connected devices: " & Address_List)
End If
```

Visual C++

```
if (MyPTE1->Get_Available_Add_List(Address_List) > 0 )
{
       MessageBox::Show("Connected devices: " + Address_List);
}
```

Visual C#

```
if (MyPTE1.Get_Available_Add_List(ref(Address_List)) > 0 )
{
       MessageBox.Show("Connected devices: " + Address_List);
}
```

MatLab

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

See Also

Get List of Connected Serial Numbers

6.5 (c) - Connect by Serial Number

Declaration

```
Short Connect(Ref 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

Data Type	Variable	Description
String	SN	Optional. The serial number of the test system. Can be omitted if only one attenuator rack is connected by USB.

Return Values

Data Type	Value	Description
Short	0	No connection was possible
	1	Connection successfully established
	2	Connection already established (Connect has been called more than once). The system will continue to operate normally.

Examples

Python	`response = MyPTE1.Connect()`
	`status = response[0]`

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

See Also

Get List of Connected Serial Numbers
Connect by Address
Disconnect

6.5 (d) - 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

Data Type	Variable	Description
Short	Address	Optional. The address of the system. Can be omitted if only one attenuator rack is connected by USB.

Return Values

Data Type	Value	Description
Short	0	No connection was possible
	1	Connection successfully established
	2	Connection already established (Connect has been called more than once)

Examples

Python	`status = MyPTE1.ConnectByAddress(5)`
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

Get List of Available Addresses
Connect by Serial Number
Disconnect

6.5 (e) - 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

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

Examples

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

See Also

Connect by Serial Number
Connect by Address

6.5 (f) - Send SCPI Command

Declaration

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

Description

Sends a SCPI command / query to set or read the attenuator states and other properties. Refer to SCPI Commands for Control of Attenuator Racks for the syntax of the SCPI commands.

Parameters

Data Type	Variable	Description
String	SndSTR	Required. The SCPI command to send.
String	RetSTR	String variable passed by reference, to be updated with the value returned from the test system.

Return Values

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

Examples

Python	`status = MyPTE1.Send_SCPI(":01:CHAN:1:SETATT:10.25", "")`
	`response = str(status[1])`

Visual Basic	`status = MyPTE1.Send_SCPI(":01:CHAN:1:SETATT:10.25", response)`
Visual C++	`status = MyPTE1->Send_SCPI(":01:CHAN:1:SETATT:10.25", response);`
Visual C#	`status = MyPTE1.Send_SCPI(":01:CHAN:1:SETATT:10.25", ref(response));`
MatLab	`[status, response] = MyPTE1.Send_SCPI(":01:CHAN:1:SETATT:10.25", '')`

See Also

SCPI Commands for Control of Attenuator Racks

6.6 - DLL Function Explanations - Ethernet Configuration

6.6 (a) - Get Ethernet Configuration

Declaration

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

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get MAC Address
Get TCP/IP Port

6.6 (b) - Get IP Address

Declaration

```
Short GetEthernet_IPAddress(Ref Int b1, Ref Int b2, Ref Int b3, Ref 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

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get Ethernet Configuration
Get TCP/IP Port
Save IP Address
Save TCP/IP Port

6.6 (c) - Get MAC Address

Declaration

```
Short GetEthernet_MACAddress(Ref Int MAC1, Ref Int MAC2, Ref Int MAC3,
                                        Ref Int MAC4, Ref Int MAC5, Ref 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

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get Ethernet Configuration

6.6 (d) - Get Network Gateway

Declaration

```
Short GetEthernet_NetworkGateway(Ref Int b1, Ref Int b2, Ref Int b3, Ref 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

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

Return Values

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

Example

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

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

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

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

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

See Also

Get Ethernet Configuration
Save Network Gateway

6.6 (e) - Get Subnet Mask

Declaration

```
Short GetEthernet_SubNetMask(Ref Int b1, Ref Int b2, Ref Int b3, Ref 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

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get Ethernet Configuration
Save Subnet Mask

6.6 (f) - Get TCP/IP Port

Declaration

```
Short GetEthernet_TCPIPPort(Ref Int 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

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Save TCP/IP Port
Get SSH Port
Get Telnet Port

6.6 (g) - Get SSH Port

Declaration

```
Short GetEthernet_SSHPort(Ref Int port)
```

Description

This function returns the port used for SSH communication. The default is port 22.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

Data Type	Variable	Description
Int	port	Integer variable which will be updated with the SSH port.

Return Values

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

Example

Python

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

Visual Basic

```
If MyPTE1.GetEthernet_SSHPort(port) > 0 Then
       MsgBox (port)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_SSHPort(port) > 0)
{
       MessageBox::Show(port);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_SSHPort(ref(port)) > 0)
{
       MessageBox.Show(port);
}
```

MatLab

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

See Also

Save SSH Port
Get TCP/IP Port
Get Telnet Port

6.6 (h) - Get Telnet Port

Declaration

```
Short GetEthernet_TelnetPort(Ref Int port)
```

Description

This function returns the port used for Telnet communication. The default is port 23.

Parameters

Data Type	Variable	Description
Int	port	Integer variable which will be updated with the Telnet port.

Return Values

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

Example

Python

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

Visual Basic

```
If MyPTE1.GetEthernet_TelnetPort(port) > 0 Then
       MsgBox (port)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_TelnetPort(port) > 0)
{
       MessageBox::Show(port);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_TelnetPort(ref(port)) > 0)
{
       MessageBox.Show(port);
}
```

MatLab

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

See Also

Save Telnet Port
Get TCP/IP Port
Get SSH Port

6.6 (i) - Get DHCP Status

Declaration

```
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

Data Type	Variable	Description
None		

Return Values

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

Example

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

See Also

Get Ethernet Configuration
Use DHCP

6.6 (j) - Get Password Status

Declaration

```
Short GetEthernet_UsePWD()
```

Description

Indicates whether or not a password is required for HTTP and Telnet communication. The password is always required for SSH communication.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

Data Type	Variable	Description
None		

Return Values

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

Example

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

See Also

Get Password
Use Password
Set Password

6.6 (k) - Get Password

Declaration

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

Description

Returns the password for Ethernet communication. The password is always required for SSH communication but only for HTTP and Telnet when password security is enabled.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

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

Return Values

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

Example

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get Password Status
Use Password
Set Password

6.6 (l) - 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

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

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)`
Visual Basic	`status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)`
Visual C++	`status = MyPTE1->SaveEthernet_IPAddress(192, 168, 1, 0);`
Visual C#	`status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0);`
MatLab	`status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0);`

See Also

Get Ethernet Configuration
Get IP Address

6.6 (m) - 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

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

Return Values

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

Example

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

See Also

Get Ethernet Configuration
Get Network Gateway

6.6 (n) - 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

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

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)`
Visual Basic	`status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)`
Visual C++	`status = MyPTE1->SaveEthernet_SubnetMask(255, 255, 255, 0);`
Visual C#	`status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0);`
MatLab	`status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0);`

See Also

Get Ethernet Configuration
Get Subnet Mask

6.6 (o) - 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

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

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_TCPIPPort(70)`
Visual Basic	`status = MyPTE1.SaveEthernet_TCPIPPort(70)`
Visual C++	`status = MyPTE1->SaveEthernet_TCPIPPort(70);`
Visual C#	`status = MyPTE1.SaveEthernet_TCPIPPort(70);`
MatLab	`status = MyPTE1.SaveEthernet_TCPIPPort(70);`

See Also

Get TCP/IP Port
Save TCP/IP Port
Save SSH Port
Save Telnet Port

6.6 (p) - Save SSH Port

Declaration

```
Short SaveEthernet_SSHPort(Int port)
```

Description

This function sets the port to be used for SSH communication. The default is port 22.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

Data Type	Variable	Description
Int	port	Numeric value of the SSH port.

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_SSHPort(22)`
Visual Basic	`status = MyPTE1.SaveEthernet_SSHPort(22)`
Visual C++	`status = MyPTE1->SaveEthernet_SSHPort(22);`
Visual C#	`status = MyPTE1.SaveEthernet_SSHPort(22);`
MatLab	`status = MyPTE1.SaveEthernet_SSHPort(22);`

See Also

Get SSH Port
Save TCP/IP Port
Save Telnet Port

6.6 (q) - Save Telnet Port

Declaration

```
Short SaveEthernet_TelnetPort(Int port)
```

Description

This function sets the port to be used for Telnet communication. The default is port 23.

Parameters

Data Type	Variable	Description
Int	port	Numeric value of the Telnet port.

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_TelnetPort(22)`
Visual Basic	`status = MyPTE1.SaveEthernet_TelnetPort(22)`
Visual C++	`status = MyPTE1->SaveEthernet_TelnetPort(22);`
Visual C#	`status = MyPTE1.SaveEthernet_TelnetPort(22);`
MatLab	`status = MyPTE1.SaveEthernet_TelnetPort(22);`

See Also

Get Telnet Port
Save TCP/IP Port
Save SSH Port
Save Telnet Port

6.6 (r) - 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

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

Return Values

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

Example

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

See Also

Get DHCP Status

6.6 (s) - Use Password

Declaration

```
Short SaveEthernet_UsePWD(Int UsePwd)
```

Description

Sets whether or not a password is required for HTTP and Telnet communication. The password is always required for SSH communication.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

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

Return Values

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

Example

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

See Also

Get Password Status
Get Password
Set Password

6.6 (t) - Set Password

Declaration

```
Short SaveEthernet_PWD(String Pwd)
```

Description

Sets the password for Ethernet communication. The password is always required for SSH communication but only for HTTP and Telnet when password security is enabled.

Note: SSH communication is not supported as standard on all models. Please contact testsolutions@minicircuits.com for details.

Parameters

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

Return Values

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

Example

Python	`status = MyPTE1.SaveEthernet_PWD("123")`
Visual Basic	`status = MyPTE1.SaveEthernet_PWD("123")`
Visual C++	`status = MyPTE1->SaveEthernet_PWD("123");`
Visual C#	`status = MyPTE1.SaveEthernet_PWD("123");`
MatLab	`status = MyPTE1.SaveEthernet_PWD("123");`

See Also

Get Password Status
Get Password
Use Password

7 - USB Control via Direct Programming (Linux)

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

7.1 - USB Interrupt Code Concept

To open a USB connection to the system, the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- Product ID: 0x22

The transmitted and received buffer sizes for the interrupt commands 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.

Worked examples can be found in the Programming Examples & Troubleshooting Guide, available from the Mini-Circuits website. The examples make use of standard USB and HID (Human Interface Device) APIs to interface with the system.

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 switch system.

Transmit Array

Byte	Data	Description
0	1 or 42	Interrupt code for Send SCPI Command
1 - 63	SCPI Transmit String	The SCPI command to send represented as a series of ASCII character codes, one character code per byte

Returned Array

Byte	Data	Description
0	1 or 42	Interrupt code for Send SCPI Command
1 to (n-1)	SCPI Return String	The SCPI return string, one character per byte, represented as ASCII character codes
n	0	Zero value byte to indicate the end of the SCPI return string
(n+1) to 63	Not significant	“Don’t care” bytes, can be any value

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:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	49	ASCII character code for :
2	48	ASCII character code for 0
3	48	ASCII character code for 0
4	49	ASCII character code for :
5	77	ASCII character code for M
3	78	ASCII character code for N
7	63	ASCII character code for ?

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

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

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.

```
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.

```
Imports System.IO
Imports System.Net

Private Function Send_HTTP_Get(Command_To_Send 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                       ' Use the standard HTTPWebRequest/Response
    Dim urlResponse As HttpWebResponse

    ' Form the URL to send (http + IP address + command to send)

    urlToSend = WebRequest.Create("http://" & ZT_IP_Address & "/:" & Command_To_Send)
    urlToSend.Method = "GET"
    urlResponse = urlToSend.GetResponse()                 ' Send Request
    webStream = urlResponse.GetResponseStream()           ' Get Response

    Dim webStreamReader As New StreamReader(webStream)

    While webStreamReader.Peek >= 0                       ' Read Response in one variable
        HTTP_Response = webStreamReader.ReadToEnd()
    End While

    Send_HTTP_Get = Trim(HTTP_Response)                   ' Return the 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.

```
########################################
# 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()

    # Catch an exception if URL is incorrect (incorrect IP or disconnected)
    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) - 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.

```
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"+a_block+":CHAN:"+a_channel+":SETATT:"+value, att_return);


      # Query and print the attenuator value
      att = ztm.Send_SCPI("0"+a_block+":CHAN:"+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.

```
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 attenuator 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.

```
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->Connect();                      # Connect to the DLL

# 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)
      $ztm->Send_SCPI(":0$a_block:CHAN:$a_channel:SETATT:$value", $return_value);

      # Query and print the attenuator value
      $ztm->Send_SCPI(":0$a_block:CHAN:$a_channel:ATT?", $return_value);
      print "Block $a_block channel $a_channel set to $return_value\n";

   }
}

$ztm->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.

```
#!/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).

```
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("-------------------------------------------");
                Console.WriteLine("ZTM.exe -s SN command :Send SCPI command to S/N");
                Console.WriteLine("ZTM.exe -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)