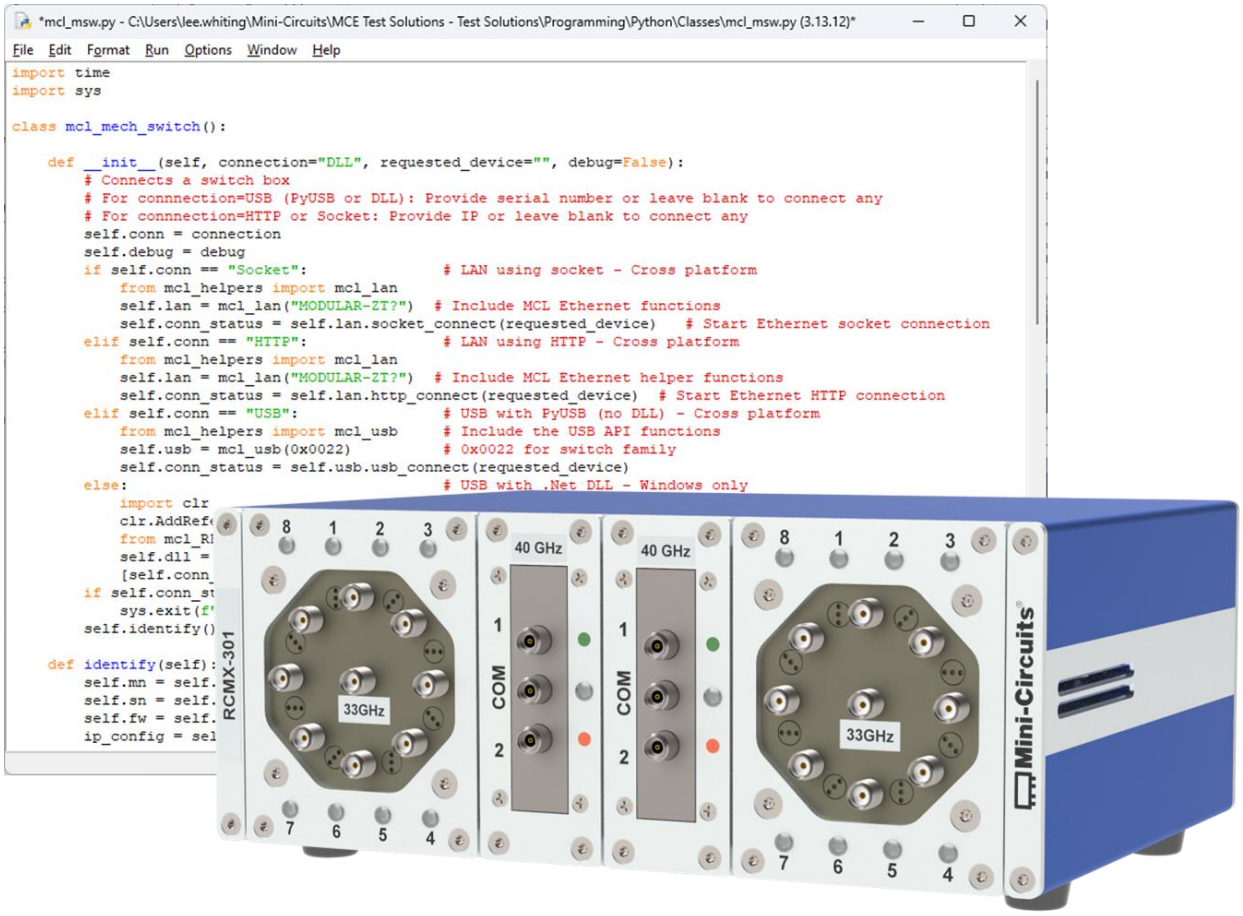
PROGRAMMING MANUAL

Mechanical Switch Assemblies

RCMX Series



testsolutions@minicircuits.com | www.minicircuits.com

Contents

1. Overview

This programming manual is intended for customers wishing to create their own interface for Mini-Circuits' RCMX series of USB & Ethernet controlled mechanical switch assemblies.

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:

www.minicircuits.com/WebStore/products/rcmx-modular-test-systems-downloads-and-documentation

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

www.minicircuits.com/WebStore/rcm.html

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

1.1. Control Methods

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

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

Setting states and querying the assembly is achieved using a set of commands based on SCPI (see SCPI Control Commands), sent via one of the above methods.

1.2. Programming Examples

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

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

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

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

1.3. Support Contacts

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

testsolutions@minicircuits.com

www.minicircuits.com/contact/worldwide_tech_support.html

2. RCMX Series Modular Concept

Mini-Circuits' modular concept is a flexible test system which can be configured with combinations of high reliability mechanical switches.

2.1. Addressing Individual Test Components

The following types of switches are available in the modular chassis and are typically addressed from 1 to *n*, left to right when looking at the front panel.

Component	Designator
SPDT Switch	SPDT
SP4T Switch	SP4T
SP6T Switch	SP6T
SP8T Switch	SP8T
Transfer / DPDT Switch	MTS

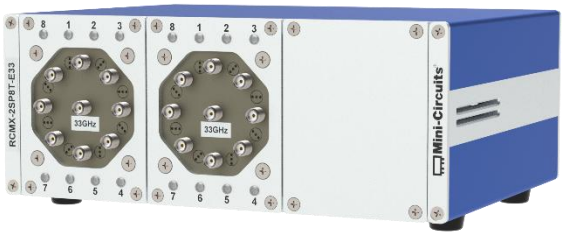
2.2. RCMX Series Examples

2.2.1. RCMX-301

Address	Component
1	SP8T
2	SPDT
3	SPDT
4	SP8T

2.2.2. RCMX-2SP8T-E33

Address	Component
1	SP8T
2	SP8T



3. SCPI Control Commands

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

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

3.1. SCPI - System Functions

3.1.1. Identify System

Command	`*IDN?`
Summary	Return the manufacturer, model name, serial number and firmware version.
Applies To	All models
Return Values	`[manufacturer], [model name], [serial number], [firmware]`
Example	`*IDN?` `> Mini-Circuits,RCMX-301,12603190025,A6-ID121`

3.1.2. Get Model Name

Command	`:MN?`	
Summary	Return the model name	
Applies To	All models	
Return Values	`[model]`	Model name
Example	`:MN?` `> MN=RCMX-301`	

3.1.3. Get Serial Number

Command	`:SN?`	
Summary	Return the serial number	
Applies To	All models	
Return Values	`[serial]`	Serial number
Example	`:SN?` `> SN=12603190025`	

3.1.4. Get Firmware

Command	`:FIRMWARE?`	
Summary	Return the firmware version	
Applies To	All models	
Return Values	`[firmware]`	Firmware version
Example	`:FIRMWARE?` `> FIRMWARE=A6-ID121`	

3.1.5. Get Configuration

Command	`:CONFIG:APP?`
Summary	Identify the switch modules fitted within this RCMX switch assembly. A string of integer codes is returned, separated by semi-colons, representing each switch type: 0 - Blank 1 - SPDT (all models) 4 - SP4T (18 GHz) 5 - MTS (18 GHz) 11 - SP6T (12-18 GHz models) 12 - SP8T (all models) 13 - SP6T (26.5-50 GHz models) 15 - SP12T (18 GHz) 33 - SP6T (Latching) 44 - SP4T (26.5-50 GHz models) 55 - MTS (26.5-40 GHz models)
Applies To	All models
Return Values	`[config]` List of switch types and states
Example	`:CONFIG:APP?` `> APP=12;1;1;12`

3.1.6. Get Configuration & Switch States

Command	`:CONFIG:STATES?`
Summary	Identify the switch modules and current switch states. A string is returned with the pair of details (switch type / state) for each module separated by semi-colons. The switch type / state are separated by an underscore within each pair of values. The codes representing each switch type are: 0 - Blank 1 - SPDT (all models) 4 - SP4T (18 GHz) 5 - MTS (18 GHz) 11 - SP6T (12-18 GHz models) 12 - SP8T (all models) 13 - SP6T (26.5-50 GHz models) 15 - SP12T (18 GHz) 33 - SP6T (Latching) 44 - SP4T (26.5-50 GHz models) 55 - MTS (26.5-40 GHz models)
Applies To	All models
Return Values	`[config]` List of switch types and states
Example	`:CONFIG:STATES?` `> STA=12_4;1_2;1_1;12_0`

3.2. SCPI - Monitoring

3.2.1. Get Internal Temperature

Command	`:TS0?`	
Summary	Returns the internal temperature, measured at the control board	
Return Values	`[temp]`	Temperature in degrees Celsius
Example	`:TS0?` `> +37.25`	

3.2.2. Get Heat Alarm

Command	`:HEATALARM?`	
Summary	Returns heat alarm status - comparing current temperature to a factory programmed safe level	
Return Values	`0`	Temperature normal
	`1`	Temperature exceeds safe limit
Example	`:HEATALARM?` `> 0`	

3.2.3. Set Fan Temperature Threshold

Command	`:FANS:TEMP:THRESHOLD:[set_point]:[value]`	
Summary	Sets the fan activation thresholds. The lower set-point defines the temperature threshold at which the fans will first turn on, spinning at 20% of max speed. The upper set-point must be a higher temperature value and sets the point at which the fans will increase to max RPM	
Parameters	`[set_point]`	The threshold to set; LOW or UP
	`[value]`	Temperature threshold in degrees Celsius
Return Values	`0- Failed`	Command failed
	`1- Success`	Command successful

Example	`:FANS:TEMP:THRESHOLD:LOW:25`
	`> 1 - Success`
	`:FANS:TEMP:THRESHOLD:UP:35`
	`> 1 - Success`	

3.2.4. Get Fan Temperature Threshold

Command	`:FANS:TEMP:THRESHOLD:[set_point]?`	
Summary	Returns the fan activation thresholds	
Parameters	`LOW`	Lower temperature threshold
	`UP`	Upper temperature threshold
Return Values	`[value]`	Temperature threshold

Example	`:FANS:TEMP:THRESHOLD:LOW?`
	`> 25.00`
	`:FANS:TEMP:THRESHOLD:UP?`
	`> 35.00`	

3.2.5. Get Fan State

Command	`FANSTATE?`	
Summary	Returns fan state	
Return Values	`0`	Fan not active / spinning
	`1`	Fan spinning
Example	`FANSTATE?` `> 1`	

3.2.6. Get Fan Alarm

Command	`FANALARM?`	
Summary	Returns fan fault status	
Return Values	`0`	No fault indicated
	`>1`	Potential fan fault; check the vent for obstructions
Example	`FANALARM?` `> 0`	

3.3. SCPI - Module Labels

3.3.1. Set Component Label

Command	`:UNIT:[address]:LABEL:[text]`	
Summary	Sets a custom name / label of up to 24 ASCII characters for the module	
Parameters	`[address]`	Module address
	`[text]`	Custom label text
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:UNIT:1:LABEL:Switch A` `> 1 - Success`	

3.3.2. Get Component Label

Command	`:UNIT:[address]:LABEL?`	
Summary	Returns the custom name / label for the module	
Parameters	`[address]`	Module address
Return Values	`LABEL="[text]"`	Custom label
Example	`:LABEL:1?` `> LABEL=Switch A`	

3.4. SCPI - Switch Control

3.4.1. Set Switch State

Command	`:[switch_type]:[address]:STATE:[value]`	
Summary	Sets a switch state - 1 to 2 for SPDT switches and 0 to n for SPnT switches, where n is the number of ports and 0 disconnects all ports. Transfer switch states are: 18 GHz models: 1 (J1-J3 & J2-J4) or 2 (J1-J2 & J3-J4) Other models: 1 (J1-J2 & J3-J4) or (J1-J3 & J2-J4)	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[address]`	Module address
	`[value]`	State to set: SPDT & MTS = 1 to 2 SPnT = 0 to n
Return Values	`0- Failed`	Command failed
	`1- Success`	Command successful

Example	`:SPDT:1:STATE:2`
	`> 1 - Success`
	`:MTS:2:STATE:1`
	`> 1 - Success`
	`:SP12T:3:STATE:8`
	`> 1 - Success`	

3.4.2. Get Switch State

Command	`:[switch_type]:[address]:STATE?`	
Summary	Returns a switch state	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[address]`	Module address
Return Values	`[state]`	Switch state

Example	`:SPDT:1:STATE?`
	`> 2`
	`:MTS:2:STATE?`
	`> 1`
	`:SP12T:3:STATE?`
	`> 8`	

3.4.3. Set All Similar Switch States

Command	`:[switch_type]:ALL:STATE:[values]`	
Summary	Sets all switches of a given type. Takes a string list of numeric switch states, 1 character per switch from address 1 to n. Use "x" to leave a switch unchanged or if the address contains a different module type.	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[states]`	String list of switch states to set
Return Values	`0- Failed`	Command failed
	`1- Success`	Command successful

Example	`:SPDT:ALL:STATE:xx21xx2`
	`> 1 - Success`
	`:SP4T:ALL:STATE:03xx44`
	`> 1 - Success`	

3.4.4. Get All Similar Switch States

Command	`:[switch_type]:ALL:STATE?`	
Summary	Returns the states of all switches of a given type	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
Return Values	`[states]`	List of switch states

Example	`:SPDT:ALL:STATE?`
	`> xx21xx2xxxxx`
	`:SP4T:ALL:STATE?`
	`> 03xx44xxxxxx`	

3.4.5. Set Switch Start-Up Mode

Command	`:[switch_type]:[address]:STARTUPSW:INDICATOR:[mode]`	
Summary	Sets the start-up state for a specific switch. The switch can be configured to load the default state or the last saved state at system power-up.	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[address]`	Module address
	`[mode]`	Switch start-up mode: L - Last Value (load last saved state) N - Normal (state 1 for SPDT / MTS, state 0 for SPnT)
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:SPDT:1:STARTUPSW:INDICATOR:L` `> 1 - Success`	

3.4.6. Get Switch Start-Up Mode

Command	`:[switch_type]:[address]:STARTUPSW:INDICATOR?`	
Summary	Returs the start-up state for a specific switch	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[address]`	Module address
Return Values	`L`	Last Value (load last saved state)
	`N`	Normal (state 1 for SPDT / MTS, state 0 for SPnT)
Example	`:SPDT:1:STARTUPSW:INDICATOR?` `> L`	

3.4.7. Get Switch Counter

Command	`:[switch_type]:[address]:SCOUNTER?`	
Summary	Returns switch cycle count	
Parameters	`[switch_type]`	Identifier for the switch type (SPDT, SP4T, SP6T, SP8T, SP12T, MTS)
	`[address]`	Module address
Return Values	`[count]`	Switch cycle count: Single count for SPDT / MTS models Semi-colon separated list of counts for SPnT models, 1 per port

Example	`:SPDT:1:SCOUNTER?`
	`> 9540`
	`:SP4T:2:SCOUNTER?`
	`> 95;450;10352;756`	

3.5. SCPI - Ethernet Configuration

These commands can be sent during a USB or Ethernet control session to change the IP configuration of the device. Send all commands needed for the new configuration and then send the "Update Ethernet Settings" commands as the last step to save and reset the controller. The device will reset and restart with the updated configuration after 1-5 seconds.

For USB connections, it will be necessary to disconnect and then reconnect the USB interface.
For Ethernet connections, the new IP configuration should be used for any further communication. If it is not possible to reconnect using the new IP configuration (for example due to a clash on the network) the USB connection can be used to recover and overwrite the settings.

3.5.1. Get Active IP Configuration

Command	`:ETHERNET:CONFIG:LISTEN?`	
Summary	Returns the IP configuration that is currently use; either the static IP entered by the user or the dynamic IP configuration when DHCP is enabled.	
Return Values	`[ip_config]`	String of active IP address, subnet mask and gateway, separated by semi-colons
Example	`:ETHERNET:CONFIG:LISTEN?` `> 192.100.1.1;255.255.255.0;192.100.1.0`	

3.5.2. Get MAC Address

Command	`:ETHERNET:CONFIG:MAC?`	
Summary	Returns the physical MAC (media access control) address of the device.	
Return Values	`[MAC]`	MAC address
Example	`:ETHERNET:CONFIG:MAC?` `> D0-73-7F-82-D8-01`	

3.5.3. Set DHCP or Static IP Mode

Command	`:ETHERNET:CONFIG:DHCPENABLED:[mode]`	
Summary	Enables or disables DHCP (dynamic host control protocol). When disabled, the device will attempt to connect using the user-entered static IP parameters (ensure the static configuration has been set before disabling). DHCP is enabled by default. The device reverts to an auto-IP of 169.254.10.10 when DHCP is enabled and no response is received from the server.	
Parameters	`0`	DHCP disabled (static IP settings will be used)
	`1`	DHCP enabled (dynamic IP address will be requested from DHCP server on the network)
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:DHCPENABLED:1` `> 1`	

3.5.4. Get DHCP Status

Command	`:ETHERNET:CONFIG:DHCPENABLED?`	
Summary	Indicates whether DHCP (dynamic host control protocol) is currently enabled. When disabled, the device will attempt to connect using the user-entered static IP parameters.	
Return Values	`0`	DHCP disabled (static IP settings will be used)
	`1`	DHCP enabled (dynamic IP address will be requested from DHCP server on the network)
Example	`:ETHERNET:CONFIG:DHCPENABLED?` `> 1`	

3.5.5. Set Static IP Address

Command	`:ETHERNET:CONFIG:IP:[ip]`	
Summary	Sets the static IP address to be used when DHCP is diabled.	
Parameters	`[ip]`	Static IP address
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:IP:192.168.1.25` `> 1`	

3.5.6. Get Static IP Address

Command	`:ETHERNET:CONFIG:IP?`	
Summary	Returns the user-entered static IP address.	
Return Values	`[ip]`	Static IP address
Example	`:ETHERNET:CONFIG:IP?` `> 192.168.1.25`	

3.5.7. Set Static Network Gateway

Command	`:ETHERNET:CONFIG:NG:[gateway]`	
Summary	Sets the IP address of the network gateway to be used when DHCP is diabled.	
Parameters	`[gateway]`	Network gateway IP address
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:NG:192.168.1.1` `> 1`	

3.5.8. Get Static Network Gateway

Command	`:ETHERNET:CONFIG:NG?`	
Summary	Returns the user-entered network gateway IP address.	
Return Values	`[gateway]`	Network gateway IP address
Example	`:ETHERNET:CONFIG:NG?` `> 192.168.1.1`	

3.5.9. Set Static Subnet Mask

Command	`:ETHERNET:CONFIG:SM:[mask]`	
Summary	Sets the IP address of the network gateway to be used when DHCP is diabled.	
Parameters	`[mask]`	Subnet mask
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:SM:255.255.255.0` `> 1`	

3.5.10. Get Static Subnet Mask

Command	`:ETHERNET:CONFIG:SM?`	
Summary	Returns the user-entered network gateway IP address.	
Return Values	`[mask]`	Subnet mask
Example	`:ETHERNET:CONFIG:SM?` `> 255.255.255.0`	

3.5.11. Enable HTTP & Set Port

Command	`:ETHERNET:CONFIG:HTPORT:[port]`	
Summary	Sets the TCP / IP port to be used for HTTP communication (default is port 80). Set to 0 or 65535 to disable HTTP communication.	
Parameters	`0`	Disable HTTP communication
	`[port]`	Enable HTTP communication on the specified port
	`65535`	Disable HTTP communication
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:HTPORT:8080` `> 1`	

3.5.12. Get HTTP Status & Port

Command	`:ETHERNET:CONFIG:HTPORT?`	
Summary	Returns the TCP / IP port set for HTTP communication. A value of 0 or 65535 indicates HTTP communication is disabled.	
Return Values	`[port]`	Port to use for HTTP communication
Example	`:ETHERNET:CONFIG:HTPORT?` `> 8080`	

3.5.13. Enable Telnet & Set Port

Command	`:ETHERNET:CONFIG:TELNETPORT:[port]`	
Summary	Sets the TCP / IP port to be used for Telnet communication (default is port 23). Set to 0 or 65535 to disable Telnet communication.	
Parameters	`0`	Disable Telnet communication
	`[port]`	Enable Telnet communication on the specified port
	`65535`	Disable Telnet communication
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:TELNETPORT:21` `> 1`	

3.5.14. Get Telnet Status & Port

Command	`:ETHERNET:CONFIG:TELNETPORT?`	
Summary	Returns the TCP / IP port set for Telnet communication. A value of 0 or 65535 indicates Telnet communication is disabled.	
Return Values	`0`	Disable Telnet communication
	`[port]`	Enable Telnet communication on the specified port
	`65535`	Disable Telnet communication
Example	`:ETHERNET:CONFIG:TELNETPORT?` `> 21`	

3.5.15. Set SSH Port

Command	`:ETHERNET:CONFIG:SSHPORT:[port]`	
Summary	Sets the TCP / IP port to be used for SSH communication (default is port 22).	
Parameters	`[port]`	Port to use for SSH communication
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:SSHPORT:20` `> 1`	

3.5.16. Get SSH Port

Command	`:ETHERNET:CONFIG:SSHPORT?`	
Summary	Returns the TCP / IP port set for SSH communication.	
Return Values	`[port]`	Port to use for SSH communication
Example	`:ETHERNET:CONFIG:SSHPORT?` `> 20`	

3.5.17. Set SSH Login Name

Command	`:ETHERNET:CONFIG:SSHLOGINNAME:[name]`	
Summary	Sets the login name for SSH communication.	
Parameters	`[name]`	Login name
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:ETHERNET:CONFIG:SSHLOGINNAME:ssh_user` `> 1`	

3.5.18. Get SSH Login Name

Command	`:ETHERNET:CONFIG:SSHLOGINNAME?`
Summary	Returns the login name for SSH communication.
Return Values	`[name]` Login name
Example	`:ETHERNET:CONFIG:SSHLOGINNAME?` `> sshuser`

3.5.19. Set Password

Command	`:ETHERNET:CONFIG:PWD:[pwd]`
Summary	Sets the password for Ethernet access. The password is required for SSH communication but can be enabled or disabled for HTTP and Telnet communication.
Parameters	`[password]` Password up to 20 characters (not case sensitive)
Example	`:ETHERNET:CONFIG:PWD:12345` `> 1`

3.5.20. Get Password

Command	`:ETHERNET:CONFIG:PWD?`
Summary	Returns the password for Ethernet access.
Return Values	`[password]` Password up to 20 characters (not case sensitive)
Example	`:ETHERNET:CONFIG:PWD?` `> 12345`

3.5.21. Set Password Requirement for HTTP & Telnet

Command	`:ETHERNET:CONFIG:PWDENABLED:[mode]`
Summary	Enables or disables the requirement to provide a password for HTTP / Telnet communication (the password is always required for SSH).
Parameters	`0` Password not required for HTTP / Telnet
	`1` Password required for HTTP / Telnet
Return Values	`0` Command failed
	`1` Command successful
Example	`:ETHERNET:CONFIG:PWDENABLED:1` `> 1`

3.5.22. Get Password Status for HTTP & Telnet

Command	`:ETHERNET:CONFIG:PWDENABLED?`
Summary	Confirms whether a password is required for HTTP / Telnet communication (the password is always required for SSH).
Return Values	`0` Password not required for HTTP / Telnet
	`1` Password required for HTTP / Telnet
Example	`:ETHERNET:CONFIG:PWDENABLED?` `> 1`

3.5.23. Update Ethernet Settings / Reset Controller

Command	`:SYSTEM:RESET_CPU`	
Summary	Resets and restarts the controller. Required after making changes to the Ethernet configuration. The device will reset and restart with the updated configuration after 1-5 seconds. For Ethernet connections, the new IP configuration should be used for any further communication. If it is not possible to reconnect using the new IP configuration (for example due to a clash on the network) the USB connection can be used to recover and overwrite the settings.	
Return Values	`0`	Command failed
	`1`	Command successful
Example	`:SYSTEM:RESET_CPU` `> 1`	

4. Ethernet Control API

Control of the device via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed above via HTTP, SSH or Telnet. UDP is supported for discovering available systems on the network.

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

4.1. 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.

4.2. Default Ethernet Configuration

Mini-Circuits' products ship with DHCP enabled by default so in most cases the device should be assigned a dynamic IP address when connected to the network. 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.

Once a valid IP address has been assigned and identified it can be re-configured in multiple ways:

1. Using the Windows GUI when connected by USB
2. Using the API when connected by USB or Ethernet
3. Using Mini-Circuits' HTML configuration tool when connected by Ethernet

4.3. Default Link-Local / Auto IP Address

Where supported, a default "link-local" auto IP address will be assumed when DHCP is enabled but the device does not receive a valid response from a DHCP server. This could be the case on networks with no DHCP server or when the device is connected directly via an Ethernet cable to a PC instead of via a network.

The default static / link-local IP address for all Mini-Circuits devices with the relevant firmware is 169.254.10.10.

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:

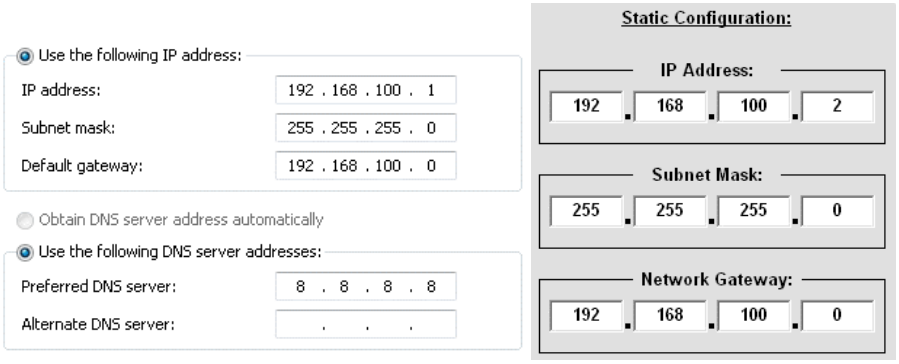
1. Connect the device directly to a PC using the Ethernet cable
2. No DHCP response will be received from the PC so the device will assume the default auto-IP
3. Connect to the device on 169.254.10.10
4. Disable DHCP, set the required static IP configuration and reset the device
5. Reconnect using the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

4.4. Direct Ethernet Cable Connection to PC

It may be necessary to set a compatible TCP / IP configuration on a PC in order to establish a direct connection by Ethernet cable between the device and PC rather than via a network. The values can be chosen arbitrarily as long as a valid and compatible range is applied to both PC and Mini-Circuits device. The key points are:

1. Set different IP addresses on the same subnet for the PC and device
2. Set another different IP address on the same subnet as the network gateway IP, using the same value on both PC and Mini-Circuits device
3. Set the same subnet mask on PC and device (ensuring the mask allows the above IP range)

An example of a working configuration is shown below, with the PC settings on the left and the static IP settings for the Mini-Circuits device on the right.



4.5. Device Discovery Using UDP

Limited support of UDP is provided for the purpose of discovering Mini-Circuits devices of the same family which are connected to the network. Full control of those units is then accomplished using SSH, HTTP or Telnet, as detailed previously.

Limitations

UDP is a simple transmission protocol that provides no method for error correction or guarantee of receipt. There are routing limitations with UDP broadcast queries which mean it is not usually possible to locate connected devices beyond the local subnet.

UDP Ports

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

Transmission

The command `MODULAR-ZT?` should be sent as a broadcast query to the broadcast address of the local network using UDP protocol on port 4950. The broadcast address is worked out from a bitwise OR operation of the host PC's IP address and the bit-complement of its subnet mask.

Receipt

All similar Mini-Circuits devices which receive the request will respond with the following information (each field separated by CrLf) on port 4951:

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

Example

Sent Data: `MODULAR-ZT?`

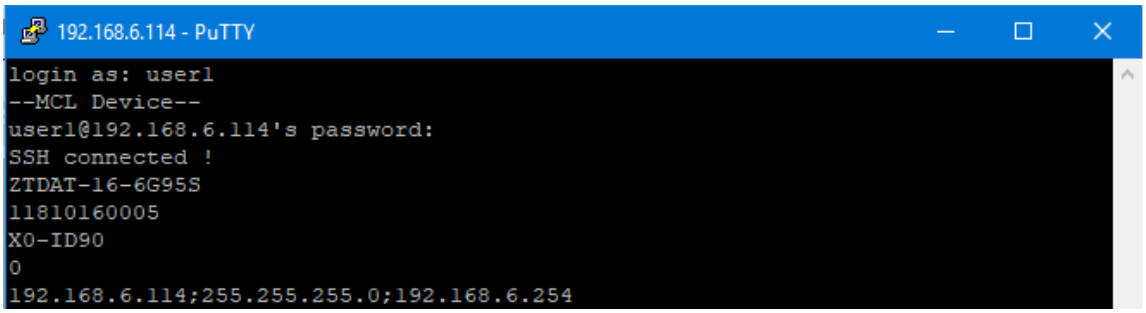
Received Data:

```
Model Name: RCMX-301
Serial Number: 12602120001
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: RCMX-301
Serial Number: 12602120002
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
```

4.6. SSH Communication

SSH allows secure communication with the device, 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.



4.7. HTTP Communication

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

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

Where:

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

Example 1:

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

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

Example 2:

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

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

4.8. Telnet Communication

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

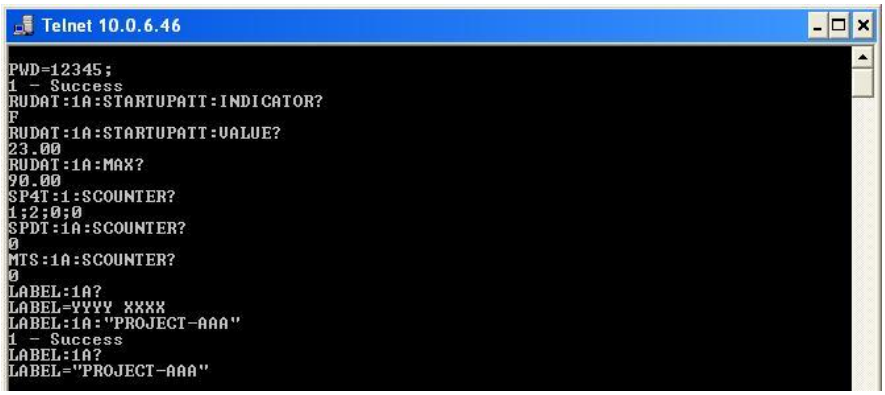
Each command must be terminated with the carriage return and line-feed characters (\r\n). Responses will be similarly terminated. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below.

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

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

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

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



5. 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 .Net Framework DLL file. 2 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) 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:

www.minicircuits.com/WebStore/products/rcmx-modular-test-systems-downloads-and-documentation

5.1. DLL API Options

5.1.1. .NET Framework 4.5 DLL

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

Filename: mcl_ZTM2_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**

5.1.2. ActiveX COM Object DLL (Legacy Support)

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

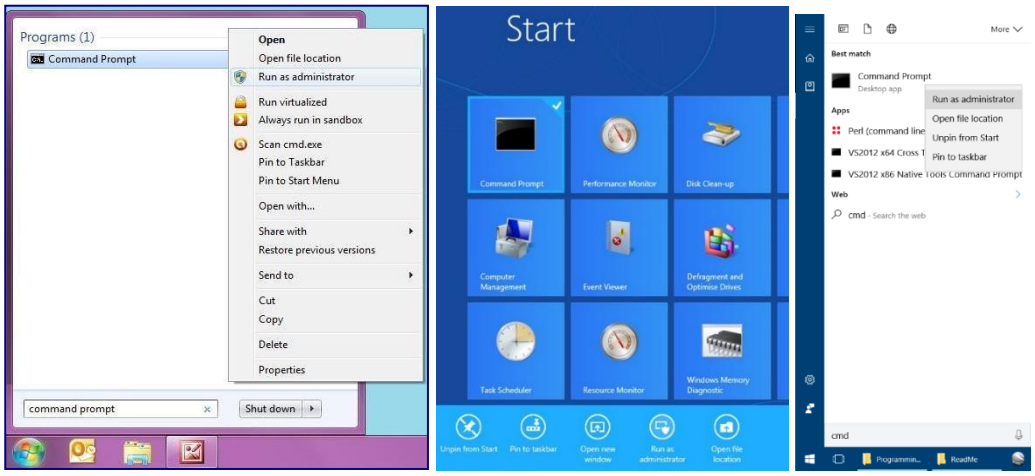
Filename: MCL_ZTM2.dll

Requirements

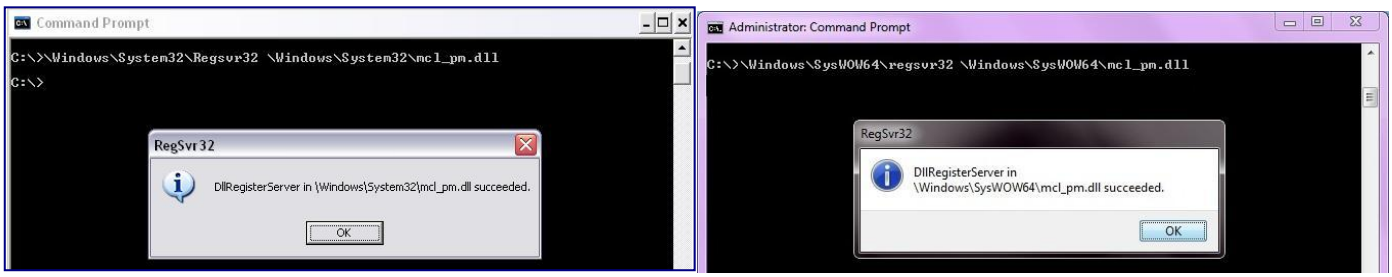
1) Microsoft Windows
2) Programming environment with support for ActiveX components

Installation

1) Copy the DLL file to the correct directory:
   For 32-bit Windows operating systems: `C:\WINDOWS\System32`
   For 64-bit Windows operating systems: `C:\WINDOWS\SysWOW64`
2) Open the Command Prompt in "Elevated" mode:
   a. Open the Start Menu/Start Screen and type "Command Prompt"
   b. Right-click on the shortcut for the Command Prompt
   c. Select "Run as Administrator"
   d. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3) Use regsvr32 to register the DLL:
4) 32-bit PC:
   ```
   Regsvr32 MCL_ZTM2.dll
   ```
5) 64-bit PC:
   ```
   \WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\MCL_ZTM2.dll
   ```
6) Register the DLL using regsvr32:
7) Hit enter to confirm and a message box will appear to advise of successful registration



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



Registering the DLL

5.2. 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_ZTM2_NET45.dll)

(For operation with the .Net 2.0 DLL, replace "mcl_ZTM2_NET45.dll" with "mcl_ZTM2_64.dll")

Python	`import clr  # Import the pythonnet CLR library`
	`clr.AddReference('mcl_ZTM2_NET45')`
	`from mcl_ZTM2_NET45 import USB_Control`
	`MyPTE1 = USB_Control()`
	`MyPTE2 = USB_Control()`

Visual Basic	`Public MyPTE1 As New mcl_ZTM2_NET45.USB_Control` `Public MyPTE2 As New mcl_ZTM2_NET45.USB_Control`
Visual C++	`mcl_ZTM2_NET45::USB_Control ^MyPTE1 = gcnew mcl_ZTM2_NET45::USB_Control();` `mcl_ZTM2_NET45::USB_Control ^MyPTE2 = gcnew mcl_ZTM2_NET45::USB_Control();`
Visual C#	`mcl_ZTM2_NET45.USB_Control MyPTE1 = new mcl_ZTM2_NET45.USB_Control();` `mcl_ZTM2_NET45.USB_Control MyPTE2 = new mcl_ZTM2_NET45.USB_Control();`

MatLab	`MCL_SW = NET.addAssembly('C:\Windows\SysWOW64\mcl_ZTM2_NET45.dll')`
	`MyPTE1 = mcl_ZTM2_NET45.USB_Control`
	`MyPTE2 = mcl_ZTM2_NET45.USB_Control`

Example Declarations using the ActiveX DLL (MCL_ZTM2.dll)

Visual Basic	`Public MyPTE1 As New MCL_ZTM2.USB_Control` `Public MyPTE2 As New MCL_ZTM2.USB_Control`
Visual C++	`MCL_ZTM2::USB_Control ^MyPTE1 = gcnew MCL_ZTM2::USB_Control();` `MCL_ZTM2::USB_Control ^MyPTE2 = gcnew MCL_ZTM2::USB_Control();`
Visual C#	`public MCL_ZTM2.USB_Control MyPTE1 = new MCL_ZTM2.USB_Control();` `public MCL_ZTM2.USB_Control MyPTE2 = new MCL_ZTM2.USB_Control();`
MatLab	`MyPTE1 = actxserver(MCL_ZTM2.USB_Control')` `MyPTE2 = actxserver(MCL_ZTM2.USB_Control')`

5.3. Additional DLL Considerations

Mini-Circuits' DLL API options are intended to support the widest possible range of modern programming environments, with the typical summaries and examples below applying in most cases. There are a few additional considerations to bear in mind for specific programming environments, as summarized below.

5.3.1. 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`

5.3.2. 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 mcl_ZTM2_NET45_USB_Control_Connect(`
	`  mcl_ZTM2_NET45_USB_Control __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)

5.4. 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);`

5.5. 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);`

5.6. 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	`MyPTE1.Disconnect()`
Visual Basic	`MyPTE1.Disconnect()`
Visual C++	`MyPTE1->Disconnect();`
Visual C#	`MyPTE1.Disconnect();`
MatLab	`MyPTE1.Disconnect();`

5.7. 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?", '')`

5.8. 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);`

5.9. 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();`

5.10. 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`

5.11. 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`

6. 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.

6.1. USB Interrupt Code Concept

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

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

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

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

6.2. Send SCPI Command

Sends a SCPI command to the modular test system and collects the response. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the system.

Transmit Array

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

Returned Array

Byte	Data	Description
0	1 or 2 or 42	Interrupt code for Send SCPI Command
1 to (n-1)	SCPI Data	The SCPI return string, one character per byte, represented as ASCII character codes
n	0	Zero value byte to indicate the end of the SCPI return string
(n+1) to 63	Not significant	

Example (Get Model Name)

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

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	77	ASCII character code for M
3	78	ASCII character code for N
4	63	ASCII character code for ?

The returned array for ZTM-999 would be as follows:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	90	ASCII character code for Z
2	84	ASCII character code for T
3	77	ASCII character code for M
4	45	ASCII character code for -
5	57	ASCII character code for 9
6	57	ASCII character code for 9
7	57	ASCII character code for 9
8	0	Zero value byte to indicate end of string

7. Control Options for MacOS

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

The key steps to get started would be as follows.

7.1. Connect & Identify Initial IP Address

For connection into a network supporting DHCP:

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

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

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