Test Solutions - Programming Manual

ZTVX Series - 2 x *n* Switch Matrices

Important Notice

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

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

Trademarks

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

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

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

Mini-Circuits
13 Neptune Avenue
Brooklyn, NY 11235, USA
Phone: +1-718-934-4500
Email: sales@minicircuits.com
Web: www.minicircuits.com

1 - Overview

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' ZTVX Series 2 x *n* switch matrix family. For instructions on using the supplied GUI program, or connecting the PTE hardware, please refer to the User Guide.

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

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

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

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

The software package includes a GUI program, ActiveX and .NET DLL files, Linux support, project examples for third party software, and detailed user manuals.

2 - Programming with Mini-Circuits' ZTVX Series Switch Matrices

Communication with the ZTVX Series can be accomplished in a number of ways:

1. Using the provided ActiveX or .Net API objects (DLL files) on a Windows operating system (see USB Control in a Windows Environment)
2. Using HTTP or Telnet communication over an Ethernet connection (see Ethernet Control over IP Networks), this is largely operating system independent

In all cases the full functionality of the ZTVX Series is accessible using a series of ASCII text commands and queries, as detailed in the following section.

2.1 - Summary of Commands / Queries

	Description	Command/Query
a	Get Model Name	:MN?
b	Get Serial Number	:SN?
c	Set Switch Path	:PATH:[a_port]:[n_port]
d	Get Switch Path	:PATH:[start_port]?
e	Get Firmware	:FIRMWARE?
f	Get Switch Counter	:[type]:[number]:SCOUNTER?
g	Save Switch Counters	:OPERATIONDATA:SAVE

2.2 - Summary of Advanced Operation Commands / Queries

	Description	Command/Query
a	Reset all Switches	:CLEARALL
b	Reset Switches for Single Path	:CLEARPATH[a_port]
c	Turn off LEDs	:CLEAR[colour]LEDS
d	Set Switch	:[type]:[number]:STATE:[port]
e	Get Switch State	:[type]:[number]:STATE?

2.3 - Description of Commands/Queries

2.3 (a) - Get Model Name

Description

Returns the Mini-Circuits model name for the switch matrix

Command Syntax

```
:MN?
```

Return String

```
MN=[model]
```

Variable	Description
[model]	The model name

Examples

String to Send	String Returned
:MN?	MN=ZTVX-10-12

DLL Implementation: `Send_Command(":MN?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:MN?`

See Also

Get Serial Number

2.3 (b) - Get Serial Number

Description

Returns the serial number of the switch matrix

Command Syntax

```
:SN?
```

Return String

```
SN=[serial_no]
```

Variable	Description
[serial_no]	The serial number

Examples

String to Send	String Returned
:SN?	SN=11608180025

DLL Implementation: `Send_Command(":SN?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:SN?`

See Also

Get Model Name

2.3 (c) - Set Switch Path

Description

Takes all necessary actions to set the switch path so that the requested "A port" is connected to the requested "N port":

1. Sets the internal switch states as necessary to connect the A and N ports
2. Sets the front panel LEDs to indicate the active paths
3. Deenergizes any switches not required in the 2 active switch paths

Command

```
:PATH:[a_port]:[n_port]
```

Variable	Description
[a_port]	The "A port" of the switch matrix to be connected to the "N port"
[n_port]	The "N port" of the switch matrix to be connected to the "A port"

Return Value

```
[status]
```

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

Examples

String to Send	String Returned
:PATH:A1:N7	1 - Success
:PATH:A2:N5	1 - Success

DLL Implementation: `Send_Command(":PATH:A1:N7", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PATH:A1:N7`

See Also

Get Switch Path

2.3 (d) - Get Switch Path

Description

Indicates the active switch path by returning the opposite port to which a specified port is connected

Command

```
:PATH:[start_port]?
```

Variable	Description
[start_port]	Any valid port name, eg: A1, A2, N1, N2, N3…

Return Value

```
[end_port]
```

Variable	Description
[end_port]	The opposite port to which the queried start port is connected. 0 will be returned for an N port which is not connected to either active path.

Examples

String to Send	String Returned
:PATH:A1?	N7
:PATH:N5?	A2
:PATH:N6?	0

DLL Implementation: `Send_Command(":PATH:A1?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PATH:A1?`

See Also

Set Switch Path

2.3 (e) - Get Firmware

Description

Returns the version number of the internal firmware.

Command Syntax

```
:FIRMWARE?
```

Return String

```
[firmware]
```

Variable	Description
[firmware]	The internal firmware version number

Examples

String to Send	String Returned
:FIRMWARE?	FIRMWARE=A3

DLL Implementation: `Send_Command(":FIRMWARE?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:FIRMWARE?`

2.3 (f) - Get Switch Counter

Description

Returns a counter value indicating the number of switching cycles undertaken in the lifetime of a specific switch.

Note: See Save Switch Counters for correct operation.

Command Syntax

```
:[type]:[number]:SCOUNTER?
```

Variable	Value	Description
[type]	SPDT	Query an SPDT switch counter
	SP4T	Query an SP4T switch counter
	SP6T	Query an SP6T switch counter
	MTS	Query a transfer switch counter
[number]		The number of the switch to query

Return String (SPDT and MTS)

```
[count]
```

Variable	Description
[count]	The number of switch cycles undertaken in the lifetime of the specified switch

Return String (SP4T and SP6T)

```
[count1];[count2];[count3];[count4]
```

Variable	Description
[count1]	The number of connections to port 1 undertaken in the lifetime of the specified switch
[count2]	The number of connections to port 2 undertaken in the lifetime of the specified switch
[count3]	The number of connections to port 3 undertaken in the lifetime of the specified switch
[count4]	The number of connections to port 4 undertaken in the lifetime of the specified switch
[count5]	SP6T only. The number of connections to port 5 undertaken in the lifetime of the specified switch
[count6]	SP6T only. The number of connections to port 6 undertaken in the lifetime of the specified switch

Examples

String to Send	String Returned
:SPDT:1:SCOUNTER?	9540
:SP4T:1:SCOUNTER?	2000;1253;1500;1685
:SP6T:1:SCOUNTER?	195;452;300;125;850;647
:MTS:1:SCOUNTER?	9540

DLL Implementation:

```
Send_SCPI(":SPDT:1:SCOUNTER?", RetStr)
Send_SCPI(":SP4T:1:SCOUNTER?", RetStr)
Send_SCPI(":SP6T:1:SCOUNTER?", RetStr)
Send_SCPI(":MTS:1:SCOUNTER?", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:SPDT:1:SCOUNTER?
http://10.10.10.10/:SP4T:1:SCOUNTER?
http://10.10.10.10/:SP6T:1:SCOUNTER?
http://10.10.10.10/:MTS:1:SCOUNTER?
```

See Also

Save Switch Counters

2.3 (g) - Save Switch Counters

Description

Transfers the latest switch counters from temporary to permanent memory. This command should be sent following completion of all switch sequences and prior to powering off the system in order to preserve the latest data. During normal operation, this data is internally stored in volatile memory but automatically updated into permanent memory every 3 minutes.

Command Syntax

```
:OPERATIONDATA:SAVE
```

Return String

```
[status]
```

Variable	Value	Description
[status]	0 - Failed	Command failed
	1 - Success	Command completed successfully
	2 - Fail	Command already sent within previous 3 minutes (wait and try again)

Examples

String to Send	String Returned
:OPERATIONDATA:SAVE	1 - Success

DLL Implementation: `Send_SCPI(":OPERATIONDATA:SAVE", RetStr)`
HTTP Implementation: `http://10.10.10.10/:OPERATIONDATA:SAVE`

See Also

Get Switch Counter

2.4 - Description of Advanced Operation Commands / Queries

2.4 (a) - Reset all Switches

Description

Resets all switches to their default state:

- SPDT switches: Com to port 1
- SP4T/SP6T switches: All ports disconnected
- MTS switches: J1 to J3; J2 to J4

Command Syntax

```
:CLEARALL
```

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:CLEARALL	1

DLL Implementation: `Send_Command(":CLEARALL", RetStr)`
HTTP Implementation: `http://10.10.10.10/:CLEARALL`

See Also

Set Switch Path
Reset Switches for Single Path
Turn off LEDs

2.4 (b) - Reset Switches for Single Path

Description

Resets all switches on a single active paths to their default state:

- SPDT switches: Com to port 1
- SP4T/SP6T switches: All ports disconnected
- MTS switches: J1 to J3; J2 to J4

Command Syntax

```
:CLEARPATH:[a_port]
```

Variable	Value	Description
[a_port]	A1	Reset all switches currently set for the active path from port A1
	A2	Reset all switches currently set for the active path from port A2

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:CLEARPATH:A1	1
:CLEARPATH:A2	1

DLL Implementation: `Send_Command(":CLEARPATH:A1", RetStr)`
HTTP Implementation: `http://10.10.10.10/:CLEARPATH:A2`

See Also

Set Switch Path
Reset all Switches
Turn off LEDs

2.4 (c) - Turn off LEDs

Description

Turns off all LEDs of a particular colour.

Command Syntax

```
:CLEAR[colour]LEDS
```

Variable	Value	Description
[colour]	ORANGE	Turn off all orange LEDs (except the port A2 LED)
	GREEN	Turn off all green LEDs (except the port A1 LED)

Return String

```
[status]
```

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

Examples

String to Send	String Returned
:CLEARORANGELEDS	1
:CLEARGREENLEDS	1

DLL Implementation: `Send_Command(":CLEARORANGELEDS", RetStr)`
HTTP Implementation: `http://10.10.10.10/:CLEARORANGELEDS`

See Also

Set Switch Path
Reset all Switches
Reset Switches for Single Path

2.4 (d) - Set Switch

Description

Sets an individual switch within the matrix to a specific state (consult block diagram for switch positions).

Command

```
:[type]:[number]:STATE:[port]
```

Variable	Value	Description
[type]	SPDT	Set an SPDT switch state
	SP4T	Set an SP4T switch state
	SP6T	Set an SP6T switch state
	MTS	Set a transfer switch state
[number]		The number of the switch to set
[port]		The port to which the Com port should be connected: • SPDT: 1 to 2 • SP4T: 0 to 4 (0 = all ports disconnected) • SP6T: 0 to 6 (0 = all ports disconnected) • MTS: 1 (J1 to J3; J2 to J4) or 2 (J1 to J2; J3 to J4)

Return Value

```
[status]
```

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

Examples

String to Send	String Returned
:SPDT:STATE:2	1 - Success
:SP4T:STATE:4	1 - Success
:SP6T:STATE:6	1 - Success
:MTS:STATE:2	1 - Success

DLL Implementation:	`Send_Command(":SPDT:STATE:2", RetStr)`
HTTP Implementation:	`http://10.10.10.10/:SPDT:STATE:2`

See Also

Set Switch Path
Get Switch Path
Get Switch State

2.4 (e) - Get Switch State

Description

Gets the state of an individual switch within the matrix (consult block diagram for switch positions).

Command

```
:[type]:[number]:STATE?
```

Variable	Value	Description
[type]	SPDT	Get an SPDT switch state
	SP4T	Get an SP4T switch state
	SP6T	Get an SP6T switch state
	MTS	Get a transfer switch state
[number]		The number of the switch to query

Return Value

```
[port]
```

Variable	Description
[port]	The port to which the Com port is connected: • SPDT: 1 to 2 • SP4T: 0 to 4 (0 = all ports disconnected) • SP6T: 0 to 6 (0 = all ports disconnected) • MTS: 1 (J1 to J3; J2 to J4) or 2 (J1 to J2; J3 to J4)

Examples

String to Send	String Returned
:SPDT:STATE?	2
:SP4T:STATE?	4
:SP6T:STATE?	6
:MTS:STATE?	2

DLL Implementation: `Send_Command(":SPDT:STATE?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:SPDT:STATE?`

See Also

Set Switch Path
Get Switch Path
Set Switch

3 - Ethernet Control over IP Networks

Control of Mini-Circuits' ZTVX Series over an Ethernet network (using the RJ45 connection) is accomplished using HTTP (Get/Post commands) or Telnet communication. These both provide a means to send the ASCII commands queries detailed above (see Summary of ASCII Commands / Queries).

UDP transmission is also supported for discovering available switch matrix devices on the network.

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

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

Notes:

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

3.1 - Configuring Ethernet Settings via USB

The switch matrix must be connected via the USB interface in order to configure the Ethernet settings. Following initial configuration, the device can be controlled via the Ethernet interface with no further need for a USB connection. The API DLL provides the below functions for configuring the Ethernet settings, please see DLL Functions for Ethernet Configuration for full details).

3.2 - Ethernet Communication Methodology

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

3.2 (a) - Sending Commands Using HTTP

The basic format of the HTTP command to set the switch matrix is:

http://ADDRESS:PORT/PWD;COMMAND

Where

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

Example 1:

http://192.168.100.100:800/PWD=123;:PATH:A1:N5

Explanation:

- The switch has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to "123"
- The command is to set the switch path so ports A1 and N5 are connected (see Summary of Commands / Queries for the full explanation of all commands/queries)

Example 2:

http://10.10.10.10/:PATH:A1:N5

Explanation:

- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to set the switch path so ports A1 and N5 are connected (see Summary of Commands / Queries for the full explanation of all commands/queries)

3.2 (b) - Sending Queries Using HTTP

The basic format of the HTTP command to query the switch matrix is:

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

Where

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

Example 1:

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

Explanation:

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

Example 2:

http://10.10.10.10/:PATH:A1?

Explanation:

- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The query is to check which path is connected from port A1 (see Summary of Commands / Queries for the full explanation of all commands/queries)

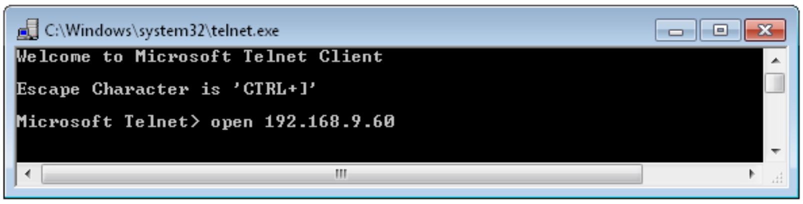
The device will return the result of the query as a string of ASCII characters.

3.2 (c) - Communication Using Telnet

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

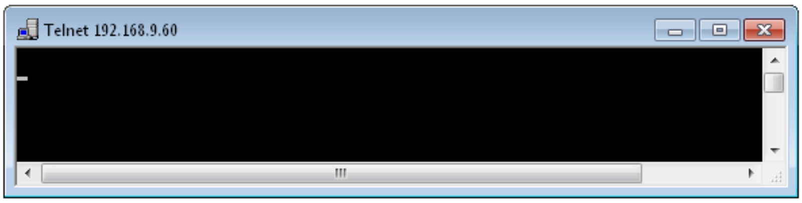
The full list of all commands and queries is detailed in Summary of Commands / Queries. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below:

1) Set up Telnet connection to a switch matrix with IP address 192.168.9.60:

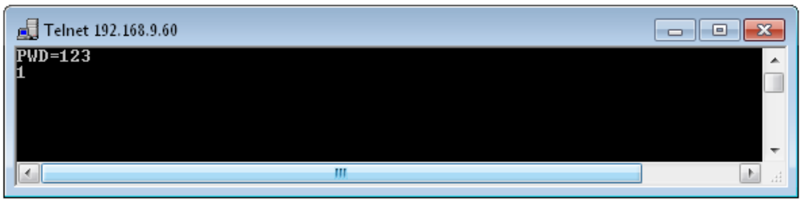


```
Welcome to Microsoft Telnet Client

Escape Character is 'CTRL+]'

Microsoft Telnet> open 192.168.9.60
```

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

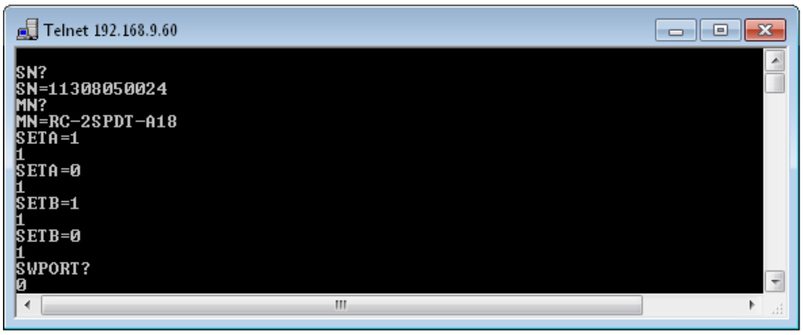


3) The password (if enabled) must be sent as the first command; a return value of 1 indicates success:



```
PWD=123
1
```

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



```
SN?
SN=11308050024
MN?
MN=RC-2SPDT-A18
SETA=1
1
SETA=0
1
SETB=1
1
SETB=0
1
SWPORT?
0
```

3.3 - Device Discovery Using UDP

In addition to HTTP and Telnet, the ZTVX switch matrix series also provides limited support of the UDP protocol for the purpose of "device discovery." This allows a user to request the IP address and configuration of all Mini-Circuits switch matrices connected on the network; full control of those units is then accomplished using HTTP or Telnet, as detailed previously.

Alternatively, the IP configuration can be identified or changed by connecting the switch matrix with the USB interface (see Configuring Ethernet Settings via USB).

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

UDP Ports

Mini-Circuits' ZTVX switch matrices 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 IP address is already known it is not necessary to use UDP.

Transmission

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

Receipt

All Mini-Circuits RC switch matrices 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:

```
MCLRFSWITCH?
```

Received Data:

```
Model Name: ZTVX-10-12-S
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: ZTVX-12-75-N
Serial Number: 11302120002
IP Address=192.168.9.102 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-02

Model Name: ZTVX-10-18-S
Serial Number: 11302120003
IP Address=192.168.9.103 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-03
```

4 - USB Control in a Windows Environment

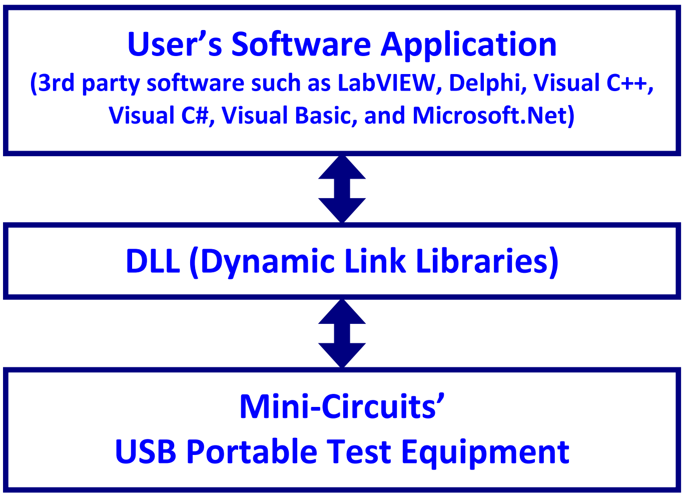
Control of Mini-Circuits' ZTVX Series using the USB connection on a Windows operating system is accomplished using the included API DLL files. These provide a means to send the ASCII commands queries detailed above (see Summary of Commands / Queries) and also expose some additional functionality.

4.1 - The DLL (Dynamic Link Library) Concept

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

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

Mini-Circuits' CD package provides DLL Objects designed to allow your own software application to interface with the functions of the Mini-Circuits ZTVX series switch matrix.



User’s Software Application
(3rd party software such as LabVIEW, Delphi, Visual C++, Visual C#, Visual Basic, and Microsoft.Net)

DLL (Dynamic Link Libraries)

Mini-Circuits’
USB Portable Test Equipment

Fig 4.1-a: DLL Interface Concept

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

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

2. **Microsoft.NET Class Library**
   A logical unit of functionality that runs under the control of the Microsoft.NET system.

4.1 (a) - ActiveX COM Object

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

Supported Programming Environments

Mini-Circuits' ZTVX Series custom switch matrices have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support ActiveX functionality. Please contact Mini-Circuits for support.

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

Installation

1. Copy the DLL file to the correct directory:
   For 32-bit Windows operating systems this is C:\WINDOWS\System32
   For 64-bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. Open the Command Prompt:
   a. For Windows XP® (see *Fig 4.1-b*):
      i. Select "All Programs" and then "Accessories" from the Start Menu
      ii. Click on "Command Prompt" to open
   b. For later versions of the Windows operating system you will need to have Administrator privileges in order to run the Command Prompt in "Elevated" mode (see *Fig 4.1-c* for Windows 7 and Windows 8):
      i. Open the Start Menu/Start Screen and type "Command Prompt"
      ii. Right-click on the shortcut for the Command Prompt
      iii. Select "Run as Administrator"
      iv. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3. Use regsvr32 to register the DLL:
   For 32-bit Windows operating systems type (see Fig 4.1-d):
   ```
   \WINDOWS\System32\Regsvr32 \WINDOWS\System32\mcl_ztvx.dll
   ```
   For 64-bit Windows operating systems type (see Fig 4.1-e):
   ```
   \WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mcl_ztvx.dll
   ```
4. Hit enter to confirm and a message box will appear to advise of successful registration.

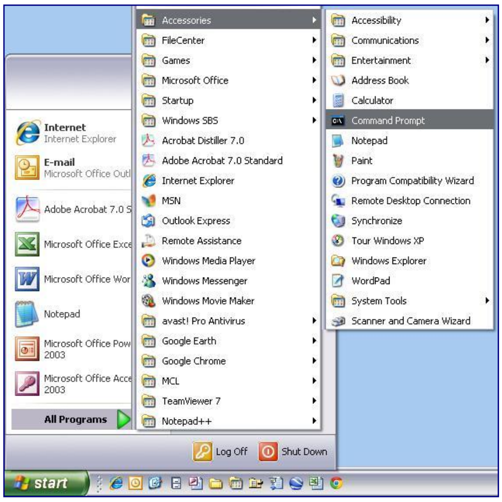


Fig 4.1-b: Opening the Command Prompt in Windows XP

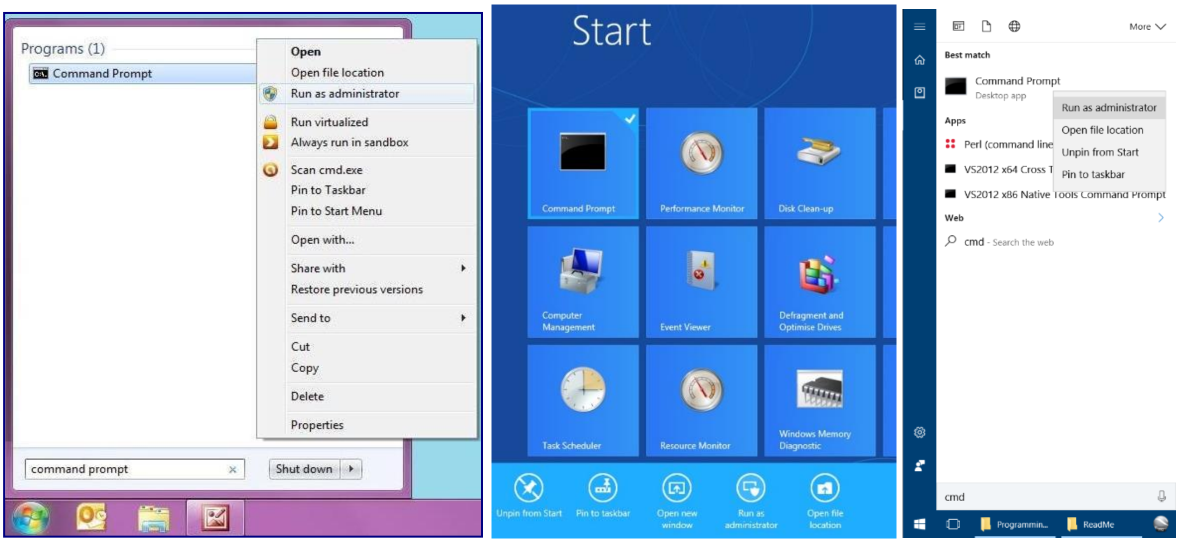
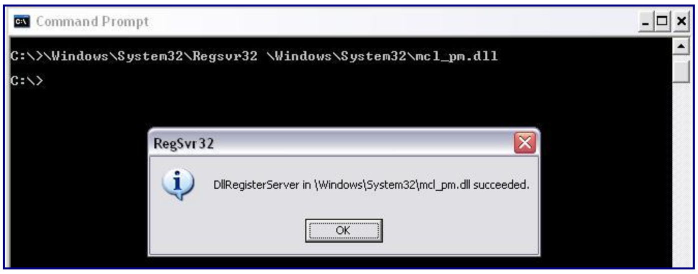


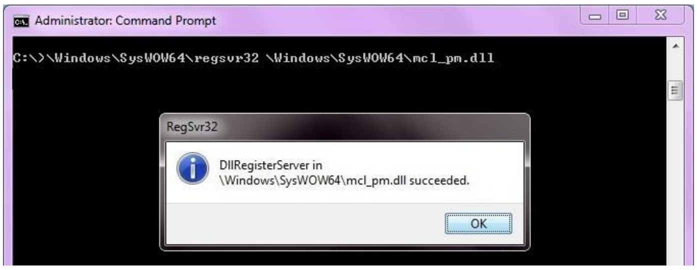
Fig 4.1-c: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)



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

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

Fig 4.1-d: Registering the DLL in a 32-bit environment



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

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

Fig 4.1-e: Registering the DLL in a 64-bit environment

4.1 (b) - Microsoft.NET Class Library

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

Supported Programming Environments

Mini-Circuits' ZTVX Series custom switch matrices has been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support Microsoft.NET functionality. Please contact Mini-Circuits for support.

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

Installation

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

4.2 - Referencing the DLL Library

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

Example Declarations using the ActiveX DLL

Visual Basic

```
Public MyPTE1 As New MCL_ZTVX.USB_Control
        ' Instantiate new switch object, assign to MyPTE1
Public MyPTE2 As New MCL_ZTVX.USB_Control
        ' Instantiate new switch object, assign to MyPTE2
```

Visual C++

```
MCL_ZTVX::USB_Control ^MyPTE1 = gcnew MCL_ZTVX::USB_Control();
        // Instantiate new switch object, assign to MyPTE1
MCL_ZTVX::USB_Control ^MyPTE2 = gcnew MCL_ZTVX::USB_Control();
        // Instantiate new switch object, assign to MyPTE2
```

Visual C#

```
public MCL_ZTVX.USB_Control MyPTE1 = new MCL_ZTVX.USB_Control();
        // Instantiate new switch object, assign to MyPTE1
public MCL_ZTVX.USB_Control MyPTE2 = new MCL_ZTVX.USB_Control();
        // Instantiate new switch object, assign to MyPTE2
```

Matlab

```
MyPTE1 = actxserver('MCL_ZTVX.USB_Control')
        % Instantiate new switch object, MyPTE1
MyPTE2 = actxserver('MCL_ZTVX.USB_Control')
        % Instantiate new switch object, MyPTE2
```

Example Declarations using the .NET DLL

Visual Basic

```
Public MyPTE1 As New MCL_ZTVX_64.USB_Control
        ' Instantiate new switch object, assign to MyPTE1
Public MyPTE2 As New MCL_ZTVX_64.USB_Control
        ' Instantiate new switch object, assign to MyPTE2
```

Visual C++

```
MCL_ZTVX_64::USB_Control^MyPTE1 = gcnew MCL_ZTVX_64::USB_Control();
        // Instantiate new switch object, assign to MyPTE1
MCL_ZTVX_64::USB_Control^MyPTE2 = gcnew MCL_ZTVX_64::USB_Control();
        // Instantiate new switch object, assign to MyPTE2
```

Visual C#

```
public MCL_ZTVX_64.USB_ControlMyPTE1 = new MCL_ZTVX_64.USB_Control();
        // Instantiate new switch object, assign to MyPTE1
public MCL_ZTVX_64.USB_ControlMyPTE2 = new MCL_ZTVX_64.USB_Control();
        // Instantiate new switch object, assign to MyPTE2
```

Matlab

```
ZTVX_USB = NET.addAssembly('C:\Windows\SysWOW64\MCL_ZTVX_64.dll')
MyPTE1 = MCL_ZTVX_64.USB_Control      % Invoke new switch object, MyPTE1
MyPTE2 = MCL_ZTVX_64.USB_Control      % Invoke new switch object, MyPTE2
```

4.3 - Summary of DLL Functions

After referencing the DLL class (as detailed above), the first step in communicating with the switch matrix is to call the Connect function to establish a connection with a specific box. Following this, and sequence of DLL functions can be used, with the final step being the Disconnect function when the program is complete. For most applications the Send_Command function is the only other function needed (in addition to Connect and Disconnect) as this allows the user to send the ASCII text commands detailed above (see Summary of ASCII Commands / Queries).

4.3 (a) - DLL Functions for USB Control

```
a) short Connect(Optional string SN)
b) short ConnectByAddress(Optional short Address)
c) void Disconnect()
d) short Read_ModelName(string ModelName)
e) short Read_SN(string SN)
f) short Set_Address(short Address)
g) short Get_Address()
h) short Get_Available_SN_List(ByRef string SN_List)
i) short Get_Available_Address_List(ByRef string Add_List)
j) short Send_SCPI(string CommandRequest, ByRef string RetStr)
k) short GetUSBConnectionStatus()
l) short GetExtFirmware(ByRef short A0, ByRef short A1,
                                ByRef short A2, ByRef string Firmware)
```

4.3 (b) - DLL Functions for Ethernet Configuration

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

4.4 - DLL Functions for USB Control

These functions provide a means to control the system over a USB connection.

4.4 (a) - Connect

Declaration

```
short Connect(Optional string SN)
```

Description

Initializes the USB connection to a switch matrix. If multiple switch matrices are connected to the same computer, then the serial number should be included, otherwise this can be omitted.

Parameters

Data Type	Variable	Description
string	SN	Optional. The serial number of the switch matrix. Can be omitted if only one switch matrix is connected.

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 switch will continue to operate normally.

Examples

Visual Basic

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

Visual C++

```
        status = MyPTE1->Connect(SN);
```

Visual C#

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

Matlab

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

See Also

Connect by Address
Get List of Connected Serial Numbers
Disconnect

4.4 (b) - Connect by Address

Declaration

```
short ConnectByAddress(Optional short Address)
```

Description

This function is called to initialize the USB connection to a switch matrix by referring to a user defined 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 connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the switch is no longer needed. The switch should be disconnected on completion of the program using the Disconnect function.

Parameters

Data Type	Variable	Description
short	Address	Optional. The address of the USB switch matrix. Can be omitted if only one switch matrix is connected.

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

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Connect
Get List of Available Addresses
Disconnect

4.4 (c) - Disconnect

Declaration

```
void Disconnect()
```

Description

This function is called to close the connection to the switch matrix after completion of the switching routine. 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 switch matrix from the computer, then reconnect the switch matrix before attempting to start again.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

Examples

Visual Basic

```
MyPTE1.Disconnect()
```

Visual C++

```
MyPTE1->Disconnect();
```

Visual C#

```
MyPTE1.Disconnect();
```

Matlab

```
MyPTE1.Disconnect
```

See Also

Connect
Connect by Address
Get List of Connected Serial Numbers
Get List of Available Addresses

4.4 (d) - Read Model Name

Declaration

```
short Read_ModelName(string ModelName)
```

Description

This function is called to determine the Mini-Circuits part number of the connected switch matrix. The user passes a string variable which is updated with the part number.

Parameters

Data Type	Variable	Description
string	ModelName	Required. A string variable that will be updated with the Mini-Circuits part number for the switch matrix.

Return Values

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

Examples

Visual Basic

```
        If MyPTE1.Read_ModelName(ModelName) > 0 Then
                MsgBox ("The connected switch matrix is " & ModelName)
                        ' Display a message stating the model name
        End If
```

Visual C++

```
        if (MyPTE1->Read_ModelName(ModelName) > 0 )
        {
                MessageBox::Show("The connected switch matrix is " + ModelName);
                        // Display a message stating the model name
        }
```

Visual C#

```
        if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
        {
                MessageBox.Show("The connected switch matrix is " + ModelName);
                        // Display a message stating the model name
        }
```

Matlab

```
        [status, ModelName]=MyPTE1.Read_ModelName(ModelName)
        if status > 0
                h = msgbox('The connected switch matrix is ', ModelName)
                        % Display a message stating the model name
        end
```

See Also

Read Serial Number

4.4 (e) - Read Serial Number

Declaration

```
short Read_SN(string SN)
```

Description

This function is called to determine the serial number of the connected switch matrix. The user passes a string variable which is updated with the serial number.

Parameters

Data Type	Variable	Description
string	ModelName	Required. string variable that will be updated with the Mini-Circuits serial number for the switch matrix.

Return Values

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

Examples

Visual Basic

```
If MyPTE1.Read_SN(SN) > 0 Then
        MsgBox (“The connected switch matrix is “ & SN)
                ' Display a message stating the serial number
End If
```

Visual C++

```
if (MyPTE1->Read_SN(SN) > 0 )
{
        MessageBox::Show("The connected switch matrix is " + SN);
                // Display a message stating the serial number
}
```

Visual C#

```
if (MyPTE1.Read_SN(ref(SN)) > 0 )
{
        MessageBox.Show("The connected switch matrix is " + SN);
                // Display a message stating the serial number
}
```

Matlab

```
[status, SN]= MyPTE1.Read_SN(SN)
if status > 0
        h = msgbox('The connected switch matrix is ', SN)
                % Display a message stating the serial number
end
```

See Also

Read Model Name

4.4 (f) - Set Address

Declaration

```
short Set_Address(short Address)
```

Description

This function allows the internal address of the connected switch matrix to be changed from the factory default of 255. The switch matrix can be referred to by the address instead of the serial number (see Connect by Address).

Parameters

Data Type	Variable	Description
short	Address	Required. An integer value from 1 to 255

Return Values

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

Example

Visual Basic

```
        status = MyPTE1.Set_Address(1)
```

Visual C++

```
        status = MyPTE1->Set_Address(1);
```

Visual C#

```
        status = MyPTE1.Set_Address(1);
```

Matlab

```
        status = MyPTE1.Set_Address(1)
```

See Also

Get Address
Connect by Address
Get List of Available Addresses

4.4 (g) - Get Address

Declaration

```
short Get_Address()
```

Description

This function returns the address of the connected switch matrix.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	Command failed
short	1-255	Address of the switch matrix

Examples

Visual Basic

```
        addr = MyPTE1.Get_Address()
```

Visual C++

```
        addr = MyPTE1->Get_Address();
```

Visual C#

```
        addr = MyPTE1.Get_Address();
```

Matlab

```
        addr = MyPTE1.Get_Address
```

See Also

Set Address
Connect by Address
Get List of Available Addresses

4.4 (h) - Get List of Connected Serial Numbers

Declaration

```
short Get_Available_SN_List(string SN_List)
```

Description

This function takes a user defined variable and updates it with a list of serial numbers for all available (currently connected) switch matrices.

Parameters

Data Type	Variable	Description
string	SN_List	Required. string variable which will be updated with a list of all available serial numbers, separated by a single space character; for example “11301020001 11301020002 11301020003”.

Return Values

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

Example

Visual Basic

```
        If MyPTE1.Get_Available_SN_List(SN_List) > 0 Then
                array_SN() = Split(SN_List, " ")
                        ' Split the list into an array of serial numbers
                For i As Integer = 0 To array_SN.Length - 1
                        ' Loop through the array and use each serial number
                Next
        End If
```

Visual C++

```
        if (MyPTE1 ->Get_Available_SN_List(SN_List) > 0)
        {
                // split the List into array of SN's
        }
```

Visual C#

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

Matlab

```
        [status, SN_List]= MyPTE1.Get_Available_SN_List(SN_List)
        if status > 0
                % split the List into array of SN's
        end
```

See Also

Connect
Get List of Available Addresses

4.4 (i) - Get List of Available Addresses

Declaration

```
short Get_Available_Address_List(string Add_List)
```

Description

This function takes a user defined variable and updates it with a list of addresses of all connected switch matrices.

Parameters

Data Type	Variable	Description
string	Add_List	Required. string variable which the function will update with a list of addresses separated by a single space character, for example, “5 101 254 255”

Return Values

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

Example

Visual Basic

```
        If MyPTE1.Get_Available_Add_List(st_Ad_List) > 0 Then
                       ' Get list of available addresses
                array_Ad() = Split(st_Ad_List, " ")
                       ' Split the list into an array of addresses
                For i As Integer = 0 To array_Ad.Length - 1
                       ' Loop through the array and use each address
                Next
        End If
```

Visual C++

```
        if (MyPTE1->Get_Available_Address_List(Add_List) > 0);
        {       // split the List into array of Addresses
        }
```

Visual C#

```
        if (MyPTE1.Get_Available_Address_List(ref(Add_List)) > 0)
        {       // split the List into array of Addresses
        }
```

Matlab

```
        [status, Add_List]= MyPTE1.Get_Available_Address_List(Add_List)
        if status > 0
                % split the List into array of Addresses
        end
```

See Also

Connect by Address
Get List of Connected Serial Numbers

4.4 (j) - Send SCPI Command

Declaration

```
short Send_SCPI(string CommandRequest, ByRef string ReturnStr)
```

Description

This function sends a command to the switch matrix in the form of an ASCII text string and returns the response. These commands provide the primary method for controlling the the switch matrix, from setting switch states, to querying the various parameters of the matrix. See Summary of Commands / Queries for the full list.

Parameters

Data Type	Variable	Description
string	CommandRequest	The text command to send to the switch matrix
string	ReturnStr	A string variable passed by reference, to be updated with the response from the switch matrix

Return Values

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

Examples

Visual Basic

```
        If MyPTE1.Send_SCPI("SN?", serial_no) > 0 Then
                MsgBox ("The connected switch matrix is " & serial_no)
                        ' Display a message stating the serial number
        End If
```

Visual C++

```
        if (MyPTE1->Send_SCPI("SN?", serial_no) > 0 )
        {
                MessageBox::Show("The connected switch matrix is " + serial_no);
                        // Display a message stating the serial number
        }
```

Visual C#

```
        if (MyPTE1.Send_SCPI("SN?", ref(serial_no)) > 0 )
        {
                MessageBox.Show("The connected switch matrix is " + serial_no);
                        // Display a message stating the serial number
        }
```

Matlab

```
        [status, serial_no] = MyPTE1.Send_SCPI("SN?", serial_no)
        if status > 0
                h = msgbox('The connected switch matrix is ', serial_no)
                        % Display a message stating the serial number
        end
```

See Also

Summary of Commands / Queries

4.4 (k) - Get USB Connection Status

Declaration

```
short GetUSBConnectionStatus()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
short	0	No connection
short	1	USB connection to switch matrix is active

Examples

Visual Basic

```
        If MyPTE1.GetUSBConnectionStatus = 1 Then
                ' switch matrix is connected
        End If
```

Visual C++

```
        if (MyPTE1->GetUSBConnectionStatus() == 1)
        {
                // switch matrix is connected
        }
```

Visual C#

```
        if (MyPTE1.GetUSBConnectionStatus() == 1)
        {
                // switch matrix is connected
        }
```

Matlab

```
        usbstatus = MyPTE1.GetUSBConnectionStatus
        if usbstatus == 1
                % switch matrix is connected
        end
```

See Also

Get USB Device Name

4.4 (l) - Get Firmware

Declaration

```
short GetExtFirmware(short A0, short A1, short A2, string Firmware)
```

Description

This function returns the internal firmware version of the switch matrix along with three reserved variables (for factory use).

Parameters

Data Type	Variable	Description
short	A0	Required. User defined variable for factory use only.
short	A1	Required. User defined variable for factory use only.
short	A2	Required. User defined variable for factory use only.
string	Firmware	Required. User defined variable which will be updated with the current firmware version, for example “B3”.

Return Values

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

Examples

Visual Basic

```
If MyPTE1.GetExtFirmware(A0, A1, A2, Firmware) > 0 Then
        MsgBox ("Firmware version is " & Firmware)
End If
```

Visual C++

```
if (MyPTE1->GetExtFirmware(A0, A1, A2, Firmware) > 0 )
{
        MessageBox::Show("Firmware version is " + Firmware);
}
```

Visual C#

```
if (MyPTE1.GetExtFirmware(ref(A0, A1, A2, Firmware)) > 0 )
{
        MessageBox.Show("Firmware version is " + Firmware);
}
```

Matlab

```
[status, A0, A1, A2, Firmware]=MyPTE1.GetExtFirmware(A0, A1, A2, Firmware)
if status > 0
        h = msgbox('Firmware version is ', Firmware)
end
```

4.5 - DLL Functions for Ethernet Configuration

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

4.5 (a) - Get Ethernet Configuration

Declaration

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

Description

Returns the current IP configuration of the connected switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

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

End If
```

Visual C++

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

}
```

Visual C#

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

}
```

Matlab

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

end
```

See Also

Get MAC Address
Get TCP/IP Port

4.5 (b) - Get IP Address

Declaration

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

Description

This function returns the current IP address of the connected switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

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

4.5 (c) - Get MAC Address

Declaration

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

Description

This function returns the MAC (media access control) address, the physical address, of the connected switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

```
[status, M1, M2, M3, M4, M5, M6] = MyPTE1.GetEthernet_MACAddess(M1, M2, M3,
i4, M5, M6)
If status > 0
       h = msgbox ("MAC address: ", M1, ".", M2, ".", M3, ".", M4, ".", M5,
                                                                   ".", M6)
end
```

See Also

Get Ethernet Configuration

4.5 (d) - Get Network Gateway

Declaration

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

Description

This function returns the IP address of the network gateway to which the switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Save Network Gateway

4.5 (e) - Get Subnet Mask

Declaration

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

Description

This function returns the subnet mask used by the network gateway to which the switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Save Subnet Mask

4.5 (f) - Get TCP/IP Port

Declaration

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

Description

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

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

Parameters

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

Return Values

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

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Save TCP/IP Port

4.5 (g) - Get DHCP Status

Declaration

```
int GetEthernet_UseDHCP()
```

Description

This function indicates whether the switch matrix 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
int	0	DHCP not in use (IP settings are static and manually configured)
int	1	DHCP in use (IP settings are assigned automatically by the network)

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

```
        DHCPstatus = MyPTE1.GetEthernet_UseDHCP
```

See Also

Get Ethernet Configuration
Use DHCP

4.5 (h) - Get Password Status

Declaration

```
int GetEthernet_UsePWD()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

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

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

```
PWDstatus = MyPTE1.GetEthernet_UsePWD
```

See Also

Get Password
Use Password
Set Password

4.5 (i) - Get Password

Declaration

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

Description

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

Parameters

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

Return Values

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

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Password Status
Use Password
Set Password

4.5 (j) - Save IP Address

Declaration

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

Description

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

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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get IP Address

4.5 (k) - Save Network Gateway

Declaration

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

Description

This function sets the IP address of the network gateway to which the switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get Network Gateway

4.5 (l) - Save Subnet Mask

Declaration

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

Description

This function sets the subnet mask of the network to which the switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Ethernet Configuration
Get Subnet Mask

4.5 (m) - Save TCP/IP Port

Declaration

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

Description

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

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

Parameters

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

Return Values

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

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get TCP/IP Port

4.5 (n) - Use DHCP

Declaration

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

Description

This function enables or disables DHCP (dynamic host control protocol). When enabled the IP configuration of the switch matrix 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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

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

Visual C++

```
        status = MyPTE1->SaveEthernet_UseDHCP(1);
```

Visual C#

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

Matlab

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

See Also

Get DHCP Status

4.5 (o) - Use Password

Declaration

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

Description

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

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
int	0	Command failed
int	1	Command completed successfully

Example

Visual Basic

```
        status = MyPTE1.SaveEthernet_UsePWD(1)
```

Visual C++

```
        status = MyPTE1->SaveEthernet_UsePWD(1);
```

Visual C#

```
        status = MyPTE1.SaveEthernet_UsePWD(1);
```

Matlab

```
        status = MyPTE1.SaveEthernet_UsePWD(1)
```

See Also

Get Password Status
Get Password
Set Password

4.5 (p) - Set Password

Declaration

```
int SaveEthernet_PWD(string Pwd)
```

Description

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

Parameters

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

Return Values

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

Example

Visual Basic

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

Visual C++

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

Visual C#

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

Matlab

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

See Also

Get Password Status
Get Password
Use Password

5 - USB Control in a Linux Environment

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

To open a connection to the switch matrix, the Vendor ID and Product ID are required:

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

Communication with the switch matrix is carried out by way of USB Interrupt. The transmitted and received buffer sizes are 64 Bytes each:

- Transmit Array = [Byte 0][Byte1][Byte2]…[Byte 63]
- Returned Array = [Byte 0][Byte1][Byte2]…[Byte 63]

In most cases, the full 64 byte buffer size is not needed so any unused bytes become "don't care" bytes; they can take on any value without affecting the operation of the matrix.

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

5.1 - Core Commands / Queries

The commands that can be sent to the switch matrix are summarized in the table below and detailed on the following pages.

	Description	Command Code (Byte 0)
a	Get Device Model Name	40
b	Get Device Serial Number	41
c	Send SCPI Command	1
d	Get Firmware	99

5.1 (a) - Get Device Model Name

Description

Returns the Mini-Circuits part number of the switch matrix.

Transmit Array

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

Returned Array

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

Example

The following array would be returned for ZTVX-10-12-S (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1	90	ASCII character code for Z
2	84	ASCII character code for T
3	86	ASCII character code for V
4	88	ASCII character code for X
5	45	ASCII character code for -
6	49	ASCII character code for 1
7	48	ASCII character code for 0
8	45	ASCII character code for -
9	49	ASCII character code for 1
10	50	ASCII character code for 2
11	0	Zero value byte to indicate end of string

See Also

Get Device Serial Number
SCPI: Get Model Name

5.1 (b) - Get Device Serial Number

Description

Returns the serial number of the switch matrix.

Transmit Array

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

Returned Array

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

Example

The following example indicates that the connected switch matrix has serial number 11309220111 (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1	49	ASCII character code for 1
2	49	ASCII character code for 1
3	51	ASCII character code for 3
4	48	ASCII character code for 0
5	57	ASCII character code for 9
6	50	ASCII character code for 2
7	50	ASCII character code for 2
8	48	ASCII character code for 0
9	49	ASCII character code for 1
10	49	ASCII character code for 1
11	49	ASCII character code for 1
12	0	Zero value byte to indicate end of string

See Also

Get Device Model Name
SCPI: Get Serial Number

5.1 (c) - Send SCPI Command

Description

This function sends an SCPI command to the switch matrix and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products.

Transmit Array

Byte	Data	Description
0	1	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	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

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

The ASCII character codes representing the 4 characters in this command should be sent in bytes 1 to 4 of the transmit array as follows (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

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

The returned array for ZTVX-10-12-S 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	86	ASCII character code for V
4	88	ASCII character code for X
5	45	ASCII character code for -
6	49	ASCII character code for 1
7	48	ASCII character code for 0
8	45	ASCII character code for -
9	49	ASCII character code for 1
10	50	ASCII character code for 2
11	0	Zero value byte to indicate end of string

See Also

Summary of Commands / Queries

5.1 (d) - Get Firmware

Description

This function returns the internal firmware version of the switch matrix.

Transmit Array

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

Returned Array

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

Example

The below returned array indicates that the system has firmware version “C3” (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	49	Not significant
2	77	Not significant
3	78	Not significant
4	63	Not significant
5	67	ASCII character code for C
6	51	ASCII character code for 3

See Also

SCPI: Get Firmware

5.2 - Ethernet Configuration Commands / Queries

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

5.2 (a) - Set Static IP Address

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Get Static IP Address
Reset Ethernet Configuration

5.2 (b) - Set Static Subnet Mask

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Get Static Subnet Mask
Reset Ethernet Configuration

5.2 (c) - Set Static Network Gateway

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Get Static Network Gateway
Reset Ethernet Configuration

5.2 (d) - Set HTTP Port

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Set Telnet Port
Get HTTP Port
Get Telnet Port
Reset Ethernet Configuration

5.2 (e) - Set Telnet Port

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Set HTTP Port
Get HTTP Port
Get Telnet Port
Reset Ethernet Configuration

5.2 (f) - Use Password

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Set Password
Get Password Status
Get Password
Reset Ethernet Configuration

5.2 (g) - Set Password

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use Password
Get Password Status
Get Password
Reset Ethernet Configuration

5.2 (h) - Use DHCP

Description

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

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	205	Interrupt code for Use DHCP
2	DHCP_Mode	0 = DCHP disabled (static IP settings in use) 1 = DHCP enabled (default - dynamic IP in use)
3 - 63	Not significant	Any value

Returned Array

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

Example

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

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

See Also

Use DHCP
Get DHCP Status
Get Dynamic Ethernet Configuration
Reset Ethernet Configuration

5.2 (i) - Get Static IP Address

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Set Static IP Address

5.2 (j) - Get Static Subnet Mask

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Set Static Subnet Mask

5.2 (k) - Get Static Network Gateway

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use DHCP
Set Static Network Gateway

5.2 (l) - Get HTTP Port

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Set HTTP Port
Set Telnet Port
Get Telnet Port

5.2 (m) - Get Telnet Port

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Set HTTP Port
Set Telnet Port
Get HTTP Port

5.2 (n) - Get Password Status

Description

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

Transmit Array

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

Returned Array

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

Example

The following returned array indicates that password protection is enabled:

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

See Also

Use Password
Set Password
Get Password

5.2 (o) - Get Password

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Use Password
Set Password
Get Password Status

5.2 (p) - Get DHCP Status

Description

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

Transmit Array

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

Returned Array

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

Example

The following returned array indicates that DHCP is enabled:

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

See Also

Use DHCP
Get Dynamic Ethernet Configuration

5.2 (q) - Get Dynamic Ethernet Configuration

Description

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

Transmit Array

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

Returned Array

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

Example

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

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

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

See Also

Use DHCP
Get DHCP Status

5.2 (r) - Get MAC Address

Description

Returns the MAC address of the switch matrix.

Transmit Array

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

Returned Array

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

Example

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

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

See Also

Get Dynamic Ethernet Configuration

5.2 (s) - Reset Ethernet Configuration

Description

Forces the switch matrix to resest and adopt the latest Ethernet configuration. Must be sent after any changes are made to the configuration.

Transmit Array

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

Returned Array

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