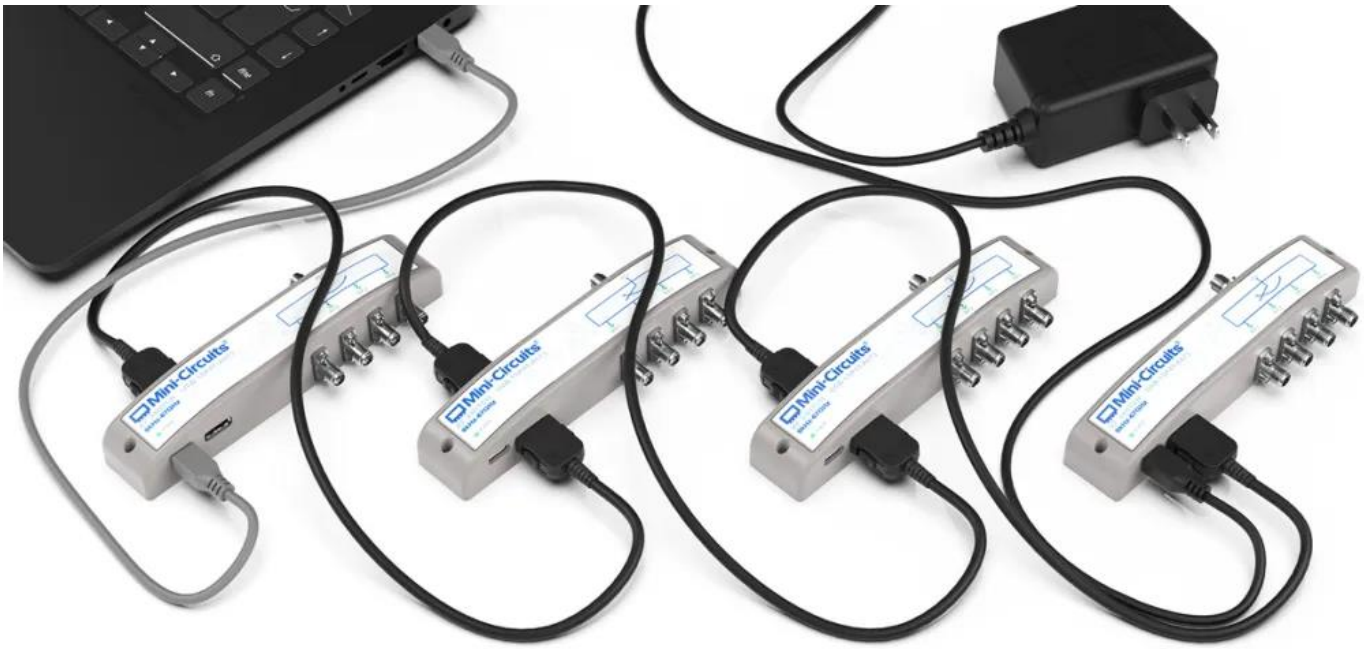
PROGRAMMING MANUAL

Solid-State Switches

eSB | RCS | U2C | USB Series



testsolutions@minicircuits.com | www.minicircuits.com

Contents

1. Overview & Supported Models 6
1.1. Control Methods 7
1.2. Programming Examples 7
1.3. Support Contacts 7
2. Mini-Circuits' Dynamic Daisy-Chain Control 8
2.1. Addressing Individual Switch Modules in the Daisy-Chain 8
3. SCPI Commands for Switch Control 9
3.1. SCPI - System Commands 9
3.1.1. Daisy-Chain - Assign Addresses 9
3.1.2. Daisy-Chain - Count Number of Slaves 9
3.1.3. Get Model Name 10
3.1.4. Get Serial Number 10
3.1.5. Get Firmware 10
3.2. SCPI - Switch Commands 11
3.2.1. Set Switch State 11
3.2.2. Get Switch State 12
3.3. SCPI - Switch Sequence Configuration 13
3.3.1. Set Number of Steps 13
3.3.2. Query Number of Steps 14
3.3.3. Set Indexed Step 15
3.3.4. Query Indexed Step Number 15
3.3.5. Set State of Indexed Step 16
3.3.6. Query State of Indexed Step 16
3.3.7. Set Dwell Time for Indexed Step 17
3.3.8. Query Dwell Time for Indexed Step 17
3.3.9. Set Dwell Time Units 18
3.3.10. Query Dwell Time Units 19
3.3.11. Set Number of Cycles 20
3.3.12. Query Number of Cycles 20
3.3.13. Set Sequence Direction 21
3.3.14. Query Sequence Direction 22
3.3.15. Start / Stop Switch Sequence 23
3.4. SCPI - Ethernet Configuration Commands 24
3.4.1. Get Current Ethernet Configuration 25
3.4.2. Get MAC Address 25
3.4.3. Get DHCP Status 26
3.4.4. Use DHCP 27
3.4.5. Get Static IP Address 28
3.4.6. Set Static IP Address 28
3.4.7. Get Static Network Gateway 29
3.4.8. Set Static Network Gateway 29
3.4.9. Get Static Subnet Mask 30
3.4.10. Set Static Subnet Mask 31
3.4.11. Get HTTP Port 32
3.4.12. Set HTTP Port & Enable / Disable HTTP 32
3.4.13. Get Telnet Port 33
3.4.14. Set Telnet Port & Enable / Disable Telnet 33

3.4.15. Get SSH Port 34
3.4.16. Set SSH Port 34
3.4.17. Get SSH Login Name 35
3.4.18. Save SSH Login Name 35
3.4.19. Get Password Requirement 36
3.4.20. Set Password Requirement 37
3.4.21. Get Password 38
3.4.22. Set Password 39
3.4.23. Update Ethernet Settings 40
4. USB Control API for Microsoft Windows 41
4.1. DLL API Options 41
4.1.1. .NET Framework 4.5 DLL (Recommended) 41
4.1.2. .NET Framework 2.0 DLL (Legacy Support) 41
4.1.3. ActiveX COM Object DLL (Legacy Support) 42
4.2. Referencing the DLL 43
4.3. Additional DLL Considerations 44
4.3.1. Mini-Circuits' DLL Use in Python / MatLab 44
4.3.2. Mini-Circuits' DLL Use in LabWindows / CVI 45
4.4. DLL – Switch Functions 46
4.4.1. Connect 47
4.4.2. Connect by Address 48
4.4.3. Disconnect 48
4.4.4. Send SCPI Command 49
4.4.5. Set USB-SP4T-63 State 50
4.4.6. Get USB-SP4T-63 State 51
4.4.7. Read Model Name 52
4.4.8. Read Serial Number 53
4.4.9. Set Address 54
4.4.10. Get Address 54
4.4.11. Get List of Connected Serial Numbers 55
4.4.12. Get List of Available Addresses 56
4.4.13. Get USB Connection Status 57
4.4.14. Get Firmware 58
4.4.15. Get Firmware Version (Antiquated) 58
4.5. DLL - Switch Sequence Functions 59
4.5.1. Set Number of Steps 59
4.5.2. Get Number of Steps 60
4.5.3. Set Step 61
4.5.4. Get Step Switch State 62
4.5.5. Get Step Dwell Time 63
4.5.6. Get Step Dwell Time Units 64
4.5.7. Set Sequence Direction 65
4.5.8. Get Sequence Direction 66
4.5.9. Set Number of Cycles 67
4.5.10. Get Number of Cycles 68
4.5.11. Enable / Disable Continuous Mode 69
4.5.12. Check Continuous Mode State 70
4.5.13. Start Sequence 71
4.5.14. Stop Sequence 72
4.6. DLL - Ethernet Configuration Functions 73
4.6.1. Get Ethernet Configuration 74

4.6.2. Get DHCP Status 76
4.6.3. Use DHCP 77
4.6.4. Get IP Address 78
4.6.5. Save IP *Address* 79
4.6.6. Get MAC Address 80
4.6.7. Get Network Gateway 82
4.6.8. Save Network Gateway 83
4.6.9. Get Subnet Mask 84
4.6.10. Save Subnet Mask 85
4.6.11. Get TCP/IP Port 86
4.6.12. Set HTTP Port & Enable / Disable HTTP 87
4.6.13. Get Telnet Port 88
4.6.14. Set Telnet Port & Enable / Disable Telnet 89
4.6.15. Get SSH Port 90
4.6.16. Save SSH Port 91
4.6.17. Get SSH Login Name 92
4.6.18. Save SSH Login Name 93
4.6.19. Get Password Requirement 94
4.6.20. Set Password Requirement 95
4.6.21. Get Password 96
4.6.22. Set Password 97
4.6.23. Reset Device 98
5. USB Control via Direct Programming (Linux) 99
5.1. USB Interrupt Code Concept 99
5.2. Interrupts – Core Functions 99
5.2.1. Get Device Model Name 100
5.2.2. Get Device Serial Number 101
5.2.3. Send SCPI Switch Command 102
5.2.4. Set USB-SP4T-63 State 103
5.2.5. Get USB-SP4T-63 State 104
5.2.6. Get Firmware 105
5.3. Interrupts - Switching Sequence Commands 106
5.3.1. Set Number of Steps 107
5.3.2. Get Number of Steps 108
5.3.3. Set Step 109
5.3.4. Get Step 110
5.3.5. Set Direction 111
5.3.6. Get Direction 112
5.3.7. Set Number of Cycles 113
5.3.8. Get Number of Cycles 114
5.3.9. Enable / Disable Continuous Mode 115
5.3.10. Check Continuous Mode State 116
5.3.11. Start / Stop Sequence 117
6. Ethernet Control API 118
6.1. Configuring Ethernet Settings 118
6.2. DHCP / Default IP Configuration 118
6.3. HTTP Communication 119
6.4. SSH Communication 119
6.5. Telnet Communication 120
6.6. Device Discovery Using UDP 121

7. Control Options for MacOS (Ethernet Only)..... 122
7.1. Connect & Identify Initial IP Address..... 122
7.2. Updating the Ethernet Configuration..... 122
8. Contact..... 123

1. Overview & Supported Models

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' USB & Ethernet controlled, high isolation solid-state switches. The contents apply to:

Model Name	Low (MHz)	High (GHz)	Switch Type	Switch Count	Control	Dynamic Daisy-Chain
U2C-1SP2T-63VH	10	6	SPDT	1	USB + I²C + SPI	No
USB-4SP2T-852H	10	8.5		4	USB	Yes
USB-2SP2T-DCH	DC	8		2	USB	Yes
USB-1SP2T-183	100	18		1	USB	Yes
USB-1SP2T-34	100	30		1	USB	Yes
USB-1SP2T-A44	100	43.5		1	USB	Yes
USB-1SP2T-673	100	67		1	USB	Yes
eSB-1SP2T-A673	100	67		1	USB	Yes
RCS-1SP2T-A673	100	67		1	LAN + USB	Yes
USB-SP4T-63	1	6	SP4T	1	USB	No
U2C-1SP4T-852H	2	8.5		1	USB + I²C	No
USB-2SP4T-852H	10	8.5		2	USB	Yes
USB-1SP4T-183	100	18		1	USB	Yes
USB-1SP4T-34	100	30		1	USB	Yes
eSB-1SP4T-A673	100	67		1	USB	Yes
RCS-1SP4T-A673	100	67		1	LAN + USB	Yes
USB-1SP8T-852H	10	8.5	SP8T	1	USB	Yes
USB-1SP8T-183	100	18		1	USB	Yes
USB-1SP8T-34	100	30		1	USB	Yes
USB-1SP16T-83H	1	8	SP16T	1	USB + TTL	Yes

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

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

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

https://www.minicircuits.com/WebStore/RF-Solid-State-Compact-Switch.html

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

1.1. Control Methods

Multiple control options are supported across Mini-Circuits' family of solid-state switches, refer to the individual model datasheets for details of which interfaces are available on a specific model:

1. Using SSH, HTTP or Telnet communication via an Ethernet TCP / IP connection (see Ethernet Control API), which is largely independent of the operating system.
2. Using the provided API DLL files (.Net or ActiveX objects) for USB control 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))
4. SPI, I^2C and TTL interfaces are available on some models for logic-based control systems. There is no software support for these methods so refer to the individual model datasheets for control instructions.

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:

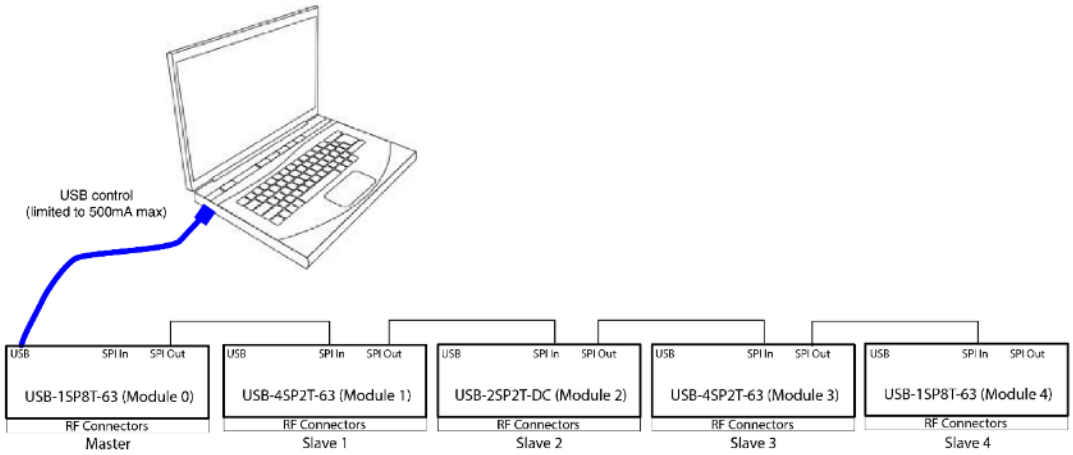
testsolutions@minicircuits.com

www.minicircuits.com/contact/worldwide_tech_support.html

2. Mini-Circuits' Dynamic Daisy-Chain Control

Mini-Circuits' dynamic daisy-chain control concept is designed to allow flexible switch systems to be configured and re-configured quickly and without the need to re-write the control software each time. Refer to the individual model datasheets or the table of section 1 to confirm which models support this feature.

Multiple supported switch modules of different types can be cascaded or "daisy-chained" together via the respective SPI In and Out connections. Only the first switch module in the chain (the "Master") is connected to a PC via its USB connection. All other modules (the "Slaves") receive their instructions and issue their replies through the Master.



The DC supply is also passed along the daisy-chain but it is important to note the maximum current available from the source and the maximum "pass-through" current supported by the individual switch modules (typically 500 mA). Additional power supplies can be connected to specific modules along the daisy-chain as required to ensure sufficient current for each module. Refer to the individual module datasheets for the current supply and pass-through requirements.

2.1. Addressing Individual Switch Modules in the Daisy-Chain

The first switch in the daisy-chain assumes the role of "Master" as soon as the unit is powered on with an active USB or Ethernet connection. Each switch module in the chain is dynamically assigned a 2-digit address, from 00 for the Master, with each daisy-chained switch taking the next in sequence from 01 to *nn*.

If new switch modules are subsequently added to the daisy-chain, or the order is changed, then the following commands can be issued to refresh the addresses and check the number of connected modules.

`:AssignAddresses`	Re-assign the addresses for the complete daisy-chain, starting from the 00 for the Master
`:NumberOfSlaves`	Query the number of slave devices connected in the daisy-chain

The full list of SCPI control commands / queries for identifying and controlling the switches is summarized in the following sections. These commands can be directed to any specific switch module by including the 2 digit address at the beginning of the string in the format shown below. Any commands sent without an address component will be directed to the Master (address 00).

For example:

`:MN?`	Return the model name of the Master module (address 00)
`:00:MN?`	Return the model name of the Master module (address 00)
`:01:MN?`	Return the model name of the first Slave module (address 01)
`:02:MN?`	Return the model name of the second Slave module (address 02)

Where only 1 switch module is connected by USB, with no additional daisy-chained switches, the address component is not required when sending SCPI commands.

3. SCPI Commands for Switch Control

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 Commands

Description	Command/Query
Assign Address	`:AssignAddresses`
Count Number of Slaves	`:NumberOfSlaves?`
Get Model Name	`:MN?`
Get Serial Number	`:SN?`
Get Firmware	`:FIRMWARE?`

3.1.1. DAISY-CHAIN - ASSIGN ADDRESSES

`:AssignAddresses`

Addresses from 00 to nn will be assigned auatomatically as soon as the daisy-chain is connected and powered up. This command can be issued after making any changes to the order of switch connections to reissue addresses to all connected switch modules.

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:AssignAddresses`	`1`

3.1.2. DAISY-CHAIN - COUNT NUMBER OF SLAVES

`:NumberOfSlaves?`

Returns the number of Slave modules connected to the Master in a daisy-chain configuration. Each module can then be queried and controlled using its unique address.

Return String

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

Examples

String to Send	String Returned
`:NumberOfSlaves`	`1`

3.1.3. 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=RC-2SPDT-A18`

3.1.4. 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`

3.1.5. 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`

3.2. SCPI - Switch Commands

Description	Command / Query
Set Switch State	`[Sw_Type]:[Sw_Channel]:STATE:[Sw_State]`
Get Switch State	`[Sw_Type]:[Sw_Channel]:STATE?`

3.2.1. SET SWITCH STATE

:[Sw_Type]:[Sw_Channel]:STATE:[Sw_State]

Set the state of a specific switch.

Parameters

Variable	Description
`[Sw_Type]`	The switch type being addressed, "SP2T", "SP4T", "SP8T", or "SP16T"
`[Sw_Channel]`	For modules with multiple switches specify the individual switch channel to set (A to D)
`[Sw_State]`	The switch port to connect to COM, from 0 to 16 (depending on switch module)

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:SP2T:A:STATE:1`	`1`
`:SP16T:STATE:16`	`1`
`:00:SP16T:STATE:16`	`00:1`
`:01:SP16T:STATE:16`	`01:1`

3.2.2. GET SWITCH STATE

:[Sw_Type]:[Sw_Channel]:STATE?

Returns the state of a specific switch.

Parameters

Variable	Description
[Sw_Type]	The switch type being addressed, "SP2T", "SP4T", "SP8T", or "SP16T"
[Sw_Channel]	For modules with multiple switches specify the individual switch channel (A to D)

Return String

Variable	Description
[State]	The switch port to which COM is connected, from 0 to 16 (depending on switch module)

Examples

String to Send	String Returned
:SP16T:STATE?	16
:00:SP16T:STATE?	00:16
:01:SP16T:STATE?	01:16

3.3. SCPI - Switch Sequence Configuration

Description	Command/Query
Set Number of Steps	:SEQ:STEPS:[steps]
Query Number of Steps	:SEQ:STEPS?
Set Indexed Step	:SEQ:STEP:[index]
Query Indexed Step Number	:SEQ:STEP?
Set State of Indexed Step	:SEQ:STATE:[state]
Query State of Indexed Step	:SEQ:STATE?
Set Dwell Time for Indexed Step	:SEQ:DWELLTIME:[time]
Query Dwell Time for Indexed Step	:SEQ:DWELLTIME?
Set Dwell Time Units	:SEQ:DWELLUNITS:[units]
Get Dwell Time Units	:SEQ:DWELLUNITS?
Set Number of Cycles	:SEQ:CYCLES:[count]
Query Number of Cycles	:SEQ:CYCLES?
Set Sequence Direction	:SEQ:DIRECTION:[mode]
Get Sequence Direction	:SEQ:DIRECTION?
Start / Stop Switch Sequence	:SEQ:MODE:[mode]

3.3.1. SET NUMBER OF STEPS

:SEQ:STEPS:[steps]

Sets the number of steps to be included in the switch sequence.

Parameters

Variable	Description
[steps]	The number of steps to program in the sequence

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:SEQ:STEPS:10	1

3.3.2. QUERY NUMBER OF STEPS

:SEQ:STEPS?

Returns the number of steps set to be included in the switch sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Variable	Description
`[steps]`	Number of steps to be included in the switch sequence

Examples

String to Send	String Returned
`:SEQ:STEPS?`	`10`

3.3.3. SET INDEXED STEP

:SEQ:STEP:[index]

Indexes a specific step number (1 to n) so that its parameters (such as dwell time and switch state) can be set.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
`[index]`	The step number to be configured

Return String

Value	Description
`0`	Command failed
`1`	Command completed successfully

Examples

String to Send	String Returned
`:SEQ:STEP:2`	`1`

3.3.4. QUERY INDEXED STEP NUMBER

:SEQ:STEPS?

Returns the index number (1 to n) of the step currently being configured.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Variable	Description
`[index]`	The step number (1 to n) to be configured

Examples

String to Send	String Returned
`:SEQ:STEP?`	`2`

3.3.5. SET STATE OF INDEXED STEP

:SEQ:STATE:[state]

The switch command to be sent for the indexed step in the switch sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
[state]	The port number to set for the switch at this step (0 to 16, model dependent). For multiple switch modules containing (eg: USB-4SP2T-63H), list all states separated by colons.

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:SEQ:STATE:1:2:2:1	1
:SEQ:STATE:8	1

3.3.6. QUERY STATE OF INDEXED STEP

:SEQ:STATE?

Returns the switch state to be set for the indexed step in the switch sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Variable	Description
[state]	The port number to set for the switch at this step (0 to 16, model dependent). For multiple switch modules containing (eg: USB-4SP2T-63H), list all states separated by colons.

Examples

String to Send	String Returned
:SEQ:STATE?	1:2:2:1
:SEQ:STATE?	8

3.3.7. SET DWELL TIME FOR INDEXED STEP

:SEQ:DWELLTIME:[time]

Sets the length of time for which the system will pause at the latest switch state, before proceeding with the switch sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
`[time]`	The length of time in to pause at this step before proceeding with the switch sequence

Return String

Value	Description
`0`	Command failed
`1`	Command completed successfully

Examples

String to Send	String Returned
`:SEQ:DWELLTIME:250`	`1`

3.3.8. QUERY DWELL TIME FOR INDEXED STEP

:SEQ:DWELLTIME?

Returns the length of time for which the system will pause at the latest switch state, before proceeding with the switch sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Variable	Description
`[time]`	The length of time for which the system will pause at this step before proceeding with the switch sequence

Examples

String to Send	String Returned
`:SEQ:DWELLTIME?`	`250`

3.3.9. SET DWELL TIME UNITS

:SEQ:DWELLUNITS:[units]

Sets the units of time to be used for the dwell time at the indexed step.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Value	Description
U	Microseconds
M	Milliseconds
S	Seconds

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:SEQ:DWELLUNITS:U	1
:SEQ:DWELLUNITS:M	1
:SEQ:DWELLUNITS:S	1

3.3.10. QUERY DWELL TIME UNITS

:SEQ:DWELLUNITS?

Returns the units of time to be used for the dwell time at the indexed step.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Value	Description
U	Microseconds
M	Milliseconds
S	Seconds

Examples

String to Send	String Returned
:SEQ:DWELLUNITS?	U
:SEQ:DWELLUNITS?	M
:SEQ:DWELLUNITS?	S

3.3.11. SET NUMBER OF CYCLES

:SEQ:CYCLES:[count]

Sets the number of times that the programmed switch sequence is to be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
`[count]`	The number of cycles / repetitions of the switch sequence to execute. Set the count to 0 to execute the sequence continuously.

Return String

Value	Description
`0`	Command failed
`1`	Command completed successfully

Examples

String to Send	String Returned
`:SEQ:CYCLES:5`	`1`

3.3.12. QUERY NUMBER OF CYCLES

:SEQ:CYCLES?

Queries the number of times that the programmed switch sequence is set to be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Variable	Description
`[count]`	The number of cycles / repetitions of the switch sequence to be executed. A count of 0 indicates that the sequence will run continuously.

Examples

String to Send	String Returned
`:SEQ:CYCLES?`	`5`

3.3.13. SET SEQUENCE DIRECTION

:SEQ:DIRECTION:[mode]

Sets which way through the programmed list of steps to run.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Value	Description
0	Forward (from first to last step)
1	Reverse (from last to first step)
2	Bi-directional (forward then reverse)

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:SEQ:DIRECTION:0	0
:SEQ:DIRECTION:1	1
:SEQ:DIRECTION:1	2

3.3.14. QUERY SEQUENCE DIRECTION

:SEQ:DIRECTION?

Queries which way the switch will run through the programmed list of steps.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return String

Value	Description
`0`	Forward (from first to last step)
`1`	Reverse (from last to first step)
`2`	Bi-directional (forward then reverse)

Examples

String to Send	String Returned
`:SEQ:DIRECTION?`	`0`
`:SEQ:DIRECTION?`	`1`
`:SEQ:DIRECTION?`	`2`

3.3.15. START / STOP SWITCH SEQUENCE

:SEQ:MODE:[mode]

Starts or stops the switch sequence based on the previously defined parameters.

Note: Any command / query sent to the system while a switch sequence is in process will stop the sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Value	Description
OFF	Disables / stops the switch sequence
ON	Enables / starts the switch sequence

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:SEQ:MODE:OFF	1
:SEQ:MODE:ON	1

3.4. 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 apply to the RCS series of Ethernet controlled solid-state switches.

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 SSH Port	:ETHERNET:CONFIG:SSHPORT?
Set SSH Port	:ETHERNET:CONFIG:SSHPORT:[port]
Get SSH Login Name	:ETHERNET:CONFIG:SSHLOGINNAME?
Save SSH Login Name	:ETHERNET:CONFIG:SSHLOGINNAME:[name]
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

3.4.1. 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

RCS series

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?`

3.4.2. GET MAC ADDRESS

:ETHERNET:CONFIG:MAC?

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

Applies To

RCS series

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?`

3.4.3. 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

RCS series

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

3.4.4. 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

RCS series

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

3.4.5. GET STATIC IP ADDRESS

:ETHERNET:CONFIG:IP?

Returns the user-entered static IP address.

Applies To

RCS series

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

3.4.6. 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

RCS series

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

3.4.7. GET STATIC NETWORK GATEWAY

:ETHERNET:CONFIG:NG?

Returns the user-entered network gateway IP address.

Applies To

RCS series

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

3.4.8. 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

RCS series

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

3.4.9. 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

RCS series

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

3.4.10. 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

RCS series

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

3.4.11. GET HTTP PORT

:ETHERNET:CONFIG:HTPORT?

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

Applies To

RCS series

Return String

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

Examples

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

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

3.4.12. 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 F1 or later) or any valid port to enable.

Applies To

RCS series

Parameters

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

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

3.4.13. GET TELNET PORT

:ETHERNET:CONFIG:TELNETPORT?

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

Applies To

RCS series

Return String

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

Examples

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

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

3.4.14. 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 F1 or later) or any valid port to enable.

Applies To

RCS series

Parameters

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

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

3.4.15. GET SSH PORT

:ETHERNET:CONFIG:SSHPORT?

Returns the TCP/IP port in use for SSH communication. The default is port 22.

Applies To

RCS series

Return String

Variable	Description
[port]	TCP / IP port to be used for SSH communication

Examples

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

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

3.4.16. SET SSH PORT

:ETHERNET:CONFIG:SSHPORT:[port]

Sets the TCP / IP port to be used for SSH communication. The default is port 22.

Applies To

RCS series

Parameters

Variable	Description
[port]	TCP / IP port to be used for SSH communication

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:SSHPORT:21

See Also

Update Ethernet Settings

3.4.17. GET SSH LOGIN NAME

:ETHERNET:CONFIG:SSHLOGINNAME?

Returns the login name to be used for SSH communication.

Applies To

RCS series

Return String

Variable	Description
[name]	Login name to be used for SSH communication

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SSHLOGINNAME?	Ssh_user

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

3.4.18. SAVE SSH LOGIN NAME

:ETHERNET:CONFIG:SSHLOGINNAME:[name]

Sets the login name to be used for SSH communication.

Applies To

RCS series

Parameters

Variable	Description
[name]	The login name for SSH communication

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ETHERNET:CONFIG:SSHLOGINNAME:ssh_user	1

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:SSHLOGINNAME:ssh_user

See Also

Update Ethernet Settings

3.4.19. 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

RCS series

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

3.4.20. SET PASSWORD REQUIREMENT

:ETHERNET:CONFIG:PWDENABLED:[enabled]

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

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

Applies To

RCS series

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

3.4.21. GET PASSWORD

:ETHERNET:CONFIG:PWD?

Returns the current password for SSH / HTTP / Telnet communication.

Applies To

RCS series

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

3.4.22. SET PASSWORD

:ETHERNET:CONFIG:PWD:[pwd]

Sets the password to be used for SSH / HTTP / Telnet communication. The password will only apply for HTTP / Telnet after enabling using Set Password Requirement.

Applies To

RCS series

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

3.4.23. 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.

Applies To

RCS series

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/solidstate.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: mcl_SolidStateSwitch_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: mcl_SolidStateSwitch64.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.

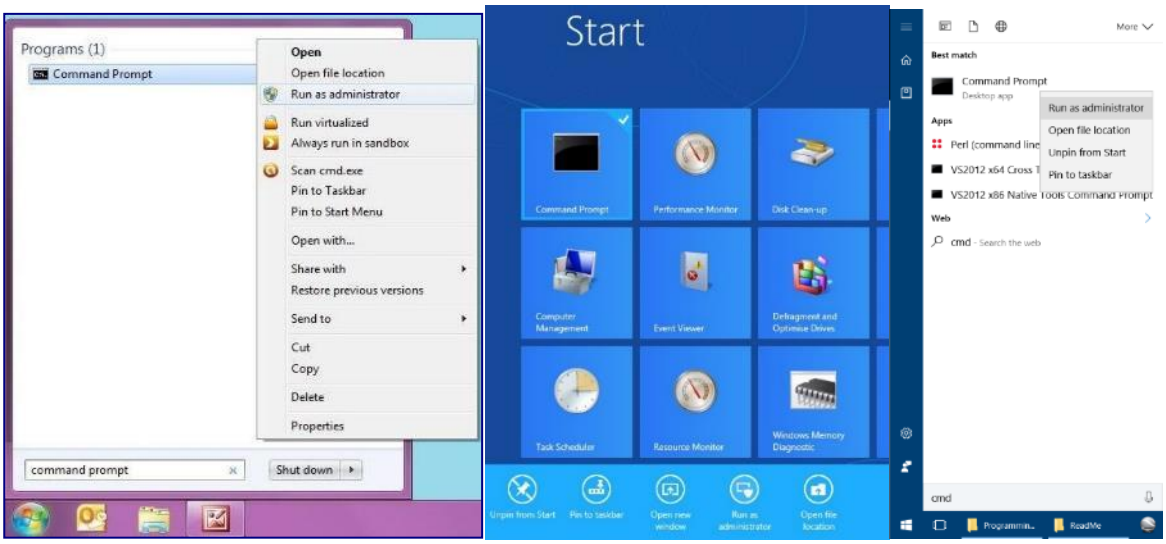
Filename: mcl_SolidStateSwitch.dll

Requirements

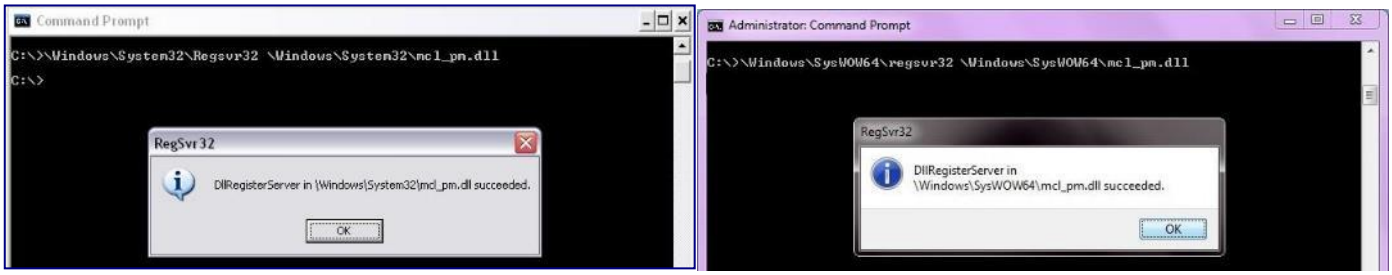
1) Microsoft Windows
2) Programming environment with support for ActiveX components

Installation

1) Download the latest DLL file from the Mini-Circuits website
2) Copy the .dll file (MCL_SolidStateSwitch.dll) to the correct directory:
   For 32-bit Windows operating systems: `C:\WINDOWS\System32`
   For 64-bit Windows operating systems: `C:\WINDOWS\SysWOW64`
3) 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. Enter the administrator credentials if requested
4) Register the DLL using regsvr32:
   32-bit PC: `Regsvr32 MCL_SolidStateSwitch.dll`
   64-bit PC: `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\MCL_SolidStateSwitch.dll`
5) Hit enter to confirm, 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 (mcl_SolidStateSwitch_NET45.dll)

(For operation with the .Net 2.0 DLL, replace "mcl_SolidStateSwitch_NET45" with "mcl_SolidStateSwitch64")

Python	`import clr  # Import the pythonnet CLR library`
	`clr.AddReference('C:\Windows\SysWOW64\mcl_SolidStateSwitch_NET45.dll')`
	`from mcl_SolidStateSwitch_NET45 import USB_Digital_Switch`
	`MyPTE1 = USB_Digital_Switch()`
	`MyPTE2 = USB_Digital_Switch()`

Visual Basic	`Public MyPTE1 As New mcl_SolidStateSwitch_NET45.USB_Digital_Switch` `Public MyPTE2 As New mcl_SolidStateSwitch_NET45.USB_Digital_Switch`

Visual C++	`mcl_SolidStateSwitch_NET45::USB_Digital_Switch ^MyPTE1 = gcnew`
	`mcl_SolidStateSwitch_NET45::USB_Digital_Switch();`
	`mcl_SolidStateSwitch_NET45::USB_Digital_Switch ^MyPTE2 = gcnew`
	`mcl_SolidStateSwitch_NET45::USB_Digital_Switch();`

Visual C#	`public mcl_SolidStateSwitch_NET45.USB_Digital_Switch MyPTE1 = new`
	`mcl_SolidStateSwitch_NET45.USB_Digital_Switch();`
	`public mcl_SolidStateSwitch_NET45.USB_Digital_Switch MyPTE2 = new`
	`mcl_SolidStateSwitch_NET45.USB_Digital_Switch();`

MatLab	`MCL_SW = NET.addAssembly('C:\Windows\SysWOW64\mcl_SolidStateSwitch_NET45.dll')`
	`MyPTE1 = mcl_SolidStateSwitch_NET45.USB_Digital_Switch`
	`MyPTE2 = mcl_SolidStateSwitch_NET45.USB_Digital_Switch`

Example Declarations using the ActiveX DLL (mcl_SolidStateSwitch.dll)

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

These functions apply to all Mini-Circuits solid state switch models and provide a means to control the device over a USB connection.

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)`
Set USB-SP4T-63 State	`int Set_SP4T_COM_To(Byte Port)`
Get USB-SP4T-63 State	`int Get_SP4T_State()`
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 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 ByRef 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 ByRef 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. SET USB-SP4T-63 STATE

int Set_SP4T_COM_To(Byte Port)

Set the switch state for USB-SP4T-63.

Applies To

USB-SP4T-63 only.

For all other models use the SCPI - Switch Commands which can be sent using the Send SCPI Command DLL function.

Parameters

Variable	Description
Port	Required. Byte value corresponding to the SP4T switch connection to be made. The 4 options for are: 1 = Com connected to port 1 2 = Com connected to port 2 3 = Com connected to port 3 4 = Com connected to port 4

Return Values

Value	Description
0	Command failed or invalid switch state requested
1	Command completed successfully

Examples

Python	`status = MyPTE1.Set_SP4T_COM_To(1)`
Visual Basic	`status = MyPTE1.Set_SP4T_COM_To(1)`
Visual C++	`status = MyPTE1->Set_SP4T_COM_To(1);`
Visual C#	`status = MyPTE1.Set_SP4T_COM_To(1);`
MatLab	`status = MyPTE1.Set_SP4T_COM_To(1);`

4.4.6. GET USB-SP4T-63 STATE

int Get_SP4T_State()

Get the switch state for USB-SP4T-63.

Applies To

USB-SP4T-63 only.

For all other models use the SCPI - Switch Commands which can be sent using the Send SCPI Command DLL function.

Return Values

Value	Description
0	Command failed
1	Switch has Com port connected to port 1
2	Switch has Com port connected to port 2
3	Switch has Com port connected to port 3
4	Switch has Com port connected to port 4

Examples

Python	`state = MyPTE1.Get_SP4T_State()`
Visual Basic	`state = MyPTE1.Get_SP4T_State()`
Visual C++	`state = MyPTE1->Get_SP4T_State();`
Visual C#	`state = MyPTE1.Get_SP4T_State();`
MatLab	`state = MyPTE1.Get_SP4T_State();`

4.4.7. READ MODEL NAME

Short Read_ModelName(ByRef 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
```

4.4.8. 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`

4.4.9. 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

4.4.10. 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

4.4.11. 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

4.4.12. 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

4.4.13. 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

4.4.14. 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`

4.4.15. GET FIRMWARE VERSION (ANTIQUATED)

Short GetFirmware()

4.5. DLL - Switch Sequence Functions

Mini-Circuits' high isolation solid-state switches have a "switching sequence mode" which allows the user to program a timed sequence of switch states into the switch's internal microcontroller, allowing very fast switching sequences to be triggered with no further USB communication.

Function	Syntax
Connect	`Short Connect(Optional String SN)`
Connect by Address	`Short ConnectByAddress(Optional Short Address)`

4.5.1. SET NUMBER OF STEPS

int SetSequence_NoOfSteps(int NoOfSteps)

Sets the number of steps to be configured for the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
NoOfSteps	Number of steps to configure (1 to 100)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetSequence_NoOfSteps(5)`
Visual Basic	`status = MyPTE1.SetSequence_NoOfSteps(5)`
Visual C++	`status = MyPTE1->SetSequence_NoOfSteps(5);`
Visual C#	`status = MyPTE1.SetSequence_NoOfSteps(5);`
MatLab	`status = MyPTE1.SetSequence_NoOfSteps(5);`

4.5.2. GET NUMBER OF STEPS

int GetSequence_NoOfSteps()

Returns the number of steps in the switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
NoOfSteps	Number of steps in the switching sequence (1 to 100)

Examples

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

4.5.3. SET STEP

int SetSequence_Step(int StepNo, int SwitchTo, int Dwell, int DwellUnits)

Configures the state and dwell time of a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
StepNo	Step index number, indexed from 0 to n - 1
SwitchTo	Switch state expressed as an integer, 0 to 16 (model dependent, the port which is to be connected to the Com port)
Dwell	Dwell time
DwellUnits	Dwell time units: 0 = microseconds (µs) 1 = milliseconds (ms) 2 = seconds (s)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetSequence_Step(2, 3, 5, 0)`
Visual Basic	`status = MyPTE1.SetSequence_Step(2, 3, 5, 0)`
Visual C++	`status = MyPTE1->SetSequence_Step(2, 3, 5, 0);`
Visual C#	`status = MyPTE1.SetSequence_Step(2, 3, 5, 0);`
MatLab	`status = MyPTE1.SetSequence_Step(2, 3, 5, 0);`

4.5.4. GET STEP SWITCH STATE

int GetSequence_SwitchTo(int StepNo)

Returns the switch state for a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
StepNo	Step index number, indexed from 0 to n - 1

Return Values

Value	Description
State	The switch state for the specified step in the sequence

Examples

Python	`result = MyPTE1.GetSequence_SwitchTo(2)`
Visual Basic	`result = MyPTE1.GetSequence_SwitchTo(2)`
Visual C++	`result = MyPTE1->GetSequence_SwitchTo(2);`
Visual C#	`result = MyPTE1.GetSequence_SwitchTo(2);`
MatLab	`result = MyPTE1.GetSequence_SwitchTo(2);`

4.5.5. GET STEP DWELL TIME

int GetSequence_Dwell(int StepNo)

Returns the dwell time (without units) for a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
StepNo	Step index number, indexed from 0 to n - 1

Return Values

Value	Description
Dwell	The dwell time (without units) for the specified step in the sequence

Examples

Python	`result = MyPTE1.GetSequence_Dwell(2)`
Visual Basic	`result = MyPTE1.GetSequence_Dwell(2)`
Visual C++	`result = MyPTE1->GetSequence_Dwell(2);`
Visual C#	`result = MyPTE1.GetSequence_Dwell(2);`
MatLab	`result = MyPTE1.GetSequence_Dwell(2);`

4.5.6. GET STEP DWELL TIME UNITS

int GetSequence_DwellUnits(int StepNo)

Returns the dwell time units for a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
StepNo	Step index number, indexed from 0 to n - 1

Return Values

Value	Description
DwellUnits	The dwell time units for the specified step in the sequence

Examples

Python	`result = MyPTE1.GetSequence_DwellUnits(2)`
Visual Basic	`result = MyPTE1.GetSequence_DwellUnits(2)`
Visual C++	`result = MyPTE1->GetSequence_DwellUnits(2);`
Visual C#	`result = MyPTE1.GetSequence_DwellUnits(2);`
MatLab	`result = MyPTE1.GetSequence_DwellUnits(2);`

4.5.7. SET SEQUENCE DIRECTION

int SetSequence_Direction(int Direction)

Sets the direction in which the sequence of switch states which will be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
Direction	Direction in which execute the sequence of switch states: 0 = Forward - ascending order from index point 0 to (n - 1) 1 = Reverse - descending order index point (n - 1) to 0 2 = Bi-Directional - ascending then descending order

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.5.8. GET SEQUENCE DIRECTION

int GetSequence_Direction()

Returns the direction in which the sequence of switch states will be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
Direction	Direction in which the sequence of switch states will be executed: 0 = Forward - ascending order from index point 0 to (n - 1) 1 = Reverse - descending order index point (n - 1) to 0 2 = Bi-Directional - ascending then descending order

Examples

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

4.5.9. SET NUMBER OF CYCLES

int SetSequence_NoOfCycles(int NoOfCycles)

Sets the number of times that the complete switching sequence will be executed. This setting will be ignored if the switch sequence has been configured to execute in continuous mode.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
NoOfSteps	Number of cycles, from 1 to 65,535

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetSequence_NoOfCycles(400)`
Visual Basic	`status = MyPTE1.SetSequence_NoOfCycles(400)`
Visual C++	`status = MyPTE1->SetSequence_NoOfCycles(400);`
Visual C#	`status = MyPTE1.SetSequence_NoOfCycles(400);`
MatLab	`status = MyPTE1.SetSequence_NoOfCycles(400);`

4.5.10. GET NUMBER OF CYCLES

int GetSequence_NoOfCycles()

Returns the number of times that the complete switching sequence will be executed. This setting will be ignored if the switch sequence has been configured to execute in continuous mode.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
NoOfCycles	Number of cycles, from 1 to 65,535

Examples

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

4.5.11. ENABLE / DISABLE CONTINUOUS MODE

int SetSequence_ContinuousMode(int Mode)

Configures whether the switch sequence will be executed continuously or for a pre-defined number of cycles. With continuous mode enabled, the sequence will repeat from the time the sequence is enabled by the user until the time it is disabled by the user; the setting for number of cycles will be ignored. With continuous mode disabled the sequence will be repeat according to the setting for number of cycles.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Parameters

Variable	Description
Mode	Setting for continuous mode: 0 = Continuous mode disabled 1 = Continuous mode enabled

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.5.12. CHECK CONTINUOUS MODE STATE

int GetSequence_ContinuousMode()

Indicates whether or not the switching sequence is configured to operate in continuous mode. With continuous mode enabled, the sequence will repeat from the time the sequence is enabled by the user until the time it is disabled by the user; the setting for number of cycles will be ignored. With continuous mode disabled the sequence will repeat according to the setting for number of cycles.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
Mode	Setting for continuous mode: 0 = Continuous mode disabled 1 = Continuous mode enabled

Examples

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

4.5.13. START SEQUENCE

int SetSequence_ON()

Starts the pre-defined switching sequence. The sequence will not operate unless all required parameters have been configured correctly.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.5.14. STOP SEQUENCE

int SetSequence_OFF()

Stops the pre-defined switching sequence. The sequence will not operate unless all required parameters have been configured correctly.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.6. 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. The controller must be reset after updating Ethernet parameters in order to load the new configuration, this can be achieved with a power cycle or by using the ResetDevice function.

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 b4)`
Get MAC Address	`int GetEthernet_MACAddress(ByRef int m1, m2, m3, m4, m5, m6)`
Get Network Gateway	`int GetEthernet_NetworkGateway(ByRef int b1, b2, b3, b4)`
Save Network Gateway	`int SaveEthernet_NetworkGateway(int b1, int b2, int b3, int b4)`
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 b4)`
Get TCP/IP Port	`int GetEthernet_TCPIPPort(ByRef int port)`
Save TCP/IP Port	`int SaveEthernet_TCPIPPort(int port)`
Get Telnet Port	`int GetEthernet_TelnetPort(ByRef int port)`
Save Telnet Port	`int SaveEthernet_TelnetPort(int port)`
Get SSH Port	`int GetEthernet_SSHPort(ByRef int port)`
Save SSH Port	`int SaveEthernet_SSHPort(int port)`
Get SSH Login Name	`int GetEthernet_SSHLoginName(ByRef string SSHLoginName)`
Save SSH Login Name	`int SaveEthernet_SSHLoginName(string SSHLoginName)`
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)`
Reset Device	`byte ResetDevice()`

4.6.1. 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.

Applies To

RCS series

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

4.6.2. 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.

Applies To

RCS series

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

4.6.3. 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.

Applies To

RCS series

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);`

See Also

Reset Device

4.6.4. 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.

Applies To

RCS series

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

4.6.5. 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.

Applies To

RCS series

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

Reset Device

4.6.6. 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.

Applies To

RCS series

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
```

4.6.7. 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.

Applies To

RCS series

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

4.6.8. 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.

Applies To

RCS series

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

Reset Device

4.6.9. GET SUBNET MASK

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

Returns the user-entered subnet mask.

Applies To

RCS series

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

4.6.10. 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.

Applies To

RCS series

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

Reset Device

4.6.11. 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.

Applies To

RCS series

Parameters

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

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`

4.6.12. 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 F1 or later) or any valid port to enable.

Applies To

RCS series

Parameters

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

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);`

See Also

Reset Device

4.6.13. GET TELNET PORT

int GetEthernet_TelnetPort(ByRef int port)

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

Applies To

RCS series

Parameters

Variable	Description
port	Required. Integer variable which will be updated with the Telnet port.

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`

4.6.14. 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 F1 or later) or any valid port to enable.

Applies To

RCS series

Parameters

Variable	Description
port	Required. Numeric value of the Telnet port.

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);`

See Also

Reset Device

4.6.15. GET SSH PORT

int GetEthernet_SSHPort(ByRef int port)

Returns the TCP/IP port in use for SSH communication. The default is port 22.

Applies To

RCS series

Parameters

Variable	Description
port	Required. Integer variable which will be updated with the SSH port.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.6.16. SAVE SSH PORT

int SaveEthernet_SSHPort(int port)

Sets the TCP / IP port to be used for SSH communication. The default is port 22.

Applies To

RCS series

Parameters

Variable	Description
port	Required. Numeric value of the SSH port.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Reset Device

4.6.17. GET SSH LOGIN NAME

int GetEthernet_SSHLoginName(ByRef string SSHLoginName)

Returns the login name for SSH communication.

Applies To

RCS series

Parameters

Variable	Description
SSHLoginName	String variable which will be updated with the SSH login name.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

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

Visual Basic

```
If MyPTE1.GetEthernet_SSHLoginName(SSHLoginName) > 0 Then
        MsgBox (SSHLoginName)
End If
```

Visual C++

```
if (MyPTE1->GetEthernet_SSHLoginName(SSHLoginName) > 0)
{
        MessageBox::Show(SSHLoginName);
}
```

Visual C#

```
if (MyPTE1.GetEthernet_SSHLoginName(ref(SSHLoginName)) > 0)
{
        MessageBox.Show(SSHLoginName);
}
```

MatLab

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

4.6.18. SAVE SSH LOGIN NAME

int SaveEthernet_SSHLoginName(string SSHLoginName)

Sets the login name for SSH communication.

Applies To

RCS series

Parameters

Variable	Description
SSHLoginName	The login name to set

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

See Also

Reset Device

4.6.19. GET PASSWORD REQUIREMENT

int GetEthernet_UsePWD()

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

Applies To

RCS series

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

4.6.20. SET PASSWORD REQUIREMENT

int SaveEthernet_UsePWD(int UsePwd)

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

Applies To

RCS series

Parameters

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

Reset Device

4.6.21. GET PASSWORD

int GetEthernet_PWD(ByRef string Pwd)

Returns the current password for SSH / HTTP / Telnet communication.

Applies To

RCS series

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

4.6.22. SET PASSWORD

int SaveEthernet_PWD(string Pwd)

Sets the password to be used for SSH / HTTP / Telnet communication. The password will only apply for HTTP / Telnet after enabling using Set Password Requirement.

Applies To

RCS series

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

Reset Device

4.6.23. RESET DEVICE

byte ResetDevice()

Called after updating Ethernet parameters, to reset the controller and reload with the updated Ethernet configuration.

Applies To

RCS series

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`MyPTE1.ResetDevice()`
Visual Basic	`MyPTE1.ResetDevice()`
Visual C++	`MyPTE1->ResetDevice();`
Visual C#	`MyPTE1.ResetDevice();`
MatLab	`MyPTE1.ResetDevice();`

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
- 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. Interrupts – Core Functions

Description	Command Code (Byte 0)
Get Device Model Name	40
Get Device Serial Number	41
Send SCPI Switch Command	42
Set USB-SP4T-63 State	1-4
Get USB-SP4T-63 State	15
Get Firmware	99

5.2.1. GET DEVICE MODEL NAME

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

Transmit Array

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

Returned Array

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

Example

The following array would be returned for Mini-Circuits' USB-2SP2T-63H switch (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1	85	ASCII character code for U
2	83	ASCII character code for S
3	42	ASCII character code for B
4	45	ASCII character code for -
5	50	ASCII character code for 2
6	83	ASCII character code for S
7	80	ASCII character code for P
8	50	ASCII character code for 2
9	84	ASCII character code for T
10	45	ASCII character code for -
11	54	ASCII character code for 6
12	51	ASCII character code for 3
13	72	ASCII character code for H
14	0	Zero value byte to indicate end of string

5.2.2. GET DEVICE SERIAL NUMBER

Returns the serial number of the connected switch.

Transmit Array

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

Returned Array

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

Example

The following example indicates that the connected switch box has serial number 1130922011 (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

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

5.2.3. SEND SCPI SWITCH COMMAND

Allows a SCPI command to be sent to the models listed below in order to set or read the switch state. Refer to Summary of SCPI Commands / Queries for the syntax of the SCPI commands.

Transmit Array

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

Returned Array

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

Example

The SCPI command to check the state of USB-SP8T-63H is `:SP8T:STATE?` (see Set SP8T Switch State).

The ASCII character codes representing the 12 characters in this command should be sent in bytes 1 to 12 of the transmit array as follows (see the Programming Examples & Troubleshooting Guide for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	42	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	83	ASCII character code for S
3	80	ASCII character code for P
4	56	ASCII character code for 8
5	84	ASCII character code for T
6	58	ASCII character code for :
7	83	ASCII character code for S
8	84	ASCII character code for T
9	65	ASCII character code for A
10	84	ASCII character code for T
11	69	ASCII character code for E
12	63	ASCII character code for ?

The returned array below would indicate the switch is in state 8 (COM <> port 8):

Byte	Data	Description
0	42	Interrupt code for Send SCPI Command
1	56	ASCII character code for 8
2	0	Zero value byte to indicate end of string

5.2.4. SET USB-SP4T-63 STATE

Set the switch state for USB-SP4T-63.

Applies To

USB-SP4T-63 only.

For all other models use the SCPI - Switch Commands which can be sent using the Send SCPI Command DLL function.

Transmit Array

Byte	Data	Description
0	1 - 4	Interrupt code for Set SP4T Switch state: 1 = Com to port 1 2 = Com to port 2 3 = Com to port 3 4 = Com to port 4
1 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	1 - 4	Interrupt code for Set SP4T Switch state: 1 = Com to port 1 2 = Com to port 2 3 = Com to port 3 4 = Com to port 4
1 - 63	Not significant	"Don't care" bytes, can be any value

Example

The following transmit array will set the SP4T switch to position 3 (Com connected to port 3):

Byte	Data	Description
0	3	Set SP4T to state 3

5.2.5. GET USB-SP4T-63 STATE

Set the switch state for USB-SP4T-63.

Applies To

USB-SP4T-63 only.

For all other models use the SCPI - Switch Commands which can be sent using the Send SCPI Command DLL function.

Transmit Array

Byte	Data	Description
0	15	Interrupt code for Get SP4T Switch State
1-63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	15	Interrupt code for Get SP4T Switch State
1	Switch State	Numeric value indicating the switch state: 1 = Com to port 1 2 = Com to port 2 3 = Com to port 3 4 = Com to port 4
2 - 63	Not significant	"Don't care" bytes, can be any value

Example

The below returned array indicates the switch is set as com to port 3:

Byte	Data	Description
0	15	Interrupt code for Get All SP4T Switch States
1	3	Switch set to state 3 (Com to port 3)

5.2.6. GET FIRMWARE

Returns the internal firmware version of the switch box.

Transmit Array

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

Returned Array

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

Example

The following returned array indicates that the switch box has firmware version C3:

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	55	Internal code for factory use only
2	52	Internal code for factory use only
3	83	Internal code for factory use only
4	87	Internal code for factory use only
5	67	ASCII code for the letter "C"
6	51	ASCII code for the number 3
7-63	Not significant	"Don't care" bytes, could be any value

5.3. Interrupts - Switching Sequence Commands

Mini-Circuits' high isolation solid-state switches have a "switching sequence mode" which allows the user to program a timed sequence of switch states into the switch's internal microcontroller, allowing very fast switching sequences to be triggered with no further USB communication.

Description	Byte 0	Byte 1
Set Number of Steps	204	0
Get Number of Steps	205	0
Set Step	204	1
Get Step	205	1
Set Direction	204	2
Get Direction	205	2
Set Number of Cycles	204	4
Get Number of Cycles	205	4
Enable / Disable Continuous Mode	204	3
Check Continuous Mode State	205	3
Start / Stop Sequence	204	5

5.3.1. SET NUMBER OF STEPS

Sets the number of steps to be configured for the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	0	Command for Number of Steps
2	Steps	Number of steps to configure (1 to 100)
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will set 5 points in the switching sequence:

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	0	Command for Number of Steps
2	5	Set 5 steps in the switching sequence

5.3.2. GET NUMBER OF STEPS

Returns the number of steps in the switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	0	Command for Number of Steps
2 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	Steps	Number of steps in the switching sequence
2 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that there are 5 steps in the pre-defined switching sequence:

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	5	5 steps in the switching sequence

5.3.3. SET STEP

Configures the state and dwell time of a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	1	Command for Step Configuration
2	Step_Index	Step index number, indexed from 0 to n - 1
3	Switch_State	Switch state expressed as an integer, 1 to 4 (the port which is to be connected to the Com port)
4	Dwell_0	Dwell time split into 2 bytes: Dwell_0 = INT (Dwell_Time / 256)
5	Dwell_1	Dwell time split into 2 bytes: Dwell_1 = Dwell_Time - (Dwell_0 * 256)
6	Dwell_Units	Dwell time units: 0 = microseconds (µs) 1 = milliseconds (ms) 2 = seconds (s)
7 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will set step 3 of 5 in the sequence (index number 2) so that Com will be connected to port 3, with a dwell time of 5 µs:

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	1	Command for Step Configuration
2	2	Step index number 2 for the third step in the sequence
3	3	Connect Com to port 3
4	0	Dwell_0 = INT (5 / 256) = 0
5	5	Dwell_1 = 5 - (0 * 256) = 5
6	0	Dwell time units are microseconds (µs)

5.3.4. GET STEP

Returns the state and dwell time of a single step within the pre-defined switching sequence.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	1	Command for Step Configuration
2	Step_Index	Step index number, indexed from 0 to n - 1
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Command for Set Switch Sequence Property
1	Step_Index	Step index number, indexed from 0 to n - 1
2	Switch_State	Switch state expressed as an integer, 1 to 4 (the port which is to be connected to the Com port)
3	Dwell_0	Dwell time split into 2 bytes: Dwell_Time = (256 * Dwell_0) + Dwell_1
4	Dwell_1	Dwell time split into 2 bytes
5	Dwell_Units	Dwell time units: 0 = microseconds (µs) 1 = milliseconds (ms) 2 = seconds (s)
6 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates step 3 of 5 in the sequence (index number 2) is configured so that Com will be connected to port 3, with a dwell time of 5 µs:

Byte	Data	Description
0	205	Command for Set Switch Sequence Property
1	2	Step index number 2 for the third step in the sequence
2	3	Connect Com to port 3
3	0	Dwell_Time = (256 * 0) + 5 = 5
4	5	Dwell_Time (calculated above)
5	0	Dwell time units are microseconds (µs)

5.3.5. SET DIRECTION

Sets the direction in which the sequence of switch states which will be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	2	Command for Direction
2	Direction	Direction in which execute the sequence of switch states: 0 = Forward - ascending order from index point 0 to (n - 1) 1 = Reverse - descending order index point (n - 1) to 0 2 = Bi-Directional - ascending then descending order
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will configure the sequence to execute in the forward direction:

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	2	Command for Direction
2	0	Set the forward direction

5.3.6. GET DIRECTION

Returns the direction in which the sequence of switch states will be executed.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	2	Command for Direction
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	Direction	Direction in which the sequence of switch states will be executed: 0 = Forward - ascending order from index point 0 to (n - 1) 1 = Reverse - descending order index point (n - 1) to 0 2 = Bi-Directional - ascending then descending order
2 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the sequence will be executed in the forward direction:

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	0	Sequence will execute in the forward direction

5.3.7. SET NUMBER OF CYCLES

Sets the number of times that the complete switching sequence will be executed. This setting will be ignored if the switch sequence has been configured to execute in continuous mode.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	4	Command for Number of Cycles
2	Cycles_0	Number of cycles (from 1 to 65,535) split into 2 bytes: Cycles_0 = INT (Cycles / 256)
3	Cycles_1	Number of cycles (from 1 to 65,535) split into 2 bytes: Cycles_1 = Cycles - (Cycles_0 * 256)
4 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will configure the switching sequence to be executed 400 times:

Byte	Data	Description	
0	204	Command for Set Switch Sequence Property	
1	4	Command for Step Configuration	
2	1	Cycles_0	= INT (400 / 256) = 1
3	144	Cycles_1	= 400 - (1 * 256) = 144

5.3.8. GET NUMBER OF CYCLES

Returns the number of times that the complete switching sequence will be executed. This setting will be ignored if the switch sequence has been configured to execute in continuous mode.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	4	Command for Number of Cycles
2 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	Cycles_0	Number of cycles (from 1 to 65,535) split into 2 bytes: Cycles = (256 * Cycles_0) + Cycles_1
2	Cycles_1	Number of cycles (from 1 to 65,535) split into 2 bytes
3 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the switching sequence has been configured to execute 400 times:

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	1	Number of cycles split into 2 bytes: Cycles = (256 * 1) + 144 = 400
2	144	Number of cycles split into 2 bytes (calculated above)

5.3.9. ENABLE / DISABLE CONTINUOUS MODE

Configures whether the switch sequence will be executed continuously or for a pre-defined number of cycles. With continuous mode enabled, the sequence will repeat from the time the sequence is enabled by the user until the time it is disabled by the user; the setting for number of cycles will be ignored. With continuous mode disabled the sequence will repeat according to the setting for number of cycles.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	3	Command for Continuous Mode
2	Mode	Enable / disable continuous mode: 0 = Continuous mode disabled 1 = Continuous mode enabled
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will configure the sequence to execute in continuous mode:

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	3	Command for Direction
2	1	Operate in continuous mode

5.3.10. CHECK CONTINUOUS MODE STATE

Indicates whether or not the switching sequence is configured to operate in continuous mode. With continuous mode enabled, the sequence will repeat from the time the sequence is enabled by the user until the time it is disabled by the user; the setting for number of cycles will be ignored. With continuous mode disabled the sequence will repeat according to the setting for number of cycles.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	3	Command for Continuous Mode
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	Mode	Setting for continuous mode: 0 = Continuous mode disabled 1 = Continuous mode enabled
2 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following returned array indicates that the sequence has been configured to operate in continuous mode:

Byte	Data	Description
0	205	Command for Get Switch Sequence Property
1	1	Sequence will operate in continuous mode

5.3.11. START / STOP SEQUENCE

Starts or stops the pre-defined switching sequence. The sequence will not operate unless all required parameters have been configured correctly.

Note: Sending any command to the switch whilst the pre-defined sequence is running will cause the sequence to stop.

Applies To

Supported Models	Required Firmware
U2C series	B9 or later
USB series	A5 or later
eSB / RCS series	All models

Transmit Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	5	Command for Start / Stop Sequence
2	Mode	Start or stop the switching sequence: 0 = Stop the sequence 1 = Start the sequence
3 - 63	Not significant	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1 - 63	Not significant	"Don't care" bytes, could be any value

Example

The following transmit array will start the switching sequence according to the pre-defined parameters:

Byte	Data	Description
0	204	Command for Set Switch Sequence Property
1	5	Command for Start / Stop Sequence
2	1	Start the sequence

6. Ethernet Control API

Control of the device via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed below via SSH, 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 and SSH are 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 F0 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;SETA=1
```

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

Example 2:

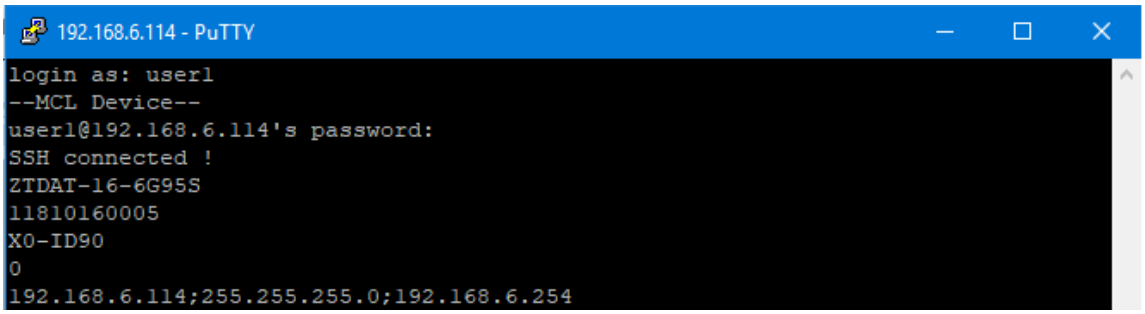
```
http://10.10.10.10/SETB=0
```

- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to set switch B to state 0

6.4. SSH Communication

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

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

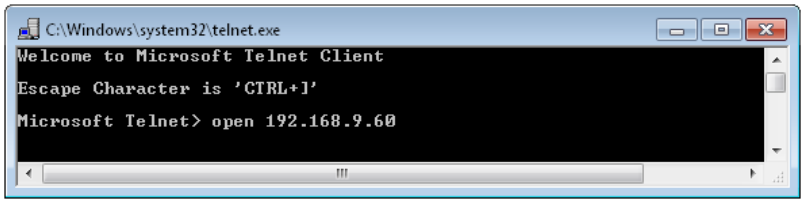


6.5. Telnet Communication

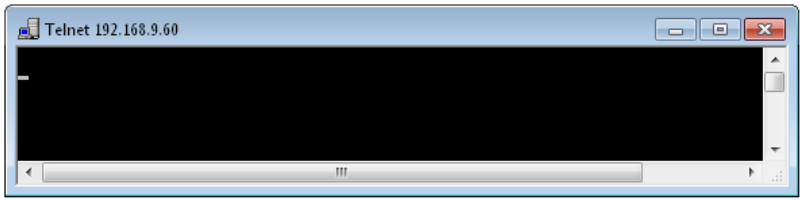
Communication is started by creating a Telnet connection to the device's IP address. On successful connection the "line feed" character will be returned. If the system has a password enabled, this must be sent as the first command after connection.

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:

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

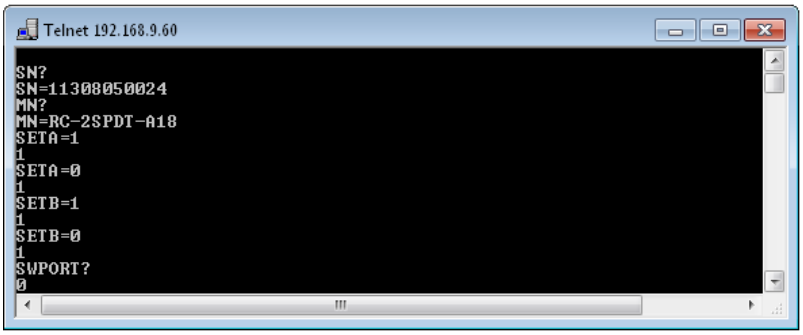


2) The "line feed" character is returned indicating the connection was successful:



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

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



6.6. 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' devices within the same family, connected on the network. Full control of those units is then accomplished using SSH, HTTP or Telnet, as detailed previously.

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

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

UDP Ports

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

Transmission

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

Receipt

All Mini-Circuits RC switch matrices that receive the request will respond with the following information (each field separated by CrLf) on port 4951:

- Model Name
- Serial Number
- IP Address/Port
- Subnet Mask
- Network Gateway
- Mac Address

Example

Sent Data: `MCLRFSWITCH?`

Received Data:

```
Model Name: RC-2SPDT-A18
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: RC-2SPDT-A18
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 (Ethernet Only)

Mini-Circuits is not able to provide formal software support (GUI & API) for MacOS users but it is possible to control Mini-Circuits' RCS series of Ethernet enabled controlled solid-state switches 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