Test Solutions - Programming Manual

ZT Series
Custom Switch & Attenuator Matrices

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

Important Notice

Trademarks

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' ZT Series custom switch matrices. 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' ZT Series Custom Switch Matrices

Communication with the ZT Series system 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 the USB interrupt codes on a Linux operating system (see USB Control in a Linux Environment)
3. 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 ZT Series is accessible using a series of ASCII text commands and queries, as detailed in the following section.

2.1 - Summary of ASCII Commands / Queries

This section summarizes the general text commands / queries supported by most Mini-Circuits' ZT Series custom switch systems. Individual models may support additional commands based on the unique configuration of the matrix, typically this can include commands to set specific paths consisting of multiple switches.

2.1 (a) - ASCII - System Queries

	Description	Command/Query
a	Get Model Name	MN?
b	Get Serial Number	SN?
c	Get Internal Temperature	T[sensor]?
d	Get Heat Alarm	HEATALARM?
e	Get Fan Status	FANSTATE?
f	Get Fan Alarm	FANALARM?
g	Get Firmware	FIRMWARE?

2.1 (b) - ASCII - Control Commands / Queries

	Description	Command/Query
a	Reset all Switch States	CLEARALL
b	Set Switch Path	:PATH:[start]:[end]
c	Get Switch Path	:PATH:[start]?
d	Set Switch State	:[switch_type]:[switch]:STATE:[value]
e	Set Switch State - Short Hand	C[switch]=[state]
f	Get Switch State	:[switch_type]:[switch]:STATE?
g	Get Switch State - Obsolete	GETSSW[switch]?
h	Get Switch Counter	:[switch_type]:[switch]:SCOUNTER?
i	Get Switch Counter (SPDT) - Obsolete	GETCSW[switch]?
j	Get Switch Counter (SP4T & SP6T) - Obsolete	GETCSW[switch]-[port]
k	Set Attenuator Value	RUDAT:[name]:ATT:[value]
l	Get Attenuator Value	RUDAT:[name]:ATT?
k	Save Switch Counters	OPERATIONDATA:SAVE

2.1 (c) - ASCII - Ethernet Configuration Commands

These functions provide a method of configuring the system's Ethernet IP settings, while connected via Ethernet. The functionality is not available in all models. Please contact testsolutions@minicircuits.com if this functionality is required.

	Description	Command/Query
a	Set Static IP Address	:ETHERNET:CONFIG:IP:[ip]
b	Get Static IP Address	:ETHERNET:CONFIG:IP?
c	Set Static Subnet Mask	:ETHERNET:CONFIG:SM:[mask]
d	Get Static Subnet Mask	:ETHERNET:CONFIG:SM?
e	Set Static Network Gateway	:ETHERNET:CONFIG:NG:[gateway]
f	Get Static Network Gateway	:ETHERNET:CONFIG:NG?
g	Set HTTP Port	:ETHERNET:CONFIG:HTPORT:[port]
h	Get HTTP Port	:ETHERNET:CONFIG:HTPORT?
i	Set Telnet Port	:ETHERNET:CONFIG:TELNETPORT:[port]
j	Get Telnet Port	:ETHERNET:CONFIG:TELNETPORT?
k	Set Password Requirement	:ETHERNET:CONFIG:PWDENABLED:[enabled]
l	Get Password Requirement	:ETHERNET:CONFIG:PWDENABLED?
m	Set Password	:ETHERNET:CONFIG:PWD:[pwd]
n	Get Password	:ETHERNET:CONFIG:PWD?
o	Set DHCP Status	:ETHERNET:CONFIG:DHCPENABLED:[enabled]
p	Get DHCP Status	:ETHERNET:CONFIG:DHCPENABLED?
q	Get MAC Address	:ETHERNET:CONFIG:MAC?
r	Get Current Ethernet Configuration	:ETHERNET:CONFIG:LISTEN?
s	Update Ethernet Settings	:ETHERNET:CONFIG:INIT

2.2 - ASCII - System Queries

2.2 (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=ZT-160

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

See Also

Get Serial Number

2.2 (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=11507140025

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

See Also

Get Model Name

2.2 (c) - Get Internal Temperature

Description

Returns the temperature in degrees Celsius from a specific sensor within the matrix.

Command

```
T[sensor]?
```

Variable	Description
[sensor]	The sensor number to poll, from 0 to 3 (model dependent)

Return Value

```
[temperature]
```

Variable	Description
[temperature]	The recorded temperature in degrees Celsius

Examples

String to Send	String Returned
T0?	25.50
T1?	27.25

DLL Implementation: `Send_SCPI("T0?", RetStr)`
HTTP Implementation: `http://10.10.10.10/T0?`

See Also

Get Heat Alarm
Get Fan Status
Get Fan Alarm

2.2 (d) - Get Heat Alarm

Description

Returns a warning if the temperature exceeds a factory specified limit.

Command Syntax

HEATALARM?

Return String

[alarm]

Variable	Value	Description
[alarm]	0	Internal temperature within specification
	1	Internal temperature exceeds specification

Examples

String to Send	String Returned
HEATALARM?	0

DLL Implementation: `Send_SCPI("HEATALARM?", RetStr)`
HTTP Implementation: `http://10.10.10.10/HEATALARM?`

See Also

Get Internal Temperature
Get Fan Status
Get Fan Alarm

2.2 (e) - Get Fan Status

Description

Indicates whether the fans are currently operating (fan operation is dictated by the internal temperature of the matrix).

Note: Fans not included on all models, refer to model specifications.

Command Syntax

FANSTATE?

Return String

[state]

Variable	Value	Description
[state]	0	Fans are not operating
	1	Fans are operating

Examples

String to Send	String Returned
FANSTATE?	1

DLL Implementation: `Send_SCPI("FANSTATE?", RetStr)`
HTTP Implementation: `http://10.10.10.10/FANSTATE?`

See Also

Get Internal Temperature
Get Heat Alarm
Get Fan Alarm

2.2 (f) - Get Fan Alarm

Description

Returns a warning if the fan operation is obstructed.

Note: Fans not included on all models, refer to model specifications.

Command Syntax

FANALARM?

Return String

[alarm]

Models with 1 fan:

Variable	Value	Description
[alarm]	0 or 2	Fans operating normally
	1 or 3	Fan is stuck

Models with 2 fans:

Variable	Value	Description
[alarm]	0	Fans operating normally
	1	Fan 1 is stuck
	2	Fan 2 is stuck
	3	Fans 1 & 2 are stuck

Examples

String to Send	String Returned
FANALARM?	0

DLL Implementation: `Send_SCPI("FANALARM?", RetStr)`
HTTP Implementation: `http://10.10.10.10/FANALARM?`

See Also

Get Internal Temperature
Get Heat Alarm
Get Fan Status

2.2 (g) - 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_SCPI("FIRMWARE?", RetStr)`
HTTP Implementation: `http://10.10.10.10/FIRMWARE?`

2.3 - ASCII - Control Commands / Queries

2.3 (a) - Reset all Switch States

Description

Resets all switches to their default state:

- SPDT switches: Com to port 1
- MTS transfer switches: J1 <> J3 & J2 <> J4
- SP4T / SP6T / SP8T switches: All ports disconnected
- Programmable attenuators are unaffected

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

See Also

Set Switch State
Set Switch State - Short Hand
Get Switch State

2.3 (b) - Set Path

Description

Applies to systems with pre-defined switch paths. Sets all required switches to connect an "input" and "output" port for a pre-defined switch path. Refer to the readme file and block diagram supplied with the system for details of the configured paths and port names.

Command Syntax

```
:PATH:[start_port]:[end_port]
```

Variable	Description
[start_port]	The "input" port for the path to be connected
[end_port]	The "output" port for the path to be connected

Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed (path not set)
	1	Command completed successfully

Examples

String to Send	String Returned
:PATH:A1:B5	1

DLL Implementation: `Send_SCPI(":PATH:A1:B5", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PATH:A1:B5`

See Also

Get Path
Set Switch State
Get Switch State

2.3 (c) - Get Path

Description

Applies to systems with pre-defined switch paths. Identifies the "output" port connected to a specified "input" port. Refer to the readme file and block diagram supplied with the system for details of the configured paths and port names.

Command Syntax

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

Variable	Description
[start_port]	The "input" port for the path to be queried

Return String

```
[end_port]
```

Variable	Description
[end_port]	The "output" port connected at the end of the path to the specified input port

Examples

String to Send	String Returned
:PATH:A1?	B5

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

See Also

Set Path
Set Switch State
Get Switch State

2.3 (d) - Set Switch State

Description

Set the state of a specific switch within the system.

Command Syntax

```
:[switch_type]:[address]:STATE:[value]
```

[switch_type]	[value]	Description
MTS	1 2	J1 <> J3 & J2 <> J4 J1 <> J2 & J3 <> J4
SPDT	1 2	COM <> 1 COM <> 2
SP4T	0 1 to 4	All ports disconnected COM<>1 to COM <> 4
SP6T	0 1 to 6	All ports disconnected COM<>1 to COM <> 6
SP8T	0 1 to 8	All ports disconnected COM<>1 to COM <> 8

Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed (switch not set)
	1	Command completed successfully

Examples

String to Send	String Returned
:SPDT:1:STATE:2	1
:SP4T:2:STATE:0	1
:MTS:3:STATE:1	1

DLL Implementation: `Send_SCPI(":SP4T:1:STATE:3", RetStr)`
HTTP Implementation: `http://10.10.10.10/:SP4T:1:STATE:3`

See Also

Set Switch State - Short Hand
Get Switch State

2.3 (e) - Set Switch State - Short Hand

Description

Set the state of a specific switch within the system. An abbreviated form of the "set switch" commands can be used to allow setting of multiple switch states in a single command string. Separate each command with a semi-colon (;).

Command Syntax

```
:C[address]=[value]
```

Switch Type	[value]	Description
MTS	1 2	J1 <> J3 & J2 <> J4 J1 <> J2 & J3 <> J4
SPDT	1 2	COM <> 1 COM <> 2
SP4T	0 1 to 4	All ports disconnected COM<>1 to COM <> 4
SP6T	0 1 to 6	All ports disconnected COM<>1 to COM <> 6
SP8T	0 1 to 8	All ports disconnected COM<>1 to COM <> 8

Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed (switch not set)
	1	Command completed successfully

Examples

String to Send	String Returned
:C1=1	1
:C2=0	1
:C3=8	1
:C1=1;:C2=0;:C3=8	1

DLL Implementation: `Send_SCPI(":C1=1;:C2=0;:C3=8", RetStr)`
HTTP Implementation: `http://10.10.10.10/:C1=1;:C2=0;:C3=8`

See Also

Set Switch State
Get Switch State

2.3 (f) - Get Switch State

Description

Read the state of a specific switch within the system.

Command Syntax

```
:[switch_type]:[address]:STATE?
```

Return String

```
[value]
```

[switch_type]	[value]	Description
MTS	1 2	J1 <> J3 & J2 <> J4 J1 <> J2 & J3 <> J4
SPDT	1 2	COM <> 1 COM <> 2
SP4T	0 1 to 4	All ports disconnected COM<>1 to COM <> 4
SP6T	0 1 to 6	All ports disconnected COM<>1 to COM <> 6
SP8T	0 1 to 8	All ports disconnected COM<>1 to COM <> 8

Examples

String to Send	String Returned
:SPDT:1:STATE?	2
:SP4T:2:STATE?	0
:MTS:3:STATE?	1

DLL Implementation: `Send_SCPI(":SP4T:1:STATE?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:SP4T:1:STATE?`

See Also

Set Switch State
Set Switch State - Short Hand

2.3 (g) - Get Switch State - Obsolete

Description

Get the state of an individual switch.

Command

```
GETSSW[switch]?
```

Variable	Description
[switch]	The number of the switch to check

Return Value

```
[state]
```

Variable	Value	Description
[state]	0	Com port disconnected (SP4T and SP6T switches only)
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3 (SP4T & SP6T switches only)
	4	Com port connected to port 4 (SP4T & SP6T switches only)
	5	Com port connected to port 5 (SP6T switches only)
	6	Com port connected to port 6 (SP6T switches only)

Examples

String to Send	String Returned
GETSSW1?	1
GETSSW2?	1

DLL Implementation: `Send_SCPI("GETSSW1?", RetStr)`
HTTP Implementation: `http://10.10.10.10/GETSSW1?`

See Also

Set Switch State
Set Switch State - Short Hand
Get Switch State

2.3 (h) - 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

:[switch_type]:[address]:SCOUNTER?

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, SP6T & SP8T)

[$count_1$];[$count_2$]; …[$count_n$]

Variable	Description
[$count_{port}$]	The number of connections to each port

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(":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/:MTS:1:SCOUNTER?
```

See Also

2.3 (i) - Get Switch Counter (SPDT) - Obsolete

Description

Indicates the number of switching cycles undertaken by an individual SPDT switch.

Note: See Save Switch Counters for correct operation.

Command

GETCSW[switch]?

Variable	Description
[switch]	The number of the switch to check

Return Value

[count]

Variable	Description
[count]	The number of switch cycles undertaken

Examples

String to Send	String Returned
GETCSW1?	7594

DLL Implementation: `Send_SCPI("GETCSW1?", RetStr)`
HTTP Implementation: `http://10.10.10.10/GETCSW1?`

See Also

Get Switch Counter
Save Switch Counters

2.3 (j) - Get Switch Counter (SP4T & SP6T) - Obsolete

Description

Indicates the number of times an SP4T or SP6T switch has been connected to a specific port.

Note: See Save Switch Counters for correct operation.

Command

```
GETCSW[switch]-[port]?
```

Variable	Description
[switch]	The number of the switch to check

Variable	Value	Description
[switch]		The number of the switch to check
[port]	1	Count of Com port connected to port 1
	2	Count of Com port connected to port 2
	3	Count of Com port connected to port 3
	4	Count of Com port connected to port 4
	5	Count of Com port connected to port 5 (SP6T only)
	6	Count of Com port connected to port 6 (SP6T only)

Return Value

```
[count]
```

Variable	Description
[count]	The number of switch cycles undertaken

Examples

String to Send	String Returned
GETCSW3-2?	7594

DLL Implementation: `Send_SCPI("GETCSW3-2?", RetStr)`
HTTP Implementation: `http://10.10.10.10/GETCSW3-2?`

See Also

Get Switch Counter
Save Switch Counters

2.3 (k) - Set Attenuator Value

Description

Sets the attenuation level of a specific programmable attenuator.

Command

RUDAT:[name]:ATT:[value]

Variable	Description
[name]	The attenuator name/number
[value]	The attenuation value in dB

Return Value

[status]

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

Examples

String to Send	String Returned
RUDAT:1:ATT:15.75	1
RUDAT:2:ATT:70.0	1

DLL Implementation: `Send_SCPI("RUDAT:1:ATT:15.75", RetStr)`
HTTP Implementation: `http://10.10.10.10/RUDAT:1:ATT:15.75`

See Also

Get Attenuator Value

2.3 (I) - Get Attenuator Value

Description

Returns the attenuation level of a specific programmable attenuator.

Command

RUDAT:[name]:ATT?

Variable	Description
[name]	The attenuator name/number

Return Value

[value]

Variable	Description
[value]	Attenuation value in dB

Examples

String to Send	String Returned
RUDAT:1:ATT?	15.75
RUDAT:2:ATT?	70.0

DLL Implementation: `Send_SCPI("RUDAT:1:ATT?", RetStr)`
HTTP Implementation: `http://10.10.10.10/RUDAT:1:ATT?`

See Also

Set Attenuator Value

2.3 (m) - 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.

Requirements

Contact Mini-Circuits (testsolutions@minicircuits.com) for required firmware version.

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 - ASCII - Ethernet Configuration Commands

These functions provide a method of configuring the system's Ethernet IP settings, while connected via Ethernet. The functionality is not available in all models. Please contact testsolutions@minicircuits.com if this functionality is required.

2.4 (a) - Set Static IP Address

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

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

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

Return String

`[status]`

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

Examples

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

HTTP Implementation:

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

See Also

Get Static IP Address
Set Static Subnet Mask
Set Static Network Gateway
Update Ethernet Settings

2.4 (b) - Get Static IP Address

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:IP?
```

Return String

```
[ip]
```

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

Examples

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

HTTP Implementation:

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

See Also

Set Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration

2.4 (c) - Set Static Subnet Mask

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:SM:[mask]

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

Return String

[status]

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

Examples

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

HTTP Implementation:

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

See Also

Set Static IP Address
Get Static Subnet Mask
Set Static Network Gateway
Update Ethernet Settings

2.4 (d) - Get Static Subnet Mask

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:SM?
```

Return String

```
[mask]
```

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

Examples

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

HTTP Implementation:

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

See Also

Get Static IP Address
Set Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration

2.4 (e) - Set Static Network Gateway

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:NG:[gateway]

Variable	Description
[gateway]	IP address of the network gateway

Return String

[status]

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

Examples

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

HTTP Implementation:
http://10.10.10.10/:ETHERNET:CONFIG:NG:192.168.100.1.0

See Also

Set Static IP Address
Set Static Subnet Mask
Get Static Network Gateway
Update Ethernet Settings

2.4 (f) - Get Static Network Gateway

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:NG?
```

Return String

```
[gateway]
```

Variable	Description
[gateway]	IP address of the network gateway

Examples

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

HTTP Implementation:

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

See Also

Get Static IP Address
Get Static Subnet Mask
Set Static Network Gateway
Get Current Ethernet Configuration

2.4 (g) - Set HTTP Port

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:HTPORT:[port]

Variable	Description
[port]	IP port to be used for HTTP communication. The port will need to be included in all HTTP commands if any other than the default port 80 is selected.

Return String

[status]

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

Examples

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

HTTP Implementation:

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

See Also

Get HTTP Port
Set Telnet Port
Update Ethernet Settings

2.4 (h) - Get HTTP Port

Description

Gets the IP port to be used for HTTP communication.

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

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

Return String

```
[port]
```

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

Examples

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

HTTP Implementation:

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

See Also

Set HTTP Port
Get Telnet Port

2.4 (i) - Set Telnet Port

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:TELNETPORT:[port]
```

Variable	Description
[port]	IP port to be used for Telnet communication. The port will need to be included when initiating a Telnet session if other than the default port 23 is selected.

Return String

```
[status]
```

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

Examples

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

HTTP Implementation:

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

See Also

Set HTTP Port
Get Telnet Port
Update Ethernet Settings

2.4 (j) - Get Telnet Port

Description

Gets the IP port to be used for Telnet communication.

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

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

Return String

```
[port]
```

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

Examples

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

HTTP Implementation:

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

See Also

Get HTTP Port
Set Telnet Port

2.4 (k) - Set Password Requirement

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

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

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

Return String

```
[status]
```

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

Examples

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

HTTP Implementation:

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

See Also

Get Password Requirement
Set Password
Get Password
Update Ethernet Settings

2.4 (l) - Get Password Requirement

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:PWDENABLED?

Return String

[enabled]

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

Examples

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

HTTP Implementation:

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

See Also

Set Password Requirement
Set Password
Get Password

2.4 (m) - Set Password

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

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

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

Return String

```
[status]
```

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

Examples

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

HTTP Implementation:

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

See Also

Set Password Requirement
Get Password Requirement
Get Password
Update Ethernet Settings

2.4 (n) - Get Password

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:PWD?

Return String

[pwd]

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

Examples

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

HTTP Implementation:

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

See Also

Set Password Requirement
Get Password Requirement
Set Password

2.4 (o) - Set DHCP Status

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:DHCPENABLED:[enabled]

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

Return String

[status]

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

Examples

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

HTTP Implementation:

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

See Also

Set Static IP Address
Get DHCP Status
Update Ethernet Settings

2.4 (p) - Get DHCP Status

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:DHCPENABLED?
```

Return String

```
[enabled]
```

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

Examples

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

HTTP Implementation:

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

See Also

Set Static IP Address
Set DHCP Status
Get Current Ethernet Configuration

2.4 (q) - Get MAC Address

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:MAC?
```

Return String

```
[mac]
```

Variable	Description
[mac]	MAC address of the attenuator

Examples

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

HTTP Implementation:

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

See Also

Get Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Get Current Ethernet Configuration

2.4 (r) - Get Current Ethernet Configuration

Description

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

```
:ETHERNET:CONFIG:LISTEN?
```

Return String

```
[ip];[mask];[gateway]
```

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

Examples

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

HTTP Implementation:

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

See Also

Get Static IP Address
Get Static Subnet Mask
Get Static Network Gateway
Update Ethernet Settings

2.4 (s) - Update Ethernet Settings

Description

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

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

Requirements

Please contact testsolutions@minicircuits.com.

Command Syntax

:ETHERNET:CONFIG:INIT

Return String

[status]

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

Examples

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

HTTP Implementation:

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

See Also

Get Current Ethernet Configuration

3 - USB Control in a Windows Environment

Control of Mini-Circuits' ZT 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 ASCII Commands / Queries) and also expose some additional functionality.

3.1 - 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 ZT custom switch matrix.

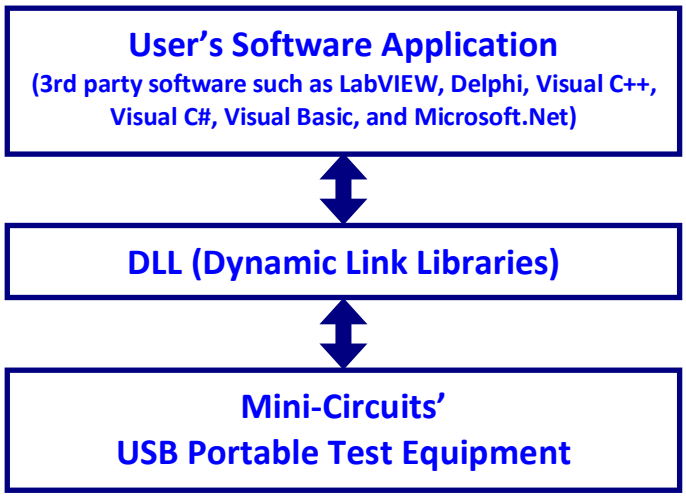


Fig 3.1-a: DLL Interface Concept

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.

3.1 (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' ZT 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 3.1-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 3.1-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 3.1-d):
   `\WINDOWS\System32\Regsvr32 \WINDOWS\System32\mcl_ztxxx.dll`
   For 64-bit Windows operating systems type (see Fig 3.1-e):
   `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mcl_ZTxxx.dll`
4. Hit enter to confirm and a message box will appear to advise of successful registration.

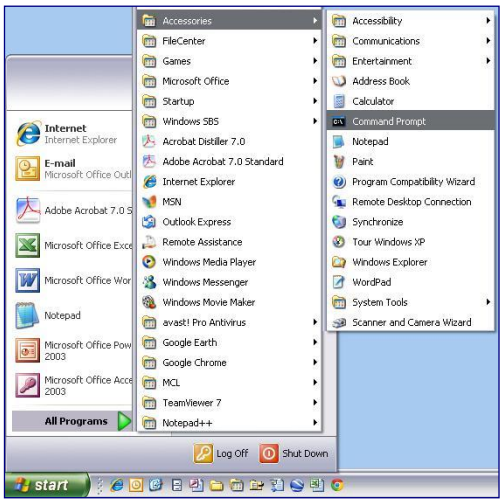


Fig 3.1-b: Opening the Command Prompt in Windows XP

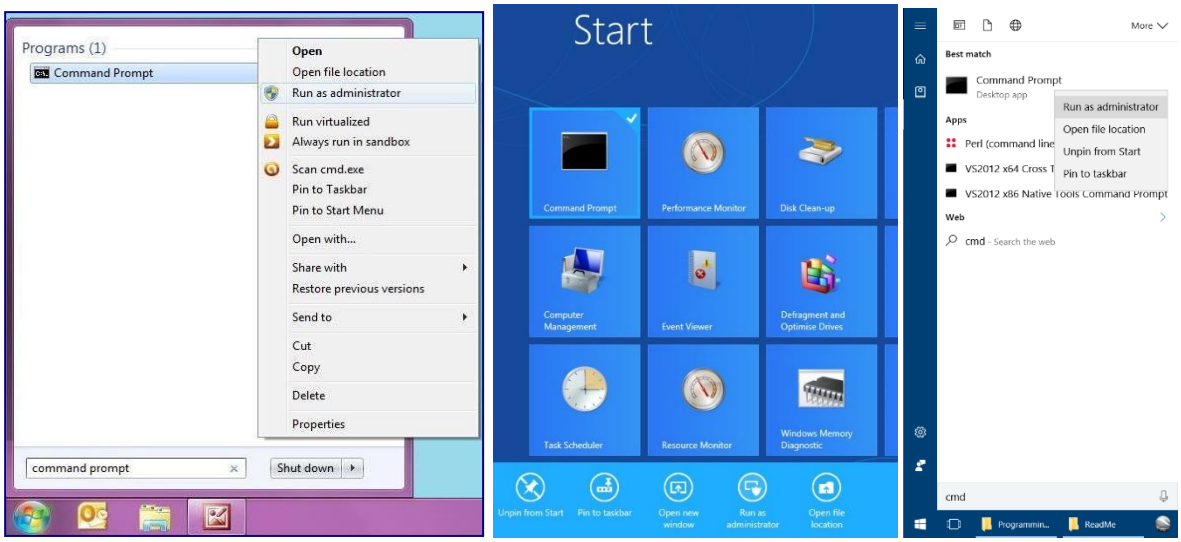


Fig 3.1-c: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

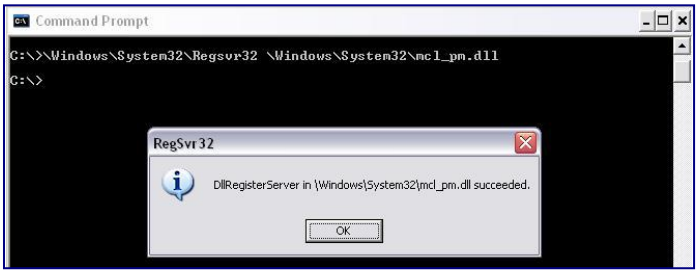


Fig 3.1-d: Registering the DLL in a 32-bit environment

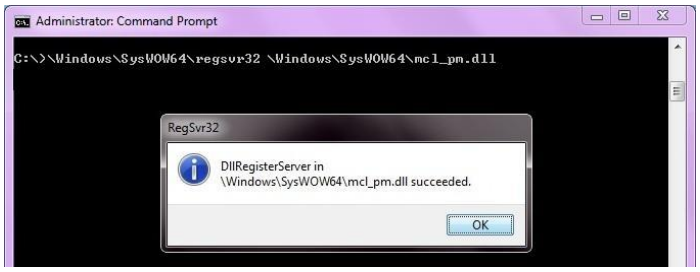


Fig 3.1-e: Registering the DLL in a 64-bit environment

3.1 (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' ZT 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**

3.2 - 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 test 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 modular test systems.

Example Declarations using the ActiveX DLL

Visual Basic

```
Public MyPTE1 As New MCL_ZTxxx.USB_Control
        ' Declare new switch object, assign to MyPTE1
Public MyPTE2 As New MCL_ZTxxx.USB_Control
        ' Declare new switch object, assign to MyPTE2
```

Visual C++

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

Visual C#

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

Matlab

```
MyPTE1 = actxserver('MCL_ZTxxx.USB_Control')
        % Declare new switch object, MyPTE1
MyPTE2 = actxserver('MCL_ZTxxx.USB_Control')
        % Declare new switch object, MyPTE2
```

Example Declarations using the .NET DLL

Visual Basic

```
Public MyPTE1 As New MCL_ZTxxx_64.USB_ZTxxx
        ' Declare new switch object, assign to MyPTE1
Public MyPTE2 As New MCL_ZTxxx_64.USB_ZTxxx
        ' Declare new switch object, assign to MyPTE2
```

Visual C++

```
MCL_ZTxxx_64::USB_ZTxxx ^MyPTE1 = gcnew MCL_ZTxxx_64::USB_ZTxxx();
        // Declare new switch object, assign to MyPTE1
MCL_ZTxxx_64::USB_ZTxxx ^MyPTE2 = gcnew MCL_ZTxxx_64::USB_ZTxxx();
        // Declare new switch object, assign to MyPTE2
```

Visual C#

```
public MCL_ZTxxx_64.USB_ZTxxx MyPTE1 = new MCL_ZTxxx_64.USB_ZTxxx();
        // Declare new switch object, assign to MyPTE1
public MCL_ZTxxx_64.USB_ZTxxx MyPTE2 = new MCL_ZTxxx_64.USB_ZTxxx();
        // Declare new switch object, assign to MyPTE2
```

Matlab

```
MCL_ATT=NET.addAssembly('C:\Windows\SysWOW64\MCL_ZTxxx_64.dll')
MyPTE1 = MCL_ZTxxx_64.USB_ZTxxx       % Declare new switch object
MyPTE1 = MCL_ZTxxx_64.USB_ZTxxx       % Declare new switch object
```

3.3 - 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. Any sequence of DLL functions can be used following this, with the final step being the Disconnect function when the program is complete. For most applications the Send_SCPI 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).

3.3 (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) float GetDeviceTemperature(short TSensor)
l) short GetUSBConnectionStatus()
m) string GetUSBDeviceName()
n) short GetExtFirmware(ByRef short A0, ByRef short A1,
                        ByRef short A2, ByRef string Firmware)
```

3.3 (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)
```

3.4 - DLL Functions for USB Control

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

3.4 (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

3.4 (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

3.4 (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

3.4 (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 then
{
        msgbox('The connected switch matrix is ', ModelName)
                % Display a message stating the model name
}
```

See Also

Read Serial Number

3.4 (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 then
{
        msgbox('The connected switch matrix is ', SN)
                % Display a message stating the serial number
}
```

See Also

Read Model Name

3.4 (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 to Switch Matrix 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

3.4 (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

3.4 (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 then
{
        % split the List into array of SN's
}
```

See Also

Connect
Get List of Available Addresses

3.4 (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 then
{       % split the List into array of Addresses
}
```

See Also

Connect by Address
Get List of Connected Serial Numbers

3.4 (j) - Send Command

Declaration (Model Dependent)

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

Description

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 system, from setting switch states, to querying the various parameters of the matrix. See Switch Matrix Commands / Queries for details of the available commands/queries.

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)
End If
```

Visual C++

```
if (MyPTE1->Send_SCPI("SN?", serial_no) > 0 )
{
        MessageBox::Show("The connected switch matrix is " + serial_no);
}
```

Visual C#

```
if (MyPTE1.Send_SCPI("SN?", ref(serial_no)) > 0 )
{
        MessageBox.Show("The connected switch matrix is " + serial_no);
}
```

Matlab

```
[status, serial_no] = MyPTE1.Send_SCPI("SN?", serial_no)
If status > 0 then
{
        msgbox('The connected switch matrix is ', serial_no)
}
```

See Also

Summary of ASCII Commands / Queries

3.4 (k) - Get Temperature

Declaration

```
float GetDeviceTemperature(short TSensor)
```

Description

This function returns the temperature measured at a specific internal sensor.

Parameters

Data Type	Variable	Description
short	TSensor	Required. short integer variable (1 to 3) to define which temperature sensor to read.

Return Values

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

Examples

Visual Basic

```
MsgBox ("Temperature is " & MyPTE1.GetDeviceTemperature(2))
        ' Display a message box with the device temperature
```

Visual C++

```
MessageBox::Show("Temperature is " + MyPTE1->GetDeviceTemperature(2));
        // Display a message box with the device temperature
```

Visual C#

```
MessageBox.Show("Temperature is " + MyPTE1.GetDeviceTemperature(2));
        // Display a message box with the device temperature
```

Matlab

```
[temp,status]=MyPTE1.GetDeviceTemperature(2)
Msgbox('Temperature is ', temp)
        % Display a message box with the device temperature
```

3.4 (I) - 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 then
{
        % switch matrix is connected
}
```

See Also

Get USB Device Name

3.4 (m) - Get USB Device Name

Declaration

```
string GetUSBDeviceName()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
string	Name	Device name of the switch matrix

Examples

Visual Basic

```
usbname = MyPTE1.GetUSBDeviceName()
        ' Return the USB device name as a string
```

Visual C++

```
usbname = MyPTE1->GetUSBDeviceName();
        // Return the USB device name as a string
```

Visual C#

```
usbname = MyPTE1.GetUSBDeviceName();
        // Return the USB device name as a string
```

Matlab

```
usbname = MyPTE1.GetUSBDeviceName
        % Return the USB device name as a string
```

See Also

Get USB Connection Status

3.4 (n) - 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 then
{
        msgbox('Firmware version is ', Firmware)
}
```

3.5 - 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.

3.5 (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 power sensor in a series of user defined variables. The settings checked are IP address, subnet mask and network gateway.

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address.
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address.
int	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 then
{
        MsgBox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
        MsgBox ("Subnet Mask: ", M1, "." & M2, "." & M3, ".", M4)
        MsgBox ("Gateway: ", GW1, ".", GW2, ".", GW3, ".", GW4)

}
```

See Also

Get MAC Address
Get TCP/IP Port

3.5 (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 power sensor in a series of user defined variables (one per octet).

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example “192” for the IP address “192.168.1.0”).
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example “168” for the IP address “192.168.1.0”).
int	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 then
{
        MsgBox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
}
```

See Also

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

3.5 (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 power sensor as a series of decimal values (one for each of the 6 numeric groups).

Parameters

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

Return Values

Data Type	Value	Description
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,
M4, M5, M6)
If status > 0 then
{
        MsgBox ("MAC address: ", M1, ".", M2, ".", M3, ".", M4, ".", M5, ".",
M6)
}
```

See Also

Get Ethernet Configuration

3.5 (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 power sensor is currently connected. A series of user defined variables are passed to the function to be updated with the IP address (one per octet).

Parameters

Data Type	Variable	Description
int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example “192” for the IP address “192.168.1.0”).
int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example “168” for the IP address “192.168.1.0”).
int	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 then
{
        MsgBox ("Gateway: ", IP1, ".", IP2, ".", IP3, ".", IP4)
}
```

See Also

Get Ethernet Configuration
Save Network Gateway

3.5 (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 power sensor is currently connected. A series of user defined variables are passed to the function to be updated with the subnet mask (one per octet).

Parameters

Data Type	Variable	Description
int	b1	Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).
int	b2	Required. Integer variable which will be updated with the second octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).
int	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 then
{
        MsgBox ("Subnet mask: ", b1, ".", b2, ".", b3, ".", b4)
}
```

See Also

Get Ethernet Configuration
Save Subnet Mask

3.5 (f) - Get TCP/IP Port

Declaration

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

Description

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

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

Parameters

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

Return Values

Data Type	Value	Description
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 then
{
        MsgBox ("Port: ", port)
}
```

See Also

Get Ethernet Configuration
Save TCP/IP Port

3.5 (g) - Get DHCP Status

Declaration

```
int GetEthernet_UseDHCP()
```

Description

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

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
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

3.5 (h) - Get Password Status

Declaration

```
int GetEthernet_UsePWD()
```

Description

This function indicates whether the power sensor 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

3.5 (i) - Get Password

Declaration

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

Description

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

Parameters

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

Return Values

Data Type	Value	Description
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 then
{
        MsgBox ("Password: ", pwd)
}
```

See Also

Get Password Status
Use Password
Set Password

3.5 (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 power sensor.

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

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the IP address to set (for example “192” for the IP address “192.168.1.0”).
int	IP2	Required. Second octet of the IP address to set (for example “168” for the IP address “192.168.1.0”).
int	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

3.5 (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 power sensor should connect.

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

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the network gateway IP address (for example “192” for the IP address “192.168.1.0”).
int	IP2	Required. Second octet of the network gateway IP address (for example “168” for the IP address “192.168.1.0”).
int	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

3.5 (I) - 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 power sensor should connect.

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

Parameters

Data Type	Variable	Description
int	IP1	Required. First (highest order) octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).
int	IP2	Required. Second octet of the subnet mask (for example “255” for the subnet mask “255.255.255.0”).
int	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

3.5 (m) - Save TCP/IP Port

Declaration

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

Description

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

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

Parameters

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

Return Values

Data Type	Value	Description
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

3.5 (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 power sensor is assigned automatically by the network server; when disabled the user defined “static” IP settings apply.

Parameters

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

Return Values

Data Type	Value	Description
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

3.5 (o) - Use Password

Declaration

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

Description

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

Parameters

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

Return Values

Data Type	Value	Description
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

3.5 (p) - Set Password

Declaration

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

Description

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

Parameters

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

Return Values

Data Type	Value	Description
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

4 - USB Control in a Linux Environment

To open a USB connection to Mini-Circuits ZT Series test systems, the Vendor ID and Product ID are required:

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

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

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

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

4.1 - Summary of Commands

	Description	Code (Byte 0)
a	Get Device Model Name	40
b	Get Device Serial Number	41
c	Send ASCII / SCPI Command	42

4.2 - Detailed Description of Commands

4.2 (a) - Get Device Model Name

Description

Returns the Mini-Circuits part number of the connected system.

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 ZTM-999:

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	77	ASCII character code for M
4	45	ASCII character code for -
5	57	ASCII character code for 9
6	57	ASCII character code for 9
7	57	ASCII character code for 9
8	0	Zero value byte to indicate end of string

See Also

Get Device Serial Number
SCPI: Get Model Name

4.2 (b) - Get Device Serial Number

Description

Returns the serial number of the connected system.

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 system has serial number 1130922011:

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	0	Zero value byte to indicate end of string

See Also

Get Device Model Name
SCPI: Get Serial Number

4.2 (c) - Send ASCII / SCPI Command

Description

This function sends an ASCII / SCPI command to the system and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the ZT Series system's internal components.

Transmit Array

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

Returned Array

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

Example 1 (Get Model Name)

The SCPI command to request the model name is `:MN?` (see Get Model Name). The ASCII character codes representing the 4 characters in this command should be sent in bytes 1 to 4 of the transmit array as follows:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	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 ZTM-999 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	77	ASCII character code for M
4	45	ASCII character code for -
5	57	ASCII character code for 9
6	57	ASCII character code for 9
7	57	ASCII character code for 9
8	0	Zero value byte to indicate end of string

Example 2 (Set Switch)

The ASCII / SCPI command to set an SPDT switch with address 1 to state 4 is
`:SP4T:1:STATE:4`

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	83	ASCII character code for S
3	80	ASCII character code for P
4	68	ASCII character code for D
5	84	ASCII character code for T
6	58	ASCII character code for :
7	83	ASCII character code for S
8	84	ASCII character code for T
9	65	ASCII character code for A
10	84	ASCII character code for T
11	69	ASCII character code for E
12	58	ASCII character code for :
13	52	ASCII character code for 4

The returned array to indicate success would be:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	49	ASCII character code for 1
2	32	ASCII character code for space character
3	45	ASCII character code for -
4	32	ASCII character code for space character
5	83	ASCII character code for S
6	85	ASCII character code for U
7	67	ASCII character code for C
8	67	ASCII character code for C
9	69	ASCII character code for E
10	83	ASCII character code for S
11	83	ASCII character code for S
12	0	Zero value byte to indicate end of string

See Also

Summary of ASCII Commands / Queries

5 - Ethernet Control over IP Networks

Control of Mini-Circuits' ZT 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

5.1 - 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).

5.2 - 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.

5.2 (a) - Setting Switch States Using HTTP

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

http://ADDRESS:PORT/PWD;COMMAND

Where

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

Example 1:

http://192.168.100.100:800/PWD=123;C1=1

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 switch 1 to state 1 (see below for the full explanation of all commands/queries)

Example 2:

http://10.10.10.10/C1=2

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 switch 1 to state 2 (see below for the full explanation of all commands/queries)

5.2 (b) - Querying Switch Properties Using HTTP

The basic format of the HTTP command to query the switch 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 below for the full explanation of all commands/queries)

Example 2:

http://10.10.10.10/GETSSW1?

Explanation:
- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The query is to return the state of switch 1 (see below for the full explanation of all commands/queries)

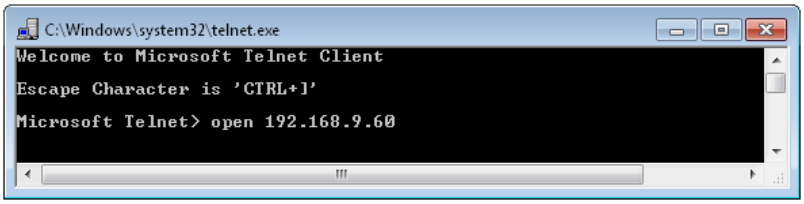
The device will return the result of the query as a string of ASCII characters.

5.2 (c) - Communication Using Telnet

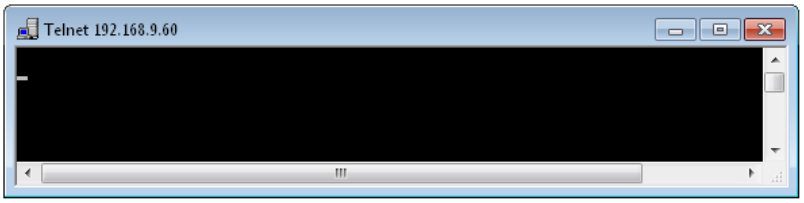
Communication with the device is started by creating a Telnet connection to the switch IP address. On successful connection the “line feed” character will be returned. If the switch matrix has a password enabled then this must be sent as the first command after connection.

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

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

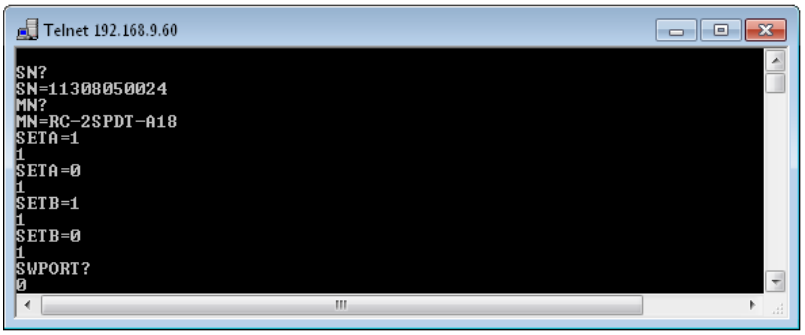


2) The “line feed” character is returned indicating the connection was successful:



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

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



5.3 - Device Discovery Using UDP

In addition to HTTP and Telnet, the RC series of Ethernet controlled switch matrices also provide limited support of the UDP protocol for the purpose of "device discovery." This allows a user to request the IP address and configuration of all Mini-Circuits RC 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).

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

UDP Ports

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

Transmission

The query to send is the model name followed by a question mark, for example **ZT-XXX?** which should be broadcast to the local network using the 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:

ZT-166?

Received Data:

Model Name: ZT-166
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: ZT-166
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: ZT-166
Serial Number: 11302120003
IP Address=192.168.9.103 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-03

6 - Programming & Application Examples

These examples are intended to demonstrate the basics of programming with Mini-Circuits' ZT Series custom switch / attenuator assemblies. If support is required for a specific programming example which isn't covered below then please contact Mini-Circuits through testsolutions@minicircuits.com.

6.1 - Visual Basic (VB) Programming

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

The .NET DLL provides the simplest API for USB control in a Windows environment. The DLL can be used on any combination of 32-bit or 64-bit operating systems and programming environments, as long as the environment supports the .NET framework.

```
Dim zt As New MCL_ZTxxx_64.USB_ZTxxx              ' Declare new instance of the USB control class
Dim Return_String As String = ""                  ' Declare a string to store returned values

zt.Connect()                                      ' Connect to the hardware

zt.Send_SCPI ("MN?", Return_String)               ' Read model name
Console.WriteLine(Return_String)                  ' Print model name

zt.Send_SCPI ("SN?", Return_String)               ' Read serial number
Console.WriteLine(Return_String)                  ' Print serial number

zt.Send_SCPI ("C3=4;C1=2;C11=1", Return_String)   ' Set switches 3, 1 and 11

zt.Send_SCPI ("C3?", Return_String)               ' Get switch 3 state
Console.WriteLine(Return_String)                  ' Print switch state

zt.Send_SCPI ("C1?", Return_String)               ' Get switch 1 state
Console.WriteLine(Return_String)                  ' Print switch state

zt.Send_SCPI ("C11?", Return_String)              ' Get switch 11 state
Console.WriteLine(Return_String)                  ' Print switch state

zt.Disconnect()                                   ' Disconnect on completion
```

6.1 (b) - Ethernet HTTP Connection

Microsoft. NET's WebRequest class can be used to send HTTP commands to the system for Ethernet control from Visual Basic.

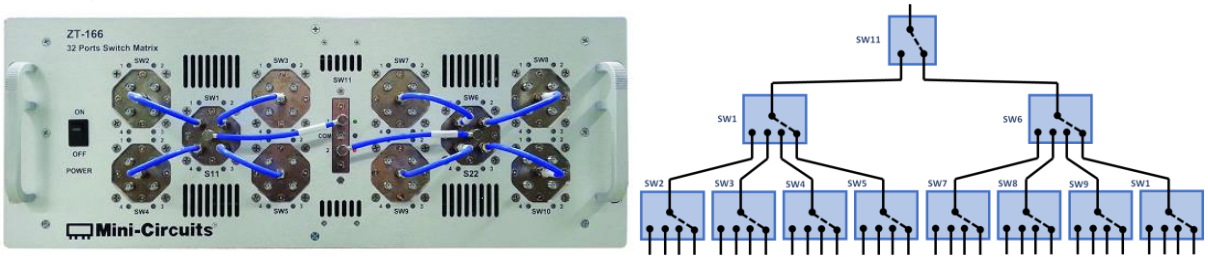
```
Imports System.IO
Imports System.Net

Private Function Send_HTTP_Get(Command_To_Send As String)

    ' Declare a function to send an HTTP command and return the response

    Dim webStream As Stream
    Dim ZT_IP_Address As String = "192.100.1.1"          ' The IP address of the system
    Dim HTTP_Response As String = ""                     ' A string to record the return value

    Dim urlToSend As HttpWebRequest                      ' Use the standard HTTPWebRequest/Response
    Dim urlResponse As HttpWebResponse

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

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

    Dim webStreamReader As New StreamReader(webStream)

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

    Send_HTTP_Get = Trim(HTTP_Response)                  ' Return the response

End Function

' ########################################
' Use the above function to send the HTTP commands to the system
' ########################################

Console.WriteLine(Send_HTTP_Get("MN?”))          ' Read model name
Console.WriteLine(Send_HTTP_Get("SN?”))          ' Read serial number

Send_HTTP_Get("C3=4;C1=2;C11=1”)                 ' Set switches 3, 1 and 11

Console.WriteLine(Send_HTTP_Get("C3?”))          ' Print switch 3 state
Console.WriteLine(Send_HTTP_Get("C1?”))          ' Print switch 1 state
Console.WriteLine(Send_HTTP_Get("C11?”))         ' Print switch 11 state
```

6.1 (c) - Programming ZT-166 as an SP32T Switch (USB & Ethernet Control)

ZT-166 houses 10 independently controlled SP4T switches and a single SPDT switch on the front panel. Using the included software and with external connections made through Mini-Circuits' 141 Series hand-flex cables as shown below, ZT-166 becomes an ultra-wideband, 1 by 32 switch matrix.

1) External Connections

The individual switches of the ZT-166 switch box (ten individual SP4T switches and one SPDT switch) are presented on the front panel, this allows different switch combinations to be constructed externally using Mini-Circuits' 141 Series hand-flex cables. To create an SP32T switch for example, you can make the 10 external connections shown below. The common port of SW11 (the SPDT switch in the centre of the panel) becomes the common port of the SP32T switch. The 32 output ports are made up of ports 1 to 4 of the 8 SP4T switches on the top and bottom rows (SW2-5 and SW7-10).



To set a path of the overall SP32T switch it is necessary to set 3 switches at a time, the input SPDT (SW11) and the two SP4T switches that make up the rest of the path. Brief programming examples are given below for USB and Ethernet control.

2) USB Control

Place the .Net DLL file (MCL_ZT166_64.dll) into the c:\Windows\SysWOW64\ folder and create a reference to the file from within the programming project. A programming example using VB.NET is shown below; the process can be summarised with 4 main steps:

1. Access the DLL
2. Connect to the hardware
3. Send the relevant switch commands
4. Disconnect the hardware

```
Dim Return_String As String = ""                    ' Declare a string to store returned values
Dim myZT166 As New mcl_ZT166_64.USB_ZT166            ' Declare a new instance of the USB_ZT166 class
myZT166.Connect()                                   ' Connect to the physical ZT-166 hardware

' Set SP32T Com port to SW3 port 4
myZT166.Send_SCPI("C3=4", Return_String)          ' Set SP4T switch 3 to port 4
myZT166.Send_SCPI("C1=2", Return_String)          ' Set SP4T switch 1 to port 2
myZT166.Send_SCPI("C11=1", Return_String)         ' Set SPDT switch 11 to port 1

' Set SP32T Com port to SW10 port 2
myZT166.Send_SCPI("C10=2", Return_String)         ' Set SP4T switch 10 to port 2
myZT166.Send_SCPI("C6=3", Return_String)          ' Set SP4T switch 6 to port 3
myZT166.Send_SCPI("C11=2", Return_String)         ' Set SPDT switch 11 to port 2

myZT166.Disconnect()                                ' Disconnect when finished
```

3) Ethernet Control (HTTP)

No DLL file is needed for Ethernet control and in most programming environments you can make use of a standard HTTP library. The process can be summarised as:

1. Define a function to send and receive HTTP commands
2. Send the relevant switch commands

An example using VB.NET is shown below:

```
Imports System.IO
Imports System.Net

Private Function Send_HTTP_Get(Command_To_Send As String)

    ' Declare a function to send an HTTP command and return the response

    Dim webStream As Stream
    Dim ZT_IP_Address As String = "192.168.9.100"' The IP address of the connected ZT-166
    Dim HTTP_Response As String = ""                    ' A string to record the return value

    Dim urlToSend As HttpWebRequest                     ' Use the standard HTTPWebRequest/Response
    Dim urlResponse As HttpWebResponse

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

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

    Dim webStreamReader As New StreamReader(webStream)

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

    Send_HTTP_Get = Trim(HTTP_Response)                 ' Return the response

End Function

' Use the above function to send the sequences of HTTP commands as required to set the switches

Dim Return_String As String = ""                        ' Declare a string to store returned values

' Set SP32T Com port to SW3 port 4

Return_String = Send_HTTP_Get("C3=4")                   ' Set SP4T switch 3 to port 4
Return_String = Send_HTTP_Get("C1=2")                   ' Set SP4T switch 1 to port 2
Return_String = Send_HTTP_Get("C11=1")                  ' Set SPDT switch 11 to port 1

' Set SP32T Com port to SW10 port 2

Return_String = Send_HTTP_Get("C10=2")                  ' Set SP4T switch 10 to port 2
Return_String = Send_HTTP_Get("C6=3")                   ' Set SP4T switch 6 to port 3
Return_String = Send_HTTP_Get("C11=2")                  ' Set SPDT switch 11 to port 2
```

6.2 - Python Programming

6.2 (a) - UDP Query to Identify Systems Connected on the Network

Demonstration of using Python's socket method for sending a broadcast query to the network in order to identify the details (IP address, model name, serial number) of any Mini-Circuits switch systems connected on the network.

```
import socket

# Broadcast address is the bitwise OR between IP and bit-complement of the subnet mask
addr_SND = ('192.168.9.255', 4950)  # broadcast address / MCL test equipment listens on
port 4950
addr_RCV = ('', 4951)               # MCL test equipment replies on port 4951

UDPSock_SND = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Create socket
UDPSock_RCV = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Create socket

UDPSock_RCV.bind(addr_RCV)
UDPSock_RCV.settimeout(1)
Data_RCV=""

Data_SND = "ZT-XXX?"           # Update this to query for the relevant product

print "Sending message '%s'..." % Data_SND
UDPSock_SND.sendto(Data_SND, addr_SND)

i=0
while i<5:                          # Search for up to 5 units

    try:
        Data_RCV,addr_RCV = UDPSock_RCV.recvfrom(4951)
        print Data_RCV

    except:                         # Timeout error if no more responses
        print "No data received."

    i=i+1

print "End of UDP listening..."

UDPSock_SND.close()             # Close sockets
UDPSock_RCV.close()

print 'Client stopped.'
```

6.2 (b) - Ethernet Connection Using Python's urllib2 Library for HTTP

Python's urllib2 library can be used to send HTTP commands to the system when programming with Python.

```
import urllib2

########################################
# Define a function to send an HTTP command and get the result
########################################

def Get_HTTP_Result(CmdToSend):

    # Specify the IP address
    CmdToSend = "http://192.100.1.1/:" + CmdToSend

    # Send the HTTP command and try to read the result
    try:
        HTTP_Result = urllib2.urlopen(CmdToSend)
        PTE_Return = HTTP_Result.read()

    # Catch an exception if URL is incorrect (incorrect IP or disconnected)
    except:
        print "Error, no response from device; check IP address and connections."
        PTE_Return = "No Response!"
        sys.exit()      # Exit the script

    # Return the response
    return PTE_Return

########################################
# Send commands / queries to the device
########################################

print Get_HTTP_Result("MN?")                # Print model name
print Get_HTTP_Result("SN?")                # Print serial number

Get_HTTP_Result("C3=4;C1=2;C11=1")          # Set switches 3, 1 and 11

Read_1 = Get_HTTP_Result(":GETSSW3?")       # Print switch 3 state
print str(Read_1)

Read_1 = Get_HTTP_Result(":GETSSW1?")       # Print switch 1 state
print str(Read_1)

Read_1 = Get_HTTP_Result(":GETSSW11?")      # Print switch 11 state
print str(Read_1)
```

6.2 (c) - USB Connection Using the ActiveX DLL (32-bit Python Distributions)

The majority of 32-bit Python distributions for Windows operating systems support ActiveX so Mini-Circuits' ActiveX DLL can be used to control the system in these environments.

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

zt = win32com.client.Dispatch("mcl_ZTxxx.USB_Control")      # Reference the DLL

zt1_serial_number = ""                          # Serial number if multiple units connected
Conn_Status = zt.Connect(zt1_serial_number)     # Connect to the test system

if Conn_Status == 1:                            # If the connection was successfully created

    MN = zt.Send_SCPI(":MN?", "")               # Read model name
    print MN[2]
    SN = zt.Send_SCPI(":SN?", "")               # Read model name
    print SN[2]

    zt.Send_SCPI(":C3=4;:C1=2;:C11=1", "")      # Set switches SW3, SW1 & SW11

    state = zt.Send_SCPI(":GETSSW3?", "")       # Read state of switch SW3
    print "Switch State:", state[2]

    state = zt.Send_SCPI(":GETSSW1?", "")       # Read state of switch SW1
    print "Switch State:", state[2]

    state = zt.Send_SCPI(":GETSSW11?", "")      # Read state of switch SW11
    print "Switch State:", state[2]

    zt1.Disconnect                              # Disconnect the test system

else:
    print "Could not connect."
```

6.2 (d) - Work-Around for 64-bit Python Distributions with a USB Connection

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

Mini-Circuits can supply on request an executable to interface with the DLL. See Creating an Executable Using the .Net DLL in C# for USB Control for the example source code for such an executable (developed using C#). The below script demonstrates use of this executable in Python script to control the system.

```
import subprocess
from subprocess import Popen, PIPE

# Define a function to use the executable and return the response

def Command_ZT(SCPI_Command):

    sn = '11406170049'      # Serial number of the system to control

    # Call the executable, passing the serial number and SCPI commands as arguments
    pipe = subprocess.Popen("ZT.exe -sn " + sn + " " + SCPI_Command,stdout=subprocess.PIPE)
    pipe.wait
    ZT_Response = pipe.stdout.read()

    return ZT_Response

# Use the above function to send any SCPI commands to the system

print Command_ZT(":MN?")                    # Read model name
print Command_ZT(":SN?")                    # Read serial number

Command_ZT(":C3=4;:C1=2;:C11=1", "")        # Set switches SW3, SW1 & SW11

Read_1 = Command_ZT(":GETSSW1?")            # Read state of switch 1
print str(Read_1)

Read_1 = Command_ZT(":GETSSW3?")            # Read state of switch 3
print str(Read_1)

Read_1 = Command_ZT(":GETSSW11?")           # Read state of switch 11
print str(Read_1)
```

6.3 - C# Programming

6.3 (a) - Console Application Using the .Net DLL with C# for USB Control

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

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

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

- `ZT.exe -s 11601250027 :C3=4;:C1=2;:C11=1` (serial number 11601250027)
- `ZT.exe -a 255 :GETSSW3?` (USB address 255)

6.4 - C++ Programming

6.4 (a) - Console Application Using the ActiveX DLL with C++ for USB Control

The below example is a simple console application using the ActiveX DLL to send specified SCPI commands to the test system. It requires the ActiveX DLL to be installed inserted in the Windows system folder and registered using regsvr32.

```
#include "stdafx.h"
#include <stdio.h>
#include <tchar.h>
#include <Windows.h>

#import "c:\windows\SysWOW64\MCL_ZTxxx.dll"      // Import relevant DLL for the ZT model
using namespace MCL_ZTxxx;                       // Use the relevant namespace (usually model name)

int _tmain(int argc, _TCHAR* argv[])
{
	HRESULT hresult;
	CLSID clsid;
	int status;
	BSTR response, command;

	CoInitialize(NULL);                          // Initialize COM library
	// Retrieve CLSID (note: correct model name must be used for the DLL)
	hresult = CLSIDFromProgID(OLESTR("MCL_ZTxxx.USB_Control"), &clsid);
	if (FAILED(hresult))
	{
		printf("failed1: x%08x\n", hresult);
		goto done;
	}
	_USB_Control *dev;                           // Instance of DLL's USB control class
	hresult=CoCreateInstance(clsid,NULL,CLSCTX_INPROC_SERVER,__uuidof(_USB_Control),(LPVOID *)&dev);
	if (FAILED(hresult))
	{
		printf("failed2: x%08x\n", hresult);
		goto done;
	}

	command = SysAllocString(L"");               // Pass serial number or leave blank for any box
	status = dev->Connect(command);              // Connect to the device
	if (status == 0) {
		printf("Cannot Connect: x%08x\n", st);
		goto done;
	} else {                                     // Send any commands / queries as required…

		command = SysAllocString(L":MN?");       // Pass any SCPI command (:MN? for model name)
		status = dev->Send_SCPI(&command, &response);
		if (status > 0) {
			printf(response);
		}

		dev->Disconnect();                       // Disconnect at end of routine

	}

done:
	CoUninitialize();
	while (1);                                   // Keep the command prompt open
}
```

6.5 - LabVIEW

6.5 (a) - Ethernet Control Using HTTP

The built-in HTTP Get function of LabVIEW provides a simple method for communication with the ZT Series using the Ethernet connection. The process to create a LabVIEW VI based on this approach is as follows.

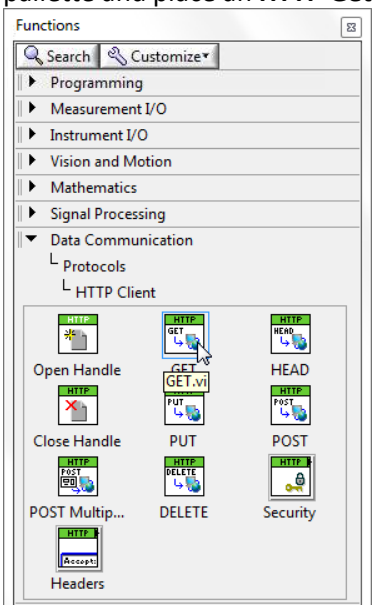
1. Create a New VI (Virtual Instrument)

a. Open LabVIEW with a new, blank VI.

b. In the Block Diagram window (accessible by pressing Ctrl+E), select Functions Palette from the "View" menu at the top of the screen.

2. Create a new HTTP GET function

a. In the Functions menu, click through *Data Communications*, to the *HTTP Client* sub-pallette and place an **HTTP Get** function on the block diagram.



b. Right-click on the *URL* input terminal of the **HTTP Get** function and select *Create* > *Constant*

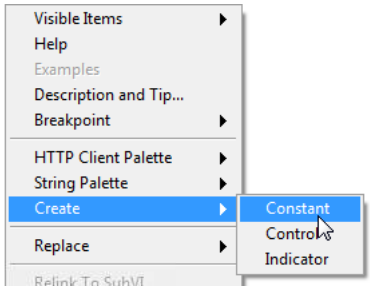
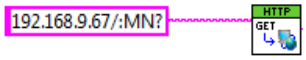
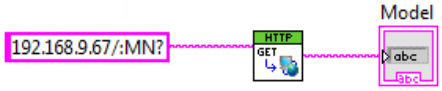


Fig 6.5-i - Create the string input constant to store the command

c. Place the string constant on the block diagram and enter the HTTP command to send (omitting the "http://"); for example "192.168.9.67/:MN?" to read the model name of a ZT system with IP address 192.168.9.67.



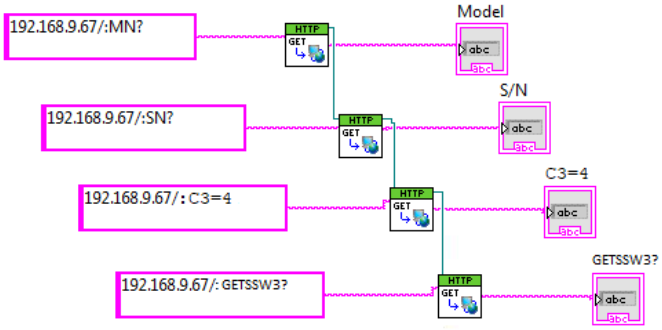
d. Right-click on the *Body* output terminal of the **HTTP Get** function and select *Create* > *Indicator* to store the return value of the HTTP command



3. Add additional HTTP Get commands

a. Step 2 above can be repeated to add as many HTTP Get functions as required

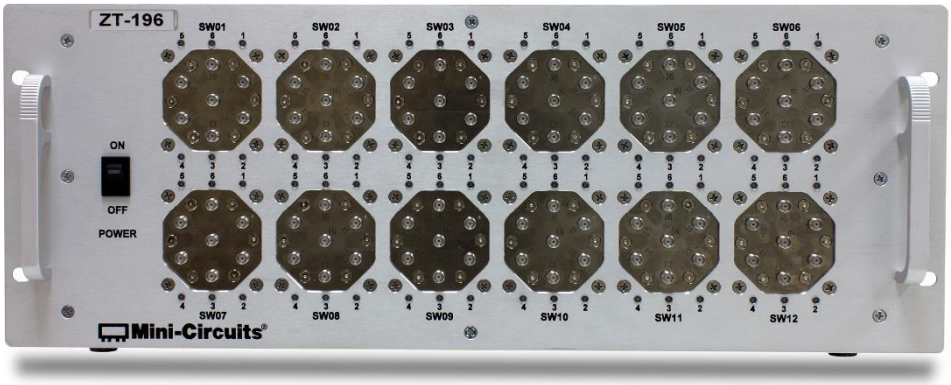
b. HTTP Get can also be used to set the switch by sending the relevant command in the string constant, for example "192.168.9.67/:C3=4" to set switch 3 to state 4; the return value in that case would be 0 or 1 to indicate whether the command was successful



6.5 (b) - USB Control Using the .NET DLL

1. Summary

This example is based on the ZT-196 flexible switch matrix which houses 12 independently controlled SP6T switches on the front panel.

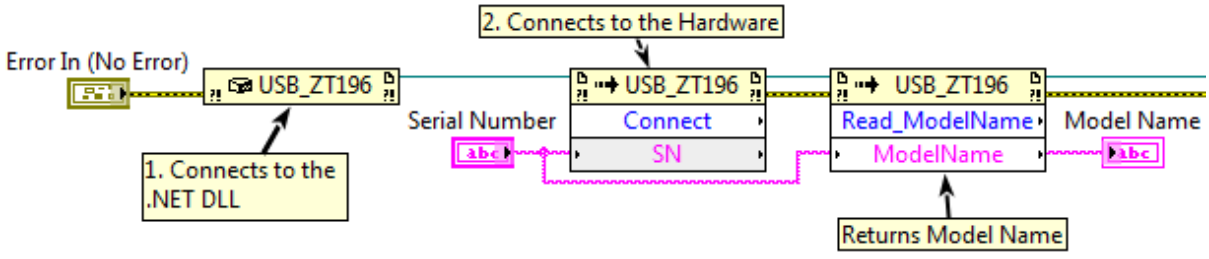


A programming example using LabVIEW and Mini-Circuits' ZT-196 .NET DLL to control the unit is shown below. The process within the VI can be summarized as:

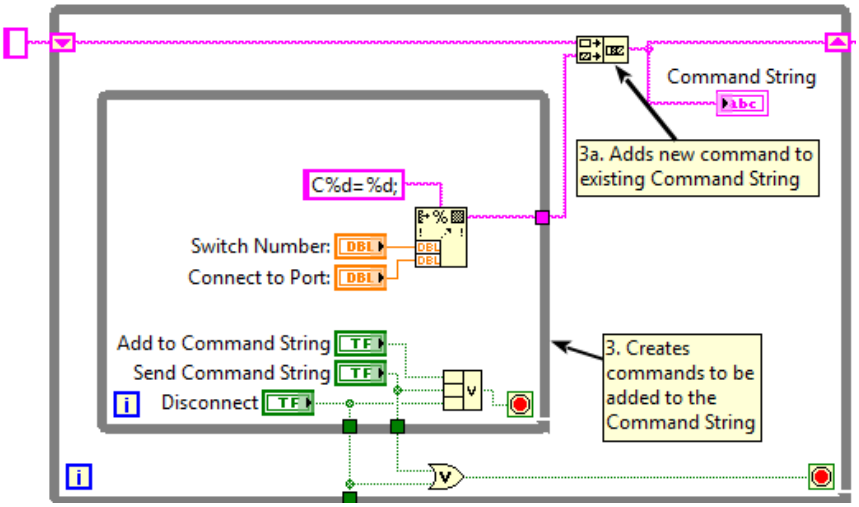
1. Connect to the .NET DLL
2. Connect to the hardware
3. Input the relevant switch commands
    a. Multiple commands can be sent at the same time (max 60 characters)
4. Send the command string to the hardware
5. Disconnect the hardware

2. Creating the VI Block Diagram

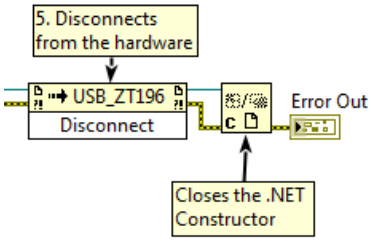
1. Create a Constructor Node from the Connectivity palette and select the ZT-196 DLL file on your PC as the Constructor
2. Place Invoke Nodes on the block diagram for each DLL function to call from the DLL, starting with the Connect function to initialize the hardware connection



3. The following VI section shows a method for creating the SCPI commands that control the individual switch states, based on the VI user's inputs.

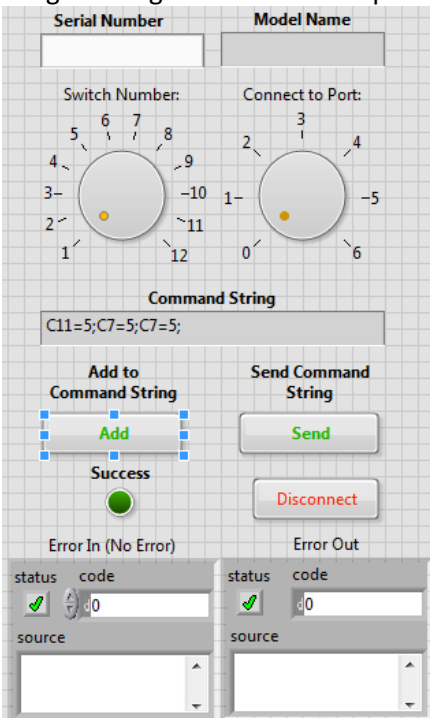


4. The final node in the block diagram should call the DLL's "Disconnect" function to close the hardware connection.



3. VI GUI

This LabVIEW VI creates the following GUI to give the user a simple method for setting switch states:



4. Full VI Block Diagram