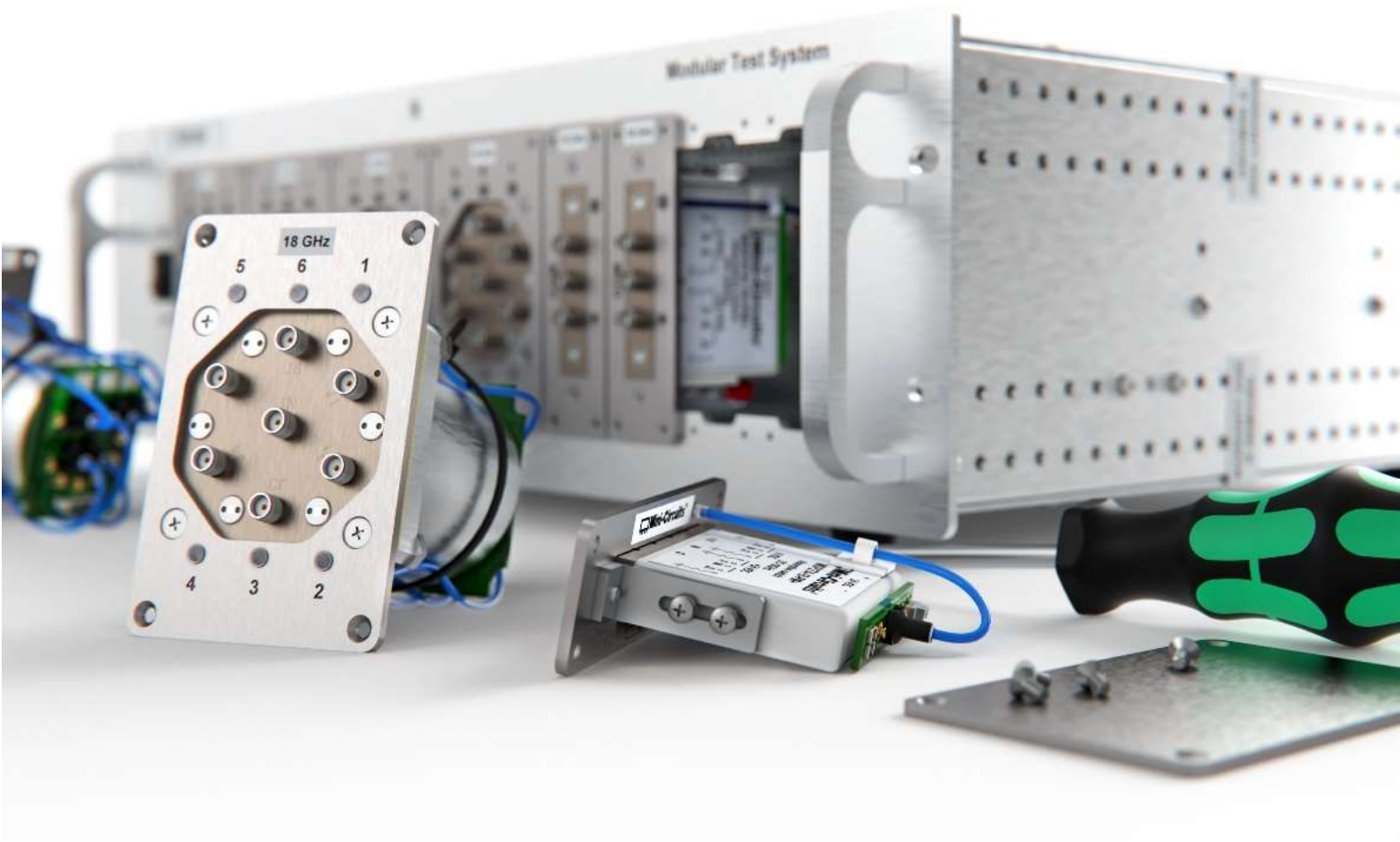
Programming Manual

MODULAR TEST SYSTEMS

ZTM Series | RCM Series



testsolutions@minicircuits.com | www.minicircuits.com

Contents

1. Overview

This programming manual is intended for customers wishing to create their own interface for Mini-Circuits' RCM & ZTM series, USB & Ethernet controlled, modular switch and attenuator systems.

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

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

For details and specifications on the individual models, please see:

1. RCM series: https://www.minicircuits.com/WebStore/rcm.html
2. ZTM series: https://www.minicircuits.com/WebStore/ztm.html

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

1.1. Control Methods

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

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

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

1.2. Programming Examples

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

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

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

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

1.3. Support Contacts

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

testsolutions@minicircuits.com

www.minicircuits.com/contact/worldwide_tech_support.html

2. Mini-Circuits' Modular Test System Concept

The ZTM & RCM Series modular concept is a flexible test system which can be configured to a user's requirements, with combinations of high reliability mechanical switches and programmable attenuators.

2.1. Addressing Individual Test Components

The internal controller is logically arranged with 6 zones for the ZTM Series and 3 zones for the RCM Series. The zones are addressed from 1 to n (left to right when looking at the front panel) and can each be specified with the following components:

- 1 or 2 high reliability, mechanical SPDT switches
- 1 or 2 high reliability, mechanical transfer / DPDT switches
- 1 high reliability, mechanical SP4T switch
- 1 high reliability, mechanical SP6T switch
- 1 high reliability, mechanical SP8T switch
- 1 amplifier
- 1 or 2 programmable attenuators

A range of passive splitter / combiner and coupler components can also be fitted to the ZTM modular panels but do not require any control and are therefore not addressed by the controller. Refer to the Mini-Circuits website or contact testsolutions@minicircuits.com to review the full list of components available, with options up to 50 GHz.

Components are addressed using the designators listed in table 1:

Component	Designator
SPDT Switch	SPDT
SP4T Switch	SP4T
SP6T Switch	SP6T
SP8T Switch	SP8T
Transfer / DPDT Switch	MTS
Amplifier	AMP
Programmable Attenuator	RUDAT

Table 1: ZTM / RCM Series Component Designators

Where 2 components are fitted within a single control zone, they are addressed with the zone number and either "A" or "B" appended to specify the left or right component respectively.

2.2. RCM Series Front Panel Layout

2.2.1. MODELS WITH SP8T SWITCHES

RCM models containing SP8T switches are supplied with 2 front panel windows and can only accommodate either 1 or 2 SP8T switches, addressed 1 and 2 (left to right).

Fig. 1 - RCM-2SP8T-26 with 2 front panel windows

The example of figure 2 includes the switch combination summarized and addressed in the table below:

Window Position	Address	Component
1	1	SP8T
2	2	SP8T

2.2.2. OTHER RCM MODELS

All other RCM models (those not containing SP8T switches) are supplied in a package with 3 fixed window positions on the front panel, addressed 1 to 3 (left to right). Where 2 components are fitted within a single window, they are addressed with either "A" or "B" appended to specify the left or right component respectively.

Fig. 2 - RCM-405 with 2 x SPDT switches (1A & 1B) and 2 x SP4T switches (2 & 3)

The example of figure 2 includes the switch combination summarized and addressed in the table below:

Window Position	Address	Component
1	1A	SPDT
	1B	SPDT
2	2	SP4T
3	3	SP4T

2.3. ZTM Series Front Panel Layout

2.3.1. FLEXIBLE WINDOW PANEL

The latest ZTM design features a flexible front panel design to accommodate switch components of varying widths, from SPDT components which occupy 1 slot, up to SP8T components which occupy 3 slots, for a total of 12 slots across the panel. The components are typically addressed 1 to n, from left to right, ignoring any passive components. There are cases where single slot components may be addressed with either "A" or "B" appended. It is important to refer to the individual model specification to confirm the addresses.

Fig. 3 - ZTM-172 featuring a flexible front panel configuration, to accommodate components of different widths

The example of figure 3 includes the switch combination summarized and addressed in the table below:

Address	Component
1	SP8T
2	SPDT
3	SP8T
4	SPDT
5	SP6T
6	SP6T

2.3.2. FIXED WINDOW PANEL

Certain component combinations (primarily those including programmable attenuators) require a front panel comprising 6 fixed width windows. The 6 windows are typically addressed 1 to 6, from left to right, ignoring any passive components. Where 2 components are fitted within a single window, they are addressed with either "A" or "B" appended, to specify the left or right component respectively. Refer to the individual model specification to confirm the addresses.

Fig. 4 - ZTM-95 featuring a fixed window front panel (6 equal sized windows)

The example of figure 4 includes the switch and attenuator combination summarized and addressed in the table below:

Window Position	Address	Component
1	1A	SPDT
	1B	SPDT
2	2	SP4T
3	3	SP4T
4	4	SP4T
5	5	SP4T
6	6A	RUDAT
	6B	RUDAT

3. SCPI Commands for Control of Modular Test Systems

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

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

3.1. SCPI – System Functions

Description	Command/Query
Identify System	*IDN?
Get Model Name	:MN?
Get Serial Number	:SN?
Get Firmware	:FIRMWARE?
Get Configuration	:CONFIG:APP?
Get Configuration & Switch States	:CONFIG:STATES?
Get Internal Temperature	:TS0?
Get Heat Alarm	:HEATALARM?
Save Counters & States	:OPERATIONDATA:SAVE
Set Fan Temperature Threshold	:FANS:Temp:Threshold:[set-point]:[value]
Get Fan Temperature Threshold	:FANS:Temp:Threshold:[set-point]?
Get Fan Status	FANSTATE?
Get Fan Alarm	FANALARM?

3.1.1. IDENTIFY SYSTEM

IDN?

Return the manufacturer, model name, serial number and firmware version.

Applies To

Firmware D4 or later

Return Values

`[manufacturer], [model name], [serial number], [firmware]`

Examples

String to Send	String Returned
*IDN?	Mini-Circuits,ZTM-999,12108100025,D4-0

HTTP Implementation: `http://10.10.10.10/*IDN?`

3.1.2. GET MODEL NAME

:MN?

Return the Mini-Circuits model name.

Return Value

`MN=[model]`

Value	Description
`[model]`	Model name of the connected device

Examples

String to Send	String Returned
`:MN?`	`MN=ZTM-999`

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

3.1.3. GET SERIAL NUMBER

:SN?

Returns the serial number.

Return Value

`SN=[serial]`

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

Examples

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

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

3.1.4. GET FIRMWARE

:FIRMWARE?

Returns the internal firmware version.

Return Value

Value	Description
[firmware]	The current firmware version, for example “B3”.

Examples

String to Send	String Returned
:FIRMWARE?	B3

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

3.1.5. GET CONFIGURATION

:CONFIG:APP?

Identifies the switch / programmable attenuator modules installed. Returns a list of integer codes (separated by semi-colons) indicating the component(s) mounted in each window of the system. The possible integer codes are:

Code	Description
0	Blank
1	SPDT (all models)
3	Dual SPDT
4	SP4T (18 GHz)
5	MTS (18 GHz)
7	Dual MTS (18 GHz)
8	Programmable attenuator
10	Dual programmable attenuators
20	Amplifier
11	SP6T (12-18 GHz models)
12	SP8T (all models)
13	SP6T (26.5-50 GHz models)
44	SP4T (26.5-50 GHz models)
55	MTS (26.5-40 GHz models)
57	Dual MTS (26.5-40 GHz models)

Return Values

`APP=[window1];[window2];…[windowN]`

Variable	Description
[window1-n]	Integer code indicating the installed component in each slot

Examples

String to Send	String Returned
:CONFIG:APP?	APP=4;7;4;44;57;20

The above return sequence would indicate the following components are installed and addressed as shown:

Address	1	2A & 2B	3	4	5A & 5B	6
Return	4	7	4	44	57	20
Component Type	SP4T (18 GHz)	Dual MTS (18 GHz)	SP4T (18 GHz)	SP4T (26.5-50 GHz)	Dual MTS (26.5-40 GHz)	Amplifier

HTTP Implementation: `http://10.10.10.10/:CONFIG:APP?`

3.1.6. GET CONFIGURATION & SWITCH STATES

:CONFIG:STATES?

Identifies the switch / programmable attenuator modules installed and the current state of each switch. Switch / attenuator types are indicated by an integer code as described in Get Configuration.

Applies To

Firmware D4 or later.

Return Values

`STA=[comp1]_ [state1]; [comp2]_ [state2];… [compN]_ [stateN]`

Variable	Description
`[comp1-n]`	Integer code indicating the component(s) installed in this window
`[state1-n]`	For switch components, an integer value indicating the state. For windows containing dual switches (eg: dual SPDT), the states of both switches will be listed, separated by commas. No value is returned for components other than switches.

Examples

String to Send	String Returned
`:CONFIG:STATES?`	`STA=4_0;3_1,1;11_5;7_0,1;10_`

The above return sequence would indicate the following components are installed and addressed as shown:

Address	1	2	3	4	5
Return	4_0	3_1,1	11_5	7_0,1	10_
Component Type	SP4T (18 GHz)	Dual SPDT	SP6T (12-18 GHz)	Dual MTS (18 GHz)	Dual Attenuators
Switch State	0	SWA = 1 SWB = 1	5	SWA = 0 SWB = 1	N/A

HTTP Implementation: `http://10.10.10.10/:CONFIG:STATES?`

3.1.7. GET INTERNAL TEMPERATURE

:TS0?

Returns the internal temperature measured at the control board.

Return Value

Value	Description
[temp]	The temperature reading of the specified internal sensor in degrees Celsius.

Examples

String to Send	String Returned
:TS0?	+37.25

HTTP Implementation: http://10.10.10.10/:TS0?

3.1.8. GET HEAT ALARM

:HEATALARM?

Returns an alarm notification if the internal temperature sensor exceeds the factory programmed limit (65°C.

Return Value

Value	Description
[status]	Integer value to indicate the heat alarm status: 0 – Device is within normal operating temperature limits 1 – Device temperature has exceeded the recommended limits

Examples

String to Send	String Returned
:HEATALARM?	0

HTTP Implementation: http://10.10.10.10/:HEATALARM?

3.1.9. SAVE COUNTERS & STATES

:OPERATIONDATA:SAVE

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

Requirements

Serial Number Range	Firmware Version
Up to 11412319999	A8 or later
From 11501010000	B2 or later

Return Values

Value	Description
`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`

HTTP Implementation: `http://10.10.10.10/:OPERATIONDATA:SAVE`

See Also

Set Start-Up Attenuation Mode

Set Switch Start-Up Mode

Get Switch Counter

3.1.10. SET FAN TEMPERATURE THRESHOLD

:FANS:Temp:Threshold:[set-point]:[value]

Refer to the individual model specification to determine whether fans are fitted. This function sets the temperature for the specified set-point, determining when the fans will activate and at which spin rate. The lower set-point defines the temperature threshold at which the fans will first turn on, spinning at 20% of max speed. The upper set-point must be a higher temperature value and sets the point at which the fans will increase to max RPM.

Applies To

Refer to individual model specifications to confirm whether fans are fitted

Parameters

Variable	Value	Description
`[set-point]`	`LOW`	The lower temperature set-point. Fans will spin at 20% of max speed above this temperature and will turn off below.
	`UP`	The upper temperature set-point. Fans will spin at max speed above this temperature but revert to the lower rate below.
`[value]`		The temperature value in degrees Celsius for the specified set-point. Note: The upper temperature set-point must be set to a higher value than the lower set-point.

Return Values

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

Examples

String to Send	String Returned
`:FANS:TEMP:THRESHOLD:LOW:25`	`1 - Success`
`:FANS:TEMP:THRESHOLD:UP:35`	`1 - Success`

HTTP Implementation: `http://10.10.10.10/:FANS:TEMP:THRESHOLD:LOW:25`

See Also

Get Internal Temperature

3.1.11. GET FAN TEMPERATURE THRESHOLD

:FANS:Temp:Threshold:[set-point]?

Refer to the individual model specification to determine whether fans are fitted. This function returns the temperature for the specified set-point, determining when the fans will activate and at which spin rate. The lower set-point is the temperature threshold at which the fans first turn on, spinning at 20% of max speed. The upper set-point must be a higher temperature value and is the point at which the fans will increase to max RPM.

Applies To

Refer to individual model specifications to confirm whether fans are fitted

Parameters

Variable	Value	Description
[set-point]	LOW	The lower temperature set-point. Fans will spin at 20% of max speed above this temperature and will turn off below.
	UP	The upper temperature set-point. Fans will spin at max speed above this temperature but revert to the lower rate below.

Return Values

Variable	Description
[value]	The temperature value in degrees Celsius for the specified set-point. Note: The upper temperature set-point must be set to a higher value than the lower set-point.

Examples

String to Send	String Returned
:FANS:TEMP:THRESHOLD:LOW?	25.00
:FANS:TEMP:THRESHOLD:UP?	25.00

HTTP Implementation: http://10.10.10.10/:FANS:TEMP:THRESHOLD:LOW?

See Also

Get Internal Temperature

3.1.12. GET FAN STATE

FANSTATE?

Refer to the individual model specification to determine whether fans are fitted. This function indicates whether the internal fan is currently spinning.

Applies To

Refer to individual model specifications to confirm whether fans are fitted

Return Value

Value	Description
0	Fan is not currently active / spinning
1	Fan is active / spinning

Examples

String to Send	String Returned
FANSTATE?	1

HTTP Implementation: http://10.10.10.10/FANSTATE?

See Also

Get Internal Temperature
Get Fan Temperature Threshold

3.1.13. GET FAN ALARM

FANALARM?

Refer to the individual model specification to determine whether fans are fitted. This function indicates a fault if the fan is unable to spin.

Applies To

Refer to individual model specifications to confirm whether fans are fitted

Return Value

Value	Description
0	No fault reported
>1	Potential fault; check the vent for obstructions

Examples

String to Send	String Returned
FANALARM?	0

HTTP Implementation: http://10.10.10.10/FANALARM?

See Also

Get Internal Temperature
Get Fan Temperature Threshold

3.2. SCPI - Programmable Attenuator Control

Description	Command / Query
Set Attenuation	:RUDAT:[address]:ATT:[value]
Get Attenuation	:RUDAT:[address]:ATT?
Set Start-Up Attenuation Mode	:RUDAT:[address]:STARTUPATT:INDICATOR:[mode]
Get Start-Up Attenuation Mode	:RUDAT:[address]:STARTUPATT:INDICATOR?
Set Start-Up Attenuation Value	:RUDAT:[address]:STARTUPATT:VALUE:[value]
Get Start-Up Attenuation Value	:RUDAT:[address]:STARTUPATT:VALUE?
Get Maximum Attenuation	:RUDAT:[address]:MAX?

3.2.1. SET ATTENUATION

:RUDAT:[address]:ATT:[value]

Set the attenuation level of the specified programmable attenuator.

Parameters

Variable	Description
[value]	The attenuation to set in dB (up to 2 decimal places).

Return Values

Value	Description
0 - Failed	Command failed (attenuation not set)
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:RUDAT:1A:ATT:70.25	1 - Success

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:ATT:70.25

3.2.2. GET ATTENUATION

:RUDAT:[address]:ATT?

Read the attenuation of a specific programmable attenuator.

Return Values

Variable	Description
[value]	The attenuation setting (in dB)

Examples

String to Send	String Returned
:RUDAT:1A:ATT?	70.25

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:ATT?

3.2.3. SET START-UP ATTENUATION MODE

:RUDAT:[address]:STARTUPATT:INDICATOR:[mode]

Set the start-up mode of a specific attenuator. The attenuator can be configured to load the last saved value, a specific fixed value or the maximum possible attenuation.

Parameters

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last saved attenuation. See Save Counters & States for correct operation.
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (default setting)

Return Values

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

Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:INDICATOR:L	1 - Success
:RUDAT:2:STARTUPATT:INDICATOR:N	1 - Success

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:STARTUPATT:INDICATOR:L

See Also

Save Counters & States

Set Start-Up Attenuation Value

3.2.4. GET START-UP ATTENUATION MODE

:RUDAT:[address]:STARTUPATT:INDICATOR?

Return the start-up mode of a specific attenuator. The attenuator can be configured to load the last saved value, a specific fixed value or the maximum possible attenuation.

Return Values

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last saved attenuation. See Save Counters & States for correct operation.
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (default setting)

Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:INDICATOR?	L
:RUDAT:2:STARTUPATT:INDICATOR?	N

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:STARTUPATT:INDICATOR?

See Also

Get Start-Up Attenuation Value

3.2.5. SET START-UP ATTENUATION VALUE

:RUDAT:[address]:STARTUPATT:VALUE:[att]

Used in conjunction with Set Start-Up Attenuation Mode to set the initial attenuation on power-up for a specific attenuator.

Parameters

Variable	Description
[att]	The initial attenuation to set on power-up (only valid when Start-Up Attenuation Mode is set to mode "F" (fixed value).

Return Values

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

Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:VALUE:15	1 - Success
:RUDAT:2:STARTUPATT:VALUE:20.25	1 - Success

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:STARTUPATT:VALUE:15

See Also

Set Start-Up Attenuation Mode

3.2.6. GET START-UP ATTENUATION VALUE

:RUDAT:[address]:STARTUPATT:VALUE?

Returns the initial attenuation value on power-up for a specific attenuator.

Return Values

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last saved attenuation
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (default setting)

Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:VALUE?	15.00
:RUDAT:2:STARTUPATT:VALUE?	20.25

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:STARTUPATT:VALUE?

See Also

Get Start-Up Attenuation Mode

3.2.7. GET MAXIMUM ATTENUATION

:RUDAT:[address]::MAX?

Check the maximum possible attenuation setting.

Return Values

Variable	Description
[value]	The maximum attenuation specification (dB)

Examples

String to Send	String Returned
:RUDAT:1A:MAX?	95.00

HTTP Implementation: http://10.10.10.10/:RUDAT:1A:MAX?

3.3. SCPI - Switch Control

Description	Command / Query
Set Switch State	:[switch_type]:[address]:STATE:[value]
Get Switch State	:[switch_type]:[address]:STATE?
Set States of All Switches of Same Type	:[switch_type]:ALL:STATE:[values]
Get States of All Switch of Same Type	:[switch_type]:ALL:STATE?
Set Switch Start-Up Mode	:[switch_type]:[address]:STARTUPSW:INDICATOR:[mode]
Get Switch Start-Up Mode	:[switch_type]:[address]:STARTUPSW:INDICATOR?
Get Switch Counter	:[switch_type]:[address]:SCOUNTER?

3.3.1. SET SWITCH STATE

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

Set a single switch state.

Parameters

[switch_type]	[value]	Description
MTS	1	18 GHz switches: J1 <> J3 & J2 <> J4 26.5-40 GHz switches: J1 <> J2 & J3 <> J4
	2	18 GHz switches: J1 <> J2 & J3 <> J4 26.5-40 GHz switches: J1 <> J3 & J2 <> J4
SPDT	1 to 2	COM <> 1 to COM <> 2
SP4T	0	All ports disconnected
	1 to 4	COM<>1 to COM <> 4
SP6T	0	All ports disconnected
	1 to 6	COM<>1 to COM <> 6
SP8T	0	All ports disconnected
	1 to 8	COM<>1 to COM <> 8

Return Values

Value	Description
0 - Failed	Command failed (switch not set)
1 - Success	Command completed successfully

Examples

String to Send	String Returned
:SPDT:1A:STATE:2	1 - Success
:SPDT:2:STATE:1	1 - Success
:SP4T:2:STATE:0	1 – Success

HTTP Implementation: http://10.10.10.10/:SPDT:1A:STATE:2

3.3.2. GET SWITCH STATE

:[switch_type]:[address]:STATE?

Read the state of a specific switch.

Parameters

Variable	Value	Description
[switch_type]	MTS	Transfer / DPDT switch
	SPDT	SPDT switch
	SP4T	SP4T switch
	SP6T	SP6T switch
	SP8T	SP8T switch

Return Values

Switch Type	[value]	Description
MTS	1	18 GHz switches: J1 <> J3 & J2 <> J4 26.5-40 GHz switches: J1 <> J2 & J3 <> J4
	2	18 GHz switches: J1 <> J2 & J3 <> J4 26.5-40 GHz switches: J1 <> J3 & J2 <> J4
SPDT	1 to 2	COM <> 1 to COM <> 2
SP4T	0	All ports disconnected
	1 to 4	COM<>1 to COM <> 4
SP6T	0	All ports disconnected
	1 to 6	COM<>1 to COM <> 6
SP8T	0	All ports disconnected
	1 to 8	COM<>1 to COM <> 8

Examples

String to Send	String Returned
:SPDT:1A:STATE?	2
:SPDT:2:STATE?	1
[Invalid switch position]	-99 Unrecognized Query

HTTP Implementation: http://10.10.10.10/:SPDT:1A:STATE?

3.3.3. SET STATES OF ALL SWITCHES OF SAME TYPE

:[switch_type]:ALL:STATE:[state_list]

Set all specified switches of the same type to the same switch state, for example all SPDT switches within the system.

The switch states are specified as a single continuous string of up to 12 characters (depending on the type of system and installed switch configuration), 1 character per switch state. The string is specified starting with location 1 (or 1A for a dual SPDT) but digits on the right-hand side of the string can be omitted if the respective switch states are to be left unchanged or if the location does not contain a switch of the relevant type.

Each switch state is specified with a single digit from 0 to n (model dependent), as per Set Switch State. The character x is used where a switch is to be left unchanged or is not of the specified switch type.

Parameters

Variable	Description
[switch_type]	The single switch type that applies to each of the switches to set.
P[state_list]	String of switch states with each character from left to right representing a single switch from address 1 to n. Each character can be 0 to n to specify the state of an individual switch, or x where the switch is to be left unchanged or is of a different switch type.

Return Values

Value	Description
0 - Failed	Command failed (switches not set)
1 - Success	Command completed successfully

Example 1

Set SPDT switches 2A, 2B & 4 to states 2, 1 & 2 respectively. Any other switches are left unchanged.

String to Send	String Returned
:SPDT:ALL:STATE:xx21xx2	1 - Success

HTTP Implementation: http://10.10.10.10/:SPDT:ALL:STATE:xx21xx21

Example 2

Set SP4T switches 1 & 3 to state 4. Any other switches are left unchanged.

String to Send	String Returned
:SP4T:ALL:STATE:4x4	1 - Success

HTTP Implementation: http://10.10.10.10/:SP4T:ALL:STATE:4x4

3.3.4. GET STATES OF ALL SWITCH OF SAME TYPE

:[switch_type]:ALL:STATE?

Returns the state of all switches of the same type, for example all SPDT switches within the system.

The switch states are indicated as a single continuous string of up to 12 characters (depending on the type of system and installed switch configuration), 1 character per switch state, starting with location 1 (or 1A for a dual SPDT).

Each switch state is indicated with a single digit from 0 to n (model dependent), as per Get Switch State. The character x is used where the position is occupied by a switch of a different type.

Parameters

Variable	Description
[switch_type]	The single switch type to check

Return Values

Variable	Description
[state_list]	String of switch states with each character from left to right representing a single switch from address 1 to n. Each character can be 0 to n to specify the state of an individual switch, or x where the switch is of a different switch type.

Example 1

The response below indicates there are SPDT switches fitted in positions 2A, 2B & 4A, with states 2, 1 & 2 respectively.

String to Send	String Returned
:SPDT:ALL:STATE?	xx21xx2xxxxx

HTTP Implementation: http://10.10.10.10/:SPDT:ALL:STATE?

Example 2

The response below indicates there are SP4T switches fitted in positions 1 & 3, both set to state 4.

String to Send	String Returned
:SP4T:ALL:STATE?	4x4xxx

HTTP Implementation: http://10.10.10.10/:SP4T:ALL:STATE?

3.3.5. SET SWITCH START-UP MODE

:[switch_type]:[address]:STARTUPSW:INDICATOR:[mode]

Set the start-up state for a specific switch when the system is powered up.

Parameters

Variable	Value	Description
[switch_type]	MTS	Transfer / DPDT switch
	SPDT	SPDT switch
	SP4T	SP4T switch
	SP6T	SP6T switch
	SP8T	SP8T switch
[mode]	L	Last Value – The switch will power up with the last saved switch state. See Save Counters & States for correct operation.
	N	Normal – The switch will power up in the default state: • SPDT: Com port connected to port 1 • SP4T / SP6T / SP8T: All ports disconnected • Transfer (18 GHz): J1 <> J3; J2 <> J4 • Transfer (26.5-40 GHz): J1 <> J2; J3 <> J4

Return Values

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

Example

String to Send	String Returned
:SP6T:1:STARTUPSW:INDICATOR:L	1 - Success

HTTP Implementation: http://10.10.10.10/:SP6T:1:STARTUPSW:INDICATOR:L

See Also

Save Counters & States

3.3.6. GET SWITCH START-UP MODE

:[switch_type]:[address]:STARTUPSW:INDICATOR?

Returns the start-up state for a specific switch when the system is powered up.

Parameters

Variable	Value	Description
[switch_type]	MTS	Transfer / DPDT switch
	SPDT	SPDT switch
	SP4T	SP4T switch
	SP6T	SP6T switch
	SP8T	SP8T switch

Return Values

Value	Description
L	Last Value – The switch will power up with the last saved switch state.
N	Normal – The switch will power up in the default state: • SPDT: Com port connected to port 1 • SP4T / SP6T / SP8T: All ports disconnected • Transfer (18 GHz): J1 <> J3; J2 <> J4 • Transfer (26.5-40 GHz): J1 <> J2; J3 <> J4

Examples

String to Send	String Returned
:SP6T:1:STARTUPATT:INDICATOR?	L

HTTP Implementation: http://10.10.10.10/:SP6T:1:STARTUPSW:INDICATOR?

3.3.7. GET SWITCH COUNTER

:[switch_type]:[address]:SCOUNTER?

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

Note: See Save Counters & States for correct operation.

Parameters

Variable	Value	Description
[switch_type]	MTS	Transfer / DPDT switch
	SPDT	SPDT switch
	SP4T	SP4T switch
	SP6T	SP6T switch
	SP8T	SP8T switch

Return Values

Variable	Description

[count]	• SPDT & MTS models:
	  ○ A single count of the total number of switch cycles undertaken by the specified switch
	• SP4T to SP8T models:
	  ○ A list of counts, each separate by a semi-colon of the total number of cycles to each port within the specified switch

Examples

String to Send	String Returned
:SPDT:1A:SCOUNTER?	9540
:SP6T:1:SCOUNTER?	195;452;300;125;850;647

HTTP Implementation: http://10.10.10.10/:SPDT:1A:SCOUNTER?

http://10.10.10.10/:SP6T:1:SCOUNTER?

See Also

Save Counters & States

3.4. SCPI - Amplifier Control

Description	Command / Query
Set Amplifier State	:AMP:[address]:STATE:[value]
Get Amplifier State	:AMP:[address]:STATE?

3.4.1. SET AMPLIFIER STATE

:AMP:[address]:STATE:[value]

Turn an amplifier on or off (connect or disconnect the DC voltage supply).

Applies To

Firmware D9 or later

Parameters

Variable	Value	Description
[value]	0	Amplifier powered off
	1	Amplifier powered on

Return Values

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

Examples

String to Send	String Returned
:AMP:1:STATE:1	1 - Success

HTTP Implementation: http://10.10.10.10/:AMP:1:STATE:1

3.4.2. GET AMPLIFIER STATE

:AMP:[address]:STATE?

Read the state of an amplifier (powered on or off).

Applies To

Firmware D9 or later

Return Values

Value	Description
0	Amplifier powered off
1	Amplifier powered on

Examples

String to Send	String Returned
:AMP:1:STATE?	1

HTTP Implementation: http://10.10.10.10/:AMP:1:STATE?

3.5. SCPI - Component Labels

Description	Command / Query
Set Component Label	:LABEL:[address]:"[text]"
Get Component Label	:LABEL:[address]?

3.5.1. SET COMPONENT LABEL

:LABEL:[address]:"[text]"

Set a custom label for easy identification of a specific component.

Parameters

Variable	Description
[text]	Up to 24 ASCII characters to provide a user-friendly label for the component

Return Values

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

Example

String to Send	String Returned
:LABEL:2:"Input_SP4T_1"	1 - Success

HTTP Implementation: http://10.10.10.10/:LABEL:1A:"Input_SPDT_1"

3.5.2. GET COMPONENT LABEL

:LABEL:[address]?

Get the custom label for a specific component (used for easy identification).

Return Values

LABEL="[text]"

Variable	Description
[text]	The component label text

Examples

String to Send	String Returned
:LABEL:2?	LABEL="Input_SP4T_1"

HTTP Implementation: http://10.10.10.10/:LABEL:1A?

3.6. SCPI - Ethernet Configuration Commands

These functions provide a method of configuring the device's Ethernet IP settings and can be sent using either the USB or Ethernet connections. Refer to Ethernet Control API for additional details on the Ethernet configuration and default behavior.

These commands require firmware C2 or later to be installed.

Function	Syntax
Get Current Ethernet Configuration	:ETHERNET:CONFIG:LISTEN?
Get MAC Address	:ETHERNET:CONFIG:MAC?
Get DHCP Status	:ETHERNET:CONFIG:DHCPENABLED?
Use DHCP	:ETHERNET:CONFIG:DHCPENABLED:[enabled]
Get Static IP Address	:ETHERNET:CONFIG:IP?
Set Static IP Address	:ETHERNET:CONFIG:IP:[ip]
Get Static Network Gateway	:ETHERNET:CONFIG:NG?
Set Static Network Gateway	:ETHERNET:CONFIG:NG:[gateway]
Get Static Subnet Mask	:ETHERNET:CONFIG:SM?
Set Static Subnet Mask	:ETHERNET:CONFIG:SM:[mask]
Get HTTP Port	:ETHERNET:CONFIG:HTPORT?
Set HTTP Port & Enable / Disable HTTP	:ETHERNET:CONFIG:HTPORT:[port]
Get Telnet Port	:ETHERNET:CONFIG:TELNETPORT?
Set Telnet Port & Enable / Disable Telnet	:ETHERNET:CONFIG:TELNETPORT:[port]
Get Password Requirement	:ETHERNET:CONFIG:PWDENABLED?
Set Password Requirement	:ETHERNET:CONFIG:PWDENABLED:[enabled]
Get Password	:ETHERNET:CONFIG:PWD?
Set Password	:ETHERNET:CONFIG:PWD:[pwd]
Update Ethernet Settings	:ETHERNET:CONFIG:INIT

3.6.1. GET CURRENT ETHERNET CONFIGURATION

:ETHERNET:CONFIG:LISTEN?

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

Applies To

Firmware C2 or later

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?`

3.6.2. GET MAC ADDRESS

:ETHERNET:CONFIG:MAC?

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

Applies To

Firmware C2 or later

Return String

Variable	Description
`[mac]`	MAC address

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?`

3.6.3. GET DHCP STATUS

:ETHERNET:CONFIG:DHCPENABLED?

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

Applies To

Firmware C2 or later

Return String

Value	Description
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

Get Current Ethernet Configuration

3.6.4. USE DHCP

:ETHERNET:CONFIG:DHCPENABLED:[enabled]

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

Applies To

Firmware C2 or later

Parameters

Value	Description
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

Value	Description
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

Update Ethernet Settings

3.6.5. GET STATIC IP ADDRESS

:ETHERNET:CONFIG:IP?

Returns the user-entered static IP address.

Applies To

Firmware C2 or later

Return String

Variable	Description
[ip]	String containing the static IP address

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

Get DHCP Status
Get Current Ethernet Configuration

3.6.6. SET STATIC IP ADDRESS

:ETHERNET:CONFIG:IP:[ip]

Sets the static IP address to be used when DHCP is disabled. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Applies To

Firmware C2 or later

Parameters

Variable	Description
[ip]	String containing the static IP address

Return String

Value	Description
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

Use DHCP
Update Ethernet Settings

3.6.7. GET STATIC NETWORK GATEWAY

:ETHERNET:CONFIG:NG?

Returns the user-entered network gateway IP address.

Applies To

Firmware C2 or later

Return String

Variable	Description
[gateway]	String containing the 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 DHCP Status
Get Current Ethernet Configuration

3.6.8. SET STATIC NETWORK GATEWAY

:ETHERNET:CONFIG:NG:[gateway]

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

Applies To

Firmware C2 or later

Parameters

Variable	Description
[gateway]	String containing IP address of the network gateway

Return String

Value	Description
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

Use DHCP
Update Ethernet Settings

3.6.9. GET STATIC SUBNET MASK

:ETHERNET:CONFIG:SM?

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.

Applies To

Firmware C2 or later

Return String

Variable	Description
[mask]	String containing the subnet mask

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 DHCP Status
Get Current Ethernet Configuration

3.6.10. SET STATIC SUBNET MASK

:ETHERNET:CONFIG:SM:[mask]

Sets the subnet mask to be used when DHCP is disabled. Changes to the Ethernet configuration only take effect after the Update Ethernet Settings command has been issued.

Applies To

Firmware C2 or later

Parameters

Variable	Description
[mask]	String containing the subnet mask

Return String

Value	Description
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

Use DHCP
Update Ethernet Settings

3.6.11. GET HTTP PORT

:ETHERNET:CONFIG:HTPORT?

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

Applies To

Firmware C2 or later

Return String

Variable	Description
[port]	TCP / IP port to be used for HTTP communication. A value of 0 or 65535 indicates that Telnet communication has been disabled (requires firmware D7 or later).

Examples

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

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

3.6.12. SET HTTP PORT & ENABLE / DISABLE HTTP

:ETHERNET:CONFIG:HTPORT:[port]

Sets the TCP / IP port to be used for HTTP communication. The default is port 80. Set port 0 or 65535 to disable HTTP (requires firmware D7 or later) or any valid port to enable.

Applies To

Firmware C2 or later

Parameters

Variable	Description
[port]	TCP / IP port to be used for HTTP communication. To disable HTTP communication, specify port 0 or 65535 (requires firmware D7 or later).

Return String

Value	Description
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

Update Ethernet Settings

3.6.13. GET TELNET PORT

:ETHERNET:CONFIG:TELNETPORT?

Returns the TCP/IP port in use for Telnet communication. The default is port 23.

Applies To

Firmware C2 or later

Return String

Variable	Description
[port]	TCP / IP port to be used for Telnet communication. A value of 0 or 65535 indicates that Telnet communication has been disabled (requires firmware D7 or later).

Examples

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

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

3.6.14. SET TELNET PORT & ENABLE / DISABLE TELNET

:ETHERNET:CONFIG:TELNETPORT:[port]

Sets the TCP / IP port to be used for Telnet communication. The default is port 23. Set port 0 or 65535 to disable Telnet (requires firmware D7 or later) or any valid port to enable.

Applies To

Firmware C2 or later

Parameters

Variable	Description
[port]	TCP / IP port to be used for Telnet communication. To disable Telnet communication, specify port 0 or 65535 (requires firmware D7 or later).

Return String

Value	Description
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

Update Ethernet Settings

3.6.15. GET PASSWORD REQUIREMENT

:ETHERNET:CONFIG:PWDENABLED?

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

Applies To

Firmware C2 or later

Return String

Value	Description
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

Get Password

3.6.16. SET PASSWORD REQUIREMENT

:ETHERNET:CONFIG:PWDENABLED:[enabled]

Enables or disables the password requirement for HTTP / Telnet.

Applies To

Firmware C2 or later

Parameters

Value	Description
0	Password not required for Ethernet communication
1	Password required for Ethernet communication

Return String

Value	Description
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

Set Password
Update Ethernet Settings

3.6.17. GET PASSWORD

:ETHERNET:CONFIG:PWD?

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

Applies To

Firmware C2 or later

Return String

Variable	Description
[pwd]	Password for Ethernet communication

Examples

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

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

See Also

Get Password Requirement

3.6.18. SET PASSWORD

:ETHERNET:CONFIG:PWD:[pwd]

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

Applies To

Firmware C2 or later

Parameters

Variable	Description
[pwd]	Password for Ethernet communication (not case sensitive, 20 characters maximum)

Return String

Value	Description
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
Update Ethernet Settings

3.6.19. UPDATE ETHERNET SETTINGS

:ETHERNET:CONFIG:INIT

Resets the Ethernet controller and restarts with the latest saved settings. Any subsequent commands / queries to the device will need to be attempted using the new Ethernet configuration. If a connection cannot be established, 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.

Return String

Value	Description
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

4. USB Control API for Microsoft Windows

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

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

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

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

4.1. DLL API Options

4.1.1. .NET FRAMEWORK 4.5 DLL (RECOMMENDED)

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

Filename: ModularZT_NET45.dll

Requirements

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

Installation

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

4.1.2. .NET FRAMEWORK 2.0 DLL (LEGACY SUPPORT)

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

Filename: ModularZT64.dll

Requirements

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

Installation

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

4.1.3. ACTIVEX COM OBJECT DLL (LEGACY SUPPORT)

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

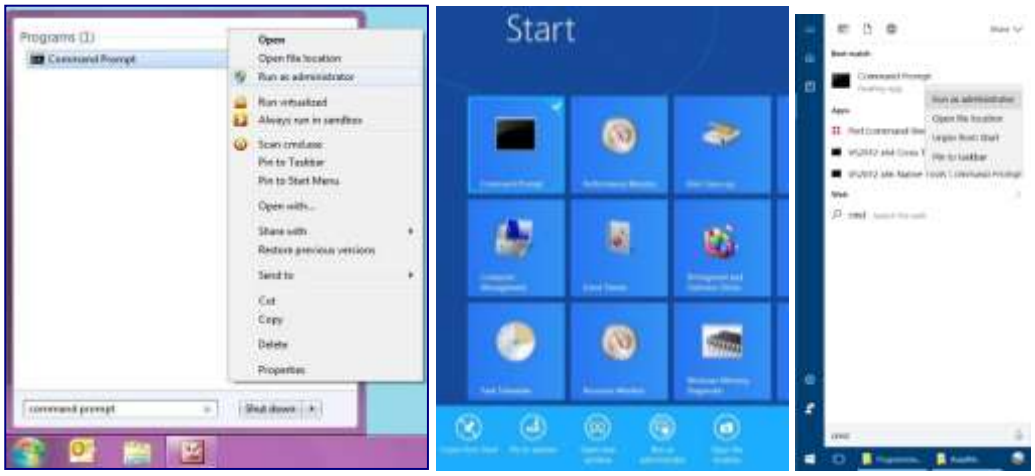
Filename: ModularZT.dll

Requirements

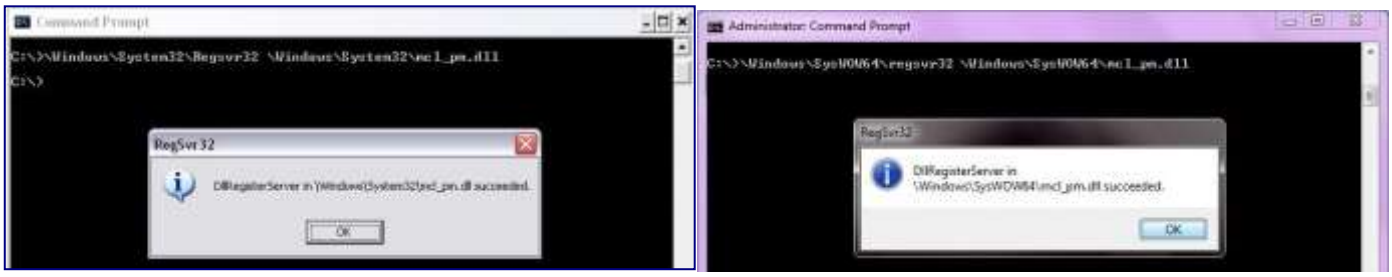
1) Microsoft Windows
2) Programming environment with support for ActiveX components

Installation

1) Copy the DLL file to the correct directory:
   For 32-bit Windows operating systems: `C:\WINDOWS\System32`
   For 64-bit Windows operating systems: `C:\WINDOWS\SysWOW64`
2) Open the Command Prompt in "Elevated" mode:
   a. Open the Start Menu/Start Screen and type "Command Prompt"
   b. Right-click on the shortcut for the Command Prompt
   c. Select "Run as Administrator"
   d. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3) Use regsvr32 to register the DLL:
4) 32-bit PC:
   `Regsvr32 ModularZT.dll`
5) 64-bit PC:
   `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\ModularZT.dll`
6) Register the DLL using regsvr32:
7) Hit enter to confirm and a message box will appear to advise of successful registration



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



Registering the DLL

4.2. Referencing the DLL

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

Example Declarations Using the .NET 4.5 DLL (ModularZT_NET45.dll)

(For operation with the .Net 2.0 DLL, replace "ModularZT_NET45.dll" with "ModularZT64.dll")

Python	`import clr  # Import the pythonnet CLR library`
	`clr.AddReference('ModularZT_NET45')`
	`from ModularZT_NET45 import USB_ZT`
	`MyPTE1 = USB_ZT()`
	`MyPTE2 = USB_ZT()`

Visual Basic	`Public MyPTE1 As New ModularZT_NET45.USB_ZT` `Public MyPTE2 As New ModularZT_NET45.USB_ZT`
Visual C++	`ModularZT_NET45::USB_ZT ^MyPTE1 = gcnew ModularZT_NET45::USB_ZT();` `ModularZT_NET45::USB_ZT ^MyPTE2 = gcnew ModularZT_NET45::USB_ZT();`
Visual C#	`ModularZT_NET45.USB_ZT MyPTE1 = new ModularZT_NET45.USB_ZT();` `ModularZT_NET45.USB_ZT MyPTE2 = new ModularZT_NET45.USB_ZT();`

MatLab	`MCL_SW = NET.addAssembly('C:\Windows\SysWOW64\ModularZT_NET45.dll')`
	`MyPTE1 = ModularZT_NET45.USB_ZT`
	`MyPTE2 = ModularZT_NET45.USB_ZT`

Example Declarations using the ActiveX DLL (ModularZT.dll)

Visual Basic	`Public MyPTE1 As New ModularZT.USB_Control` `Public MyPTE2 As New ModularZT.USB_Control`
Visual C++	`ModularZT::USB_Control ^MyPTE1 = gcnew ModularZT::USB_Control();` `ModularZT::USB_Control ^MyPTE2 = gcnew ModularZT::USB_Control();`
Visual C#	`public ModularZT.USB_Control MyPTE1 = new ModularZT.USB_Control();` `public ModularZT.USB_Control MyPTE2 = new ModularZT.USB_Control();`
MatLab	`MyPTE1 = actxserver(ModularZT.USB_Control')` `MyPTE2 = actxserver(ModularZT.USB_Control')`

4.3. Additional DLL Considerations

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

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

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

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

Definition	`int Read_SN(ByRef string SN)`

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

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

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

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

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

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

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

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

Mini-Circuits' DLL Definition	`short Connect(string [SN])`

Implementation in CVI instrument driver	`int CVIFUNC ModularZT_NET45_USB_ZT_Open_Sensor(`
	`  ModularZT_NET45_USB_ZT __instance,`
	`  char ** SN,`
	`  short * __returnValue,`
	`  CDotNetHandle * __exception);`

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

4.4. DLL - System Functions

Function	Syntax
Connect	Short Connect(Optional String SN)
Connect by Address	Short ConnectByAddress(Optional Short Address)
Disconnect	Void Disconnect()
Send SCPI Command	Short Send_SCPI(ByRef String SndSTR, ByRef String RetSTR)
Read Model Name	Short Read_ModelName(String ModelName)
Read Serial Number	Short Read_SN(String SN)
Set Address	Short Set_Address(Short Address)
Get Address	Short Get_Address()
Get List of Connected Serial Numbers	Short Get_Available_SN_List(ByRef String SN_List)
Get List of Available Addresses	Short Get_Available_Address_List(ByRef String Add_List)
Get Software Connection Status	Short GetConnectionStatus()
Get USB Connection Status	Short GetUSBConnectionStatus()
Get Firmware	Short GetExtFirmware(ByRef Short A0, ByRef Short A1, By Ref Short A2, By Ref Short A3, By Ref String Firmware)
Get Firmware Version (Antiquated)	Short GetFirmware()

4.4.1. CONNECT

Short Connect(Optional String SN)

Initializes the USB connection. The serial number should be included if multiple devices are connected. The device should be disconnected on completion of the program using the Disconnect function.

Parameters

Variable	Description
SN	Serial number for the specific device to connect (if omitted the first device found on the bus will be connected).

Return Values

Value	Description
0	No connection was possible
>0	Connection successfully established

Examples

Python	response = MyPTE1.Connect() status = response[0]
Visual Basic	status = MyPTE1.Connect(SN)
Visual C++	status = MyPTE1->Connect(SN);
Visual C#	status = MyPTE1.Connect(ref(SN));
MatLab	status = MyPTE1.Connect(SN);

See Also

Disconnect

4.4.2. CONNECT BY ADDRESS

Short ConnectByAddress(Optional Short Address)

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

Parameters

Variable	Description
Address	The USB address of the device to connect (if omitted the first device found on the bus will be connected).

Return Values

Value	Description
0	No connection was possible
>0	Connection successfully established

Examples

Python	`response = MyPTE1.ConnectByAddress(Address)` `status = response[0]`
Visual Basic	`status = MyPTE1.ConnectByAddress(Address)`
Visual C++	`status = MyPTE1->ConnectByAddress(Address);`
Visual C#	`status = MyPTE1.ConnectByAddress(ref(Address));`
MatLab	`[status, Address] = MyPTE1.ConnectByAddress(Address);`

See Also

Connect
Disconnect
Get List of Available Addresses

4.4.3. DISCONNECT

Void Disconnect()

Terminates the connection. It is strongly recommended that this function is used prior to ending the program. Failure to do so may result in a connection failure on the next attempt, requiring a power cycle to rectify.

Examples

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

See Also

Connect

4.4.4. SEND SCPI COMMAND

Short Send_SCPI(ByRef String SndSTR, ByRef String RetSTR)

Sends a SCPI command to the system and collects the response. 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 modular test system's internal test components.

Parameters

Variable	Description
SndSTR	The SCPI command to send
RetSTR	String which will be updated with the value returned from the system

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.Send_SCPI(":MN?", "")` `response = str(status[2])`
Visual Basic	`status = MyPTE1.Send_SCPI(":MN?", response)`
Visual C++	`status = MyPTE1->Send_SCPI(":MN?", response);`
Visual C#	`status = MyPTE1.Send_SCPI(":MN?", ref(response));`
MatLab	`[status, command, response] = MyPTE1.Send_SCPI(":MN?", '')`

See Also

SCPI Commands for Control of Modular Test Systems

4.4.5. READ MODEL NAME

Short Read_ModelName(String ModelName)

Returns the Mini-Circuits part number.

Parameters

Variable	Description
ModelName	Required. A string variable that will be updated with the Mini-Circuits part number.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

```
status = MyPTE1.Read_ModelName("")
if status[0] > 0:
        ModelName = str(status[1])
        print('The connected device is ', ModelName)
```

Visual Basic

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
        MsgBox ("The connected device is " & ModelName)
End If
```

Visual C++

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
        MessageBox::Show("The connected device is " + ModelName);
}
```

Visual C#

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
        MessageBox.Show("The connected device is " + ModelName);
}
```

MatLab

```
[status, ModelName] = MyPTE1.Read_ModelName('')
if status > 0
        h = msgbox('The connected device is ', ModelName)
end
```

4.4.6. READ SERIAL NUMBER

Short Read_SN(String SN)

Returns the serial number.

Parameters

Variable	Description
ModelName	Required. String variable that will be updated with the serial number.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

Visual Basic	`If MyPTE1.Read_SN(SN) > 0 Then`
	`  MsgBox ("The connected device is " & SN)`
	`End If`

Visual C++	`if (MyPTE1->Read_SN(SN) > 0 )`
	`{`
	`  MessageBox::Show("The connected device is " + SN);`
	`}`

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

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

4.4.7. SET ADDRESS

Short Set_Address(Short Address)

Sets the internal USB address which can be used in place of the serial number for future connections. The factory default for all units is 255.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Example

Python	`status = MyPTE1.Set_Address(1)`
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

Connect by Address
Get List of Available Addresses

4.4.8. GET ADDRESS

Short Get_Address()

Returns the internal USB address which can be used to connect instead of serial number. The factory default for all units is 255.

Return Values

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

Examples

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

See Also

Get List of Available Addresses

4.4.9. GET LIST OF CONNECTED SERIAL NUMBERS

Short Get_Available_SN_List(ByRef String SN_List)

Provides a list of serial numbers for all available devices.

Parameters

Variable	Description
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

Value	Description
0	Command failed
1	Command completed successfully

Example

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

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

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

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

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

See Also

Connect

4.4.10. GET LIST OF AVAILABLE ADDRESSES

Short Get_Available_Address_List(ByRef String Add_List)

Provides a list of addresses for all available devices.

Parameters

Variable	Description
Add_List	Required. String variable which will be updated with a list of addresses separated by a single space character, for example, "5 101 254 255"

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Example

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

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

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

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

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

See Also

Connect by Address
Get List of Connected Serial Numbers

4.4.11. GET SOFTWARE CONNECTION STATUS

Short GetConnectionStatus()

Confirms whether the USB connection to the device is active. This will be true if the Connect function (or similar) has previously been called.

Return Values

Value	Description
0	No connection
1	Switch matrix is connected

Examples

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

See Also

Connect

4.4.12. GET USB CONNECTION STATUS

Short GetUSBConnectionStatus()

Checks whether the USB connection to the device is still active.

Return Values

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

Examples

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

See Also

Connect

4.4.13. GET FIRMWARE

Short GetExtFirmware(ByRef Short A0, ByRef Short A1, By Ref Short A2, By Ref Short A3, By Ref String Firmware)

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

Parameters

Variable	Description
A0	Required. User defined variable for factory use only.
A1	Required. User defined variable for factory use only.
A2	Required. User defined variable for factory use only.
A3	Required. User defined variable for factory use only.
Firmware	Required. User defined string variable which will be updated with the current firmware version, for example "B3".

Return Values

Value	Description
0	Command failed
>0	Command completed successfully

Examples

Python	`status = MyPTE1.GetExtFirmware(A0, A1, A2, A3, Firmware)`
	`if status[0] > 0:`
	`  Firmware = str(status[5])`
	`  print("Firmware Version:", Firmware)`

Visual Basic	`If MyPTE1.GetExtFirmware(A0, A1, A2, A3, Firmware) > 0 Then`
	`  MsgBox ("Firmware Version: " & Firmware)`
	`End If`

Visual C++	`if (MyPTE1->GetExtFirmware(A0, A1, A2, A3, Firmware) > 0)`
	`{`
	`  MessageBox::Show("Firmware Version: " + Firmware);`
	`}`

Visual C#	`if(MyPTE1.GetExtFirmware(ref(A0),ref(A1),ref(A2),ref(A3),ref(Firmware)) > 0)`
	`{`
	`  MessageBox.Show("Firmware Version: " + Firmware);`
	`}`

MatLab	`[status,A0, A1, A2, A3, Firmware] = MyPTE1.GetExtFirmware(0, 0, 0, 0, '')`
	`if status > 0`
	`  h = msgbox('Firmware Version: ', Firmware)`
	`end`

4.4.14. GET FIRMWARE VERSION (ANTIQUATED)

Short GetFirmware()

4.5. DLL - Ethernet Configuration Functions

These functions provide a method of configuring the device's Ethernet IP settings, they can only be sent using the USB connection. Refer to Ethernet Control API for additional details on the Ethernet configuration and default behavior.

Function	Syntax
Get Ethernet Configuration	int GetEthernet_CurrentConfig(ByRef int IP1, ByRef int IP2, ByRef int IP3, ByRef int IP4, ByRef int Mask1, ByRef int Mask2, ByRef int Mask3, ByRef int Mask4, ByRef int GW1, ByRef int GW2, ByRef int GW3, ByRef int GW4)
Get DHCP Status	int GetEthernet_UseDHCP()
Use DHCP	int SaveEthernet_UseDHCP(int UseDHCP)
Get IP Address	int GetEthernet_IPAddress(ByRef int b1, b2, b3, b4)
Save IP Address	int SaveEthernet_IPAddress(int b1, int b2, int b3, int
Get MAC Address	int GetEthernet_MACAddress(ByRef int m1, m2, m3, m4, m5,
Get Network Gateway	int GetEthernet_NetworkGateway(ByRef int b1, b2, b3, b4)
Save Network Gateway	int SaveEthernet_NetworkGateway(int b1, int b2, int b3,
Get Subnet Mask	int GetEthernet_SubNetMask(ByRef int b1, b2, b3, b4)
Save Subnet Mask	int SaveEthernet_SubnetMask(int b1, int b2, int b3, int
Get TCP/IP Port	int GetEthernet_TCPIPPort(ByRef int port)
Set HTTP Port & Enable / Disable HTTP	int SaveEthernet_TCPIPPort(int port)
Get Telnet Port	int GetEthernet_TelnetPort(ByRef int port)
Set Telnet Port & Enable / Disable Telnet	int SaveEthernet_TelnetPort(int port)
Get Password Requirement	int GetEthernet_UsePWD()
Set Password Requirement	int SaveEthernet_UsePWD(int UsePwd)
Get Password	int GetEthernet_PWD(ByRef string Pwd)
Set Password	int SaveEthernet_PWD(string Pwd)
Set Telnet Prompt	Int SaveEthernet_PromptMN(Int Enable_Prompt)
Get Telnet Prompt	Int GetEthernet_PromptMN()

4.5.1. GET ETHERNET CONFIGURATION

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

See Also

Get DHCP Status

4.5.2. GET DHCP STATUS

int GetEthernet_UseDHCP()

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

Return Values

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

Examples

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

See Also

Get Ethernet Configuration

4.5.3. USE DHCP

int SaveEthernet_UseDHCP(int UseDHCP)

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.5.4. GET IP ADDRESS

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

Returns the user-entered static IP address.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

See Also

Get Ethernet Configuration
Get DHCP Status

4.5.5. SAVE IP *ADDRESS*

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Use DHCP

4.5.6. GET MAC ADDRESS

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.5.7. GET NETWORK GATEWAY

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

Returns the user-entered network gateway IP address.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

See Also

Get Ethernet Configuration
Get DHCP Status

4.5.8. SAVE NETWORK GATEWAY

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Use DHCP

4.5.9. GET SUBNET MASK

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

Returns the user-entered subnet mask.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

See Also

Get Ethernet Configuration
Get DHCP Status

4.5.10. SAVE SUBNET MASK

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Use DHCP

4.5.11. GET TCP/IP PORT

int GetEthernet_TCPIPPort(ByRef int port)

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

Parameters

Variable	Description
port	Integer variable which will be updated with the TCP / IP port. A value of 0 or 65535 indicates that HTTP communication has been disabled (requires firmware D7 or later).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.5.12. SET HTTP PORT & ENABLE / DISABLE HTTP

int SaveEthernet_TCPIPPort(int port)

Sets the TCP / IP port to be used for HTTP communication. The default is port 80. Set port 0 or 65535 to disable HTTP (requires firmware D7 or later) or any valid port to enable.

Parameters

Variable	Description
port	TCP / IP port to be used for HTTP communication. To disable HTTP communication, specify port 0 or 65535 (requires firmware D7 or later).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.5.13. GET TELNET PORT

int GetEthernet_TelnetPort(ByRef int port)

Returns the TCP/IP port in use for Telnet communication. The default is port 23.

Parameters

Variable	Description
port	Integer variable which will be updated with the Telnet port. A value of 0 or 65535 indicates that Telnet communication has been disabled (requires firmware D7 or later).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.5.14. SET TELNET PORT & ENABLE / DISABLE TELNET

int SaveEthernet_TelnetPort(int port)

Sets the TCP / IP port to be used for Telnet communication. The default is port 23. Set port 0 or 65535 to disable Telnet (requires firmware D7 or later) or any valid port to enable.

Parameters

Variable	Description
port	Numeric value of the Telnet port. To disable Telnet communication, specify port 0 or 65535 (requires firmware D7 or later).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SaveEthernet_TelnetPort(21)`
Visual Basic	`status = MyPTE1.SaveEthernet_TelnetPort(21)`
Visual C++	`status = MyPTE1->SaveEthernet_TelnetPort(21);`
Visual C#	`status = MyPTE1.SaveEthernet_TelnetPort(21);`
MatLab	`status = MyPTE1.SaveEthernet_TelnetPort(21);`

4.5.15. GET PASSWORD REQUIREMENT

int GetEthernet_UsePWD()

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

Return Values

Value	Description
0	Password not required
1	Password required

Examples

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

See Also

Get Password

4.5.16. SET PASSWORD REQUIREMENT

int SaveEthernet_UsePWD(int UsePwd)

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

Parameters

Variable	Description
UseDHCP	Required. Integer value to set the password mode (0 – not required; 1 – required)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Set Password

4.5.17. GET PASSWORD

int GetEthernet_PWD(ByRef string Pwd)

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

See Also

Get Password Requirement

4.5.18. SET PASSWORD

int SaveEthernet_PWD(string Pwd)

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

Parameters

Variable	Description
Pwd	Password for Ethernet communication (not case sensitive, 20 characters maximum)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Set Password Requirement

4.5.19. SET TELNET PROMPT

Int SaveEthernet_PromptMN(Int Enable_Prompt)

Determines the prompt to be returned by the test system for Telnet communication. By default, the prompt is disabled so the response for Telnet communication is a new line character. When enabled, a full prompt is returned to the unit in response to all Telnet communication, taking the form MODEL_NAME>.

Parameters

Variable	Value	Description
Enable_Prompt	0	Disabled (new line character returned)
	1	Enabled (full model name prompt returned)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Example

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

4.5.20. GET TELNET PROMPT

Int GetEthernet_PromptMN()

Indicates whether a full prompt is to be returned by the test system for Telnet communication. By default, the prompt is disabled so the response for Telnet communication is a new line character. When enabled, a full prompt is returned to the unit in response to all Telnet communication, taking the form MODEL_NAME>.

Return Values

Value	Description
0	Disabled (new line character returned)
1	Enabled (full model name prompt returned)

Example

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

5. USB Control via Direct Programming (Linux)

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

5.1. USB Interrupt Code Concept

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

- Mini-Circuits Vendor ID: 0x20CE
- Modular Test System Product ID: 0x22

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

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

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

5.2. Summary of Interrupt Codes

Function	Code (Byte 0)	Summary
Get Device Model Name	40	
Get Device Serial Number	41	
Send SCPI Command	1 or 2 or 42	
Get Firmware	99	
Get Internal Temperature	114	Sensor 1
	115	Sensor 2

5.3.1. GET DEVICE MODEL NAME

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	

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	

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

SCPI: Get Model Name

5.3.2. GET DEVICE SERIAL NUMBER

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	

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	

Example

The following example indicates that the connected ZTM Series 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

SCPI: Get Serial Number

5.3.3. SEND SCPI COMMAND

Sends a SCPI command to the modular test system and collects the response. 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 system.

Transmit Array

Byte	Data	Description
0	1 or 2 or 42	Interrupt code for Send SCPI Command Note: Any of 1, 2 or 42 can be used as the command byte and the same value will be received in byte 0 of the returned array. 42 can be convenient in some environments since it can easily be represented by its character value of "*".
1 - 63	SCPI String	The SCPI command to send represented as a series of ASCII character codes, one character code per byte

Returned Array

Byte	Data	Description
0	1 or 2 or 42	Interrupt code for Send SCPI Command
1 to (n-1)	SCPI Data	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	

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 Attenuator)

The SCPI command to set an attenuator in slot 1A to 70.25dB is `:RUDAT:1A:ATT:70.25` (see Set Attenuation). The ASCII character codes representing the 19 characters in this command should be sent in bytes 1 to 19 of the transmit array as follows:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	82	ASCII character code for R
3	85	ASCII character code for U
4	68	ASCII character code for D
5	65	ASCII character code for A
6	84	ASCII character code for T
7	58	ASCII character code for :
8	49	ASCII character code for 1
9	65	ASCII character code for A
10	58	ASCII character code for :
11	65	ASCII character code for A
12	84	ASCII character code for T
13	84	ASCII character code for T
14	58	ASCII character code for :
15	55	ASCII character code for 7
16	48	ASCII character code for 0
17	46	ASCII character code for .
18	50	ASCII character code for 2
19	53	ASCII character code for 5

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

5.3.4. GET FIRMWARE

Returns the internal firmware version of the system.

Transmit Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1- 63	Not significant	

Returned Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	Reserved	Internal code for factory use only
2	Reserved	Internal code for factory use only
3	Reserved	Internal code for factory use only
4	Reserved	Internal code for factory use only
5	Firmware Letter	ASCII code for the first character in the firmware revision identifier
6	Firmware Number	ASCII code for the second character in the firmware revision identifier
7-63	Not significant	

Example

The below returned array indicates that the system has firmware version "C3":

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

See Also

SCPI: Get Firmware

5.3.5. GET INTERNAL TEMPERATURE

Returns the internal temperature of the system in degrees Celsius to 2 decimal places.

Transmit Array

Byte	Data	Description
0	114, 115 or 118	Interrupt code for Get Internal Temperature: 114 = Check temperature sensor 1 115 = Check temperature sensor 2 (if available) 118 = Check temperature sensor 3 (if available)
163	Not significant	

Returned Array

Byte	Data	Description
0	114, 115 or 118	Interrupt code for Get Internal Temperature
1	43 or 45	ASCII code for the first character of the temperature: 43 = positive (+) 45 = negative (-)
2	Temperature Digit 1	ASCII character code for the first digit of the temperature reading
3	Temperature Digit 2	ASCII character code for the second digit of the temperature reading
4	46	ASCII character code for the decimal point symbol (".")
5	Temperature DP 1	ASCII character code for the first decimal place of the temperature reading
6	Temperature DP 2	ASCII character code for the second decimal place of the temperature reading
7-63	Not significant	

Example

To check the internal temperature measured by sensor 2, send the following transmit array:

Byte	Data	Description
0	115	Interrupt code for Get Internal Temperature (Sensor 2)

The below returned array would indicate a temperature of +28.43°C:

Byte	Data	Description
0	115	Interrupt code for Get Internal Temperature (Sensor 2)
1	43	ASCII character code for +
2	50	ASCII character code for 2
3	56	ASCII character code for 8
4	46	ASCII character code for .
5	52	ASCII character code for 4
6	51	ASCII character code for 3

See Also

SCPI: Get Internal Temperature

6. Ethernet Control API

Control of the device via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed below via HTTP or Telnet.

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

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

6.1. Configuring Ethernet Settings

The device's Ethernet IP settings can be configured using either the USB or Ethernet connections. Refer to the SCPI - Ethernet Configuration Commands section for details.

Configure all required parameters and then use the Update Ethernet Settings command to reset the controller and restart with the updated configuration. If connected via Ethernet, all subsequent commands / queries to the device will need to be attempted using the new Ethernet configuration. If a connection cannot be established, 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.

6.2. DHCP / Default IP Configuration

All models ship with DHCP enabled by default so a dynamic IP address should be assigned by the network (when supported). For units with firmware D3 or later, the device will revert to a default link-local / auto-IP address of 169.254.10.10 when no valid DHCP response is received (including when the device is directly connected to a PC via the LAN port).

The assigned IP can be identified from the network administrator, by using UDP to broadcast a query, or by using the USB connection. The latter 2 options can be accomplished using the Mini-Circuits GUI, or via a custom program. Please contact testsolutions@minicircuits.com for support.

The default auto-IP features provide a method to implement a static IP configuration, without first relying on DHCP, or resorting to the USB connection. The process would be:

1. Connect the device directly to a PC using the Ethernet interface
2. No DHCP response will be received from the PC so the device will assume the default auto-IP
3. Connect to the device on 169.254.10.10
4. Disable DHCP, set the required static IP configuration and reset the device
5. Re-connect using the updated IP configuration

6.3. HTTP Communication

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

```
http://ADDRESS:PORT/PWD;COMMAND
```

Where:

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

Example 1:

```
http://192.168.100.100:800/PWD=123;:SPDT:1A:STATE:2
```

- The system has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to "123"
- The command is to set switch 1A to state 2

Example 2:

```
http://10.10.10.10/:SP4T:1:STATE?
```

- The system has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to check the state of switch 4

6.4. Telnet Communication

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

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

The system can be optionally configured to return a full prompt to the user with each Telnet response. The prompt will take the form ZTM-X> where ZTM-X is the model name of the connected system.

1) Set up Telnet connection to a modular test system with IP address 10.0.6.46:

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

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

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

6.5. Device Discovery Using UDP

Limited support of UDP is provided for the purpose of "device discovery." This allows a user to request the IP address and configuration of all Mini-Circuits RCM & ZTM modular systems connected on the network. Full control of those units is then accomplished using HTTP or Telnet, as detailed previously.

Alternatively, the IP configuration can be identified or changed by connecting the system with the USB interface (see SCPI - Ethernet Configuration Commands).

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

UDP Ports

Mini-Circuits' test systems are configured to listen on UDP port 4950 and answer on UDP port 4951. Communication on these ports must be allowed through the computer's firewall in order to use UDP for device discovery. If the system's IP address is already known, it is not necessary to use UDP.

Transmission

The command `MODULAR-ZT?` should be broadcast to the local network using UDP protocol on port 4950.

Receipt

All similar Mini-Circuits devices which 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: `MODULAR-ZT?`

Received Data:

```
Model Name: ZTM-999
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: ZTM-999
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
```

7. Control Options for MacOS

Mini-Circuits is not able to provide formal software support (GUI & API) for MacOS users but it is possible to control Mini-Circuits' Ethernet enabled devices without any software installation, including from MacOS.

The key steps to get started would be as follows.

7.1. Connect & Identify Initial IP Address

For connection into a network supporting DHCP:

1. DHCP is enabled by default so an IP address should be assigned automatically when the device is connected to the network
2. Identify the assigned IP address by referring to the network administrator or router.
3. Alternatively, a broadcast query can be sent to the network using UDP so that all Mini-Circuits devices respond with their IP (refer to Device Discovery Using UDP)
4. Once identified, the dynamic IP can be used to connect and control the device, including to set a new static IP configuration if required

For a direct connection between the Mac and Mini-Circuits device:

1. For devices with the latest firmware, a default "link-local" IP of 169.254.10.10 will be set if no response is received from a DHCP server (which will be the case for a direct computer connection)
2. This IP can be used to connect to the device and update the Ethernet configuration as needed
3. Refer to DHCP / Default IP Configuration for details of supported models

7.2. Updating the Ethernet Configuration

Once the initial IP address has been identified using the above steps, the device can be connected in order to set a new static IP address configuration. This can be achieved by writing an automation program based on the ASCII / SCPI commands detailed in this manual.

Alternatively, for a one-time step as part of initial commissioning, it may be simpler to use Mini-Circuits' HTML tool which can be downloaded from:

https://www.minicircuits.com/softwaredownload/MCL_PTE_Ethernet_Config.zip

The tool is an HTML file which can be downloaded and opened on the computer, it provides a simple form which the user populates with the current IP address and the updated configuration to load. The HTML file connects and updates the Ethernet configuration as specified.

With a valid IP address, the full list of ASCII / SCPI commands summarized in this programming manual can be used to control the device.

The fallback in the event of an unknown or invalid IP configuration would be to connect the device by USB in order to overwrite the configuration.

8. Contact

Mini-Circuits

13 Neptune Avenue

Brooklyn, NY 11235

Phone: +1-718-934-4500

Email: sales@minicircuits.com

Web: www.minicircuits.com