PROGRAMMING MANUAL

Signal Generators

SSG 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' SSG series, USB and Ethernet controlled signal generators.

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/sg.html

For details and specifications of individual models please see:

https://www.minicircuits.com/WebStore/RF-Synthesized-Signal-Generators.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 device can use any of the following methods:

1. For Ethernet connected models, using HTTP or Telnet communication over 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 COM 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))

In all cases the full functionality of the system is accessible using a command set based on SCPI (see SCPI Commands).

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. Ethernet Control API

Control of the device via Ethernet TCP / IP networks involves sending the SCPI commands / queries detailed below via HTTP or Telnet. SSH is also available as an option for secure communication with all Mini-Circuits' signal generators except SSG-6000RC & SSG-6001RC.

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.

2.1. Configuring Ethernet Settings

The device's Ethernet IP settings can be configured using the USB connection for all models. Some models also support configuration of the Ethernet IP settings whilst connected by Ethernet. Refer to the Ethernet Configuration section for details. 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.

2.2. 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. Please contact Mini-Circuits for support.

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

2.3. Link-Local / Auto IP Address

A default "link-local" auto IP address will be assumed for the models listed below 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 when the device is connected directly via an Ethernet cable to a PC instead of via a network.

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	Firmware C6 or later
SSG-30G(HP)-RC	Firmware C6 or later
SSG-44G(HP)-RC	All units

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:

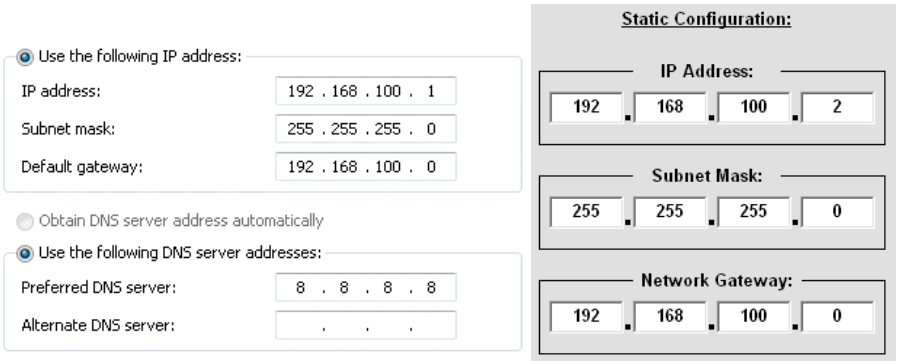
1. Connect the device directly to a PC using the Ethernet cable
2. No DHCP response will be received from the PC so the device will assume the default auto-IP
3. Connect to the device on 169.254.10.10
4. Disable DHCP, set the required static IP configuration and reset the device
5. Reconnect using the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

2.4. 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.

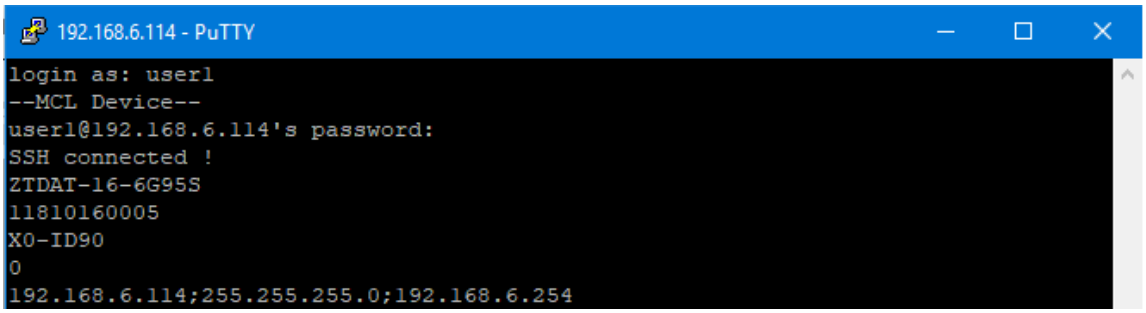


2.5. SSH Communication

SSH communication is supported as standard on all Mini-Circuits' signal generators except SSG-6000RC & SSG-6001RC.

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.



2.6. HTTP Communication

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

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

Where:

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

Examples 1:

```
http://192.168.100.100:800/PWD=123;:FREQ:2105MHZ
```

- The device has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to "123"
- The command is to set the output frequency to 2105 MHz

Examples 2:

```
http://10.10.10.10/:POWER:8.5
```

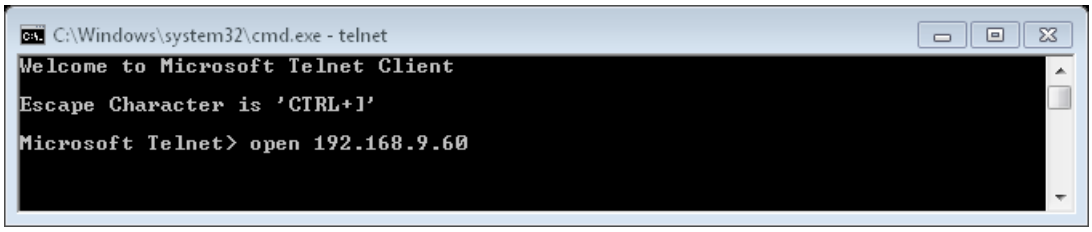
- The device has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to set the output power to +8.5 dBm

2.7. Telnet Communication

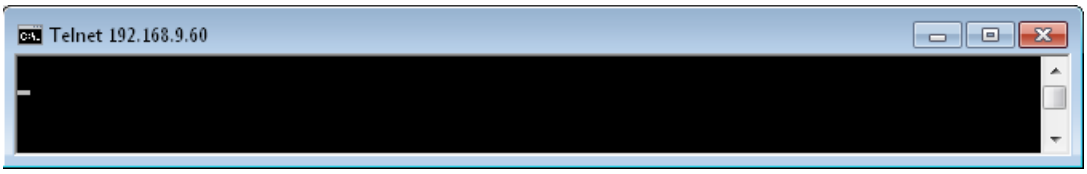
Communication is started by creating a Telnet connection to the device's IP address. On successful connection the "line feed" character will be returned. If the system has a password enabled, this must be sent as the first command after connection.

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:

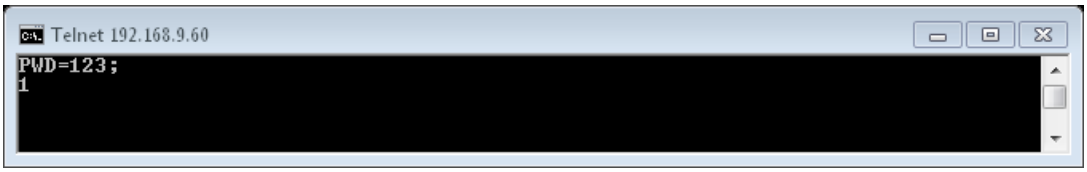
1) Set up Telnet connection to a generator with IP address 192.168.9.60:



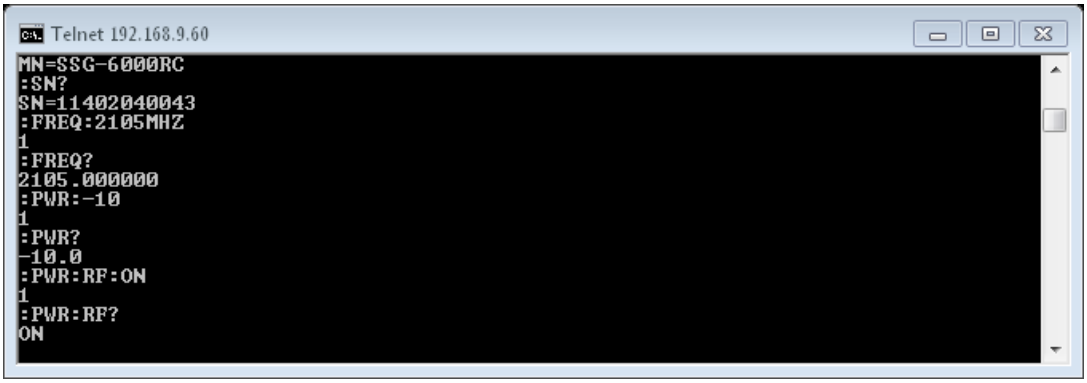
2) The "line feed" character is returned indicating the connection was successful:



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



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



2.8. 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.

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 `MCL_SSG?` 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: `MCL_SSG?`

Received Data:

```
Model Name: SSG-15G-RC
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-D8-03

Model Name: SSG-15G-RC
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-D8-04

Model Name: SSG-15G-RC
Serial Number: 11302120003
IP Address=192.168.9.103 Port: 80
Subnet Mask=255.255.0.0
Network Gateway=192.168.9.0
Mac Address=D0-73-7F-82-D8-05
```

3. SCPI Commands

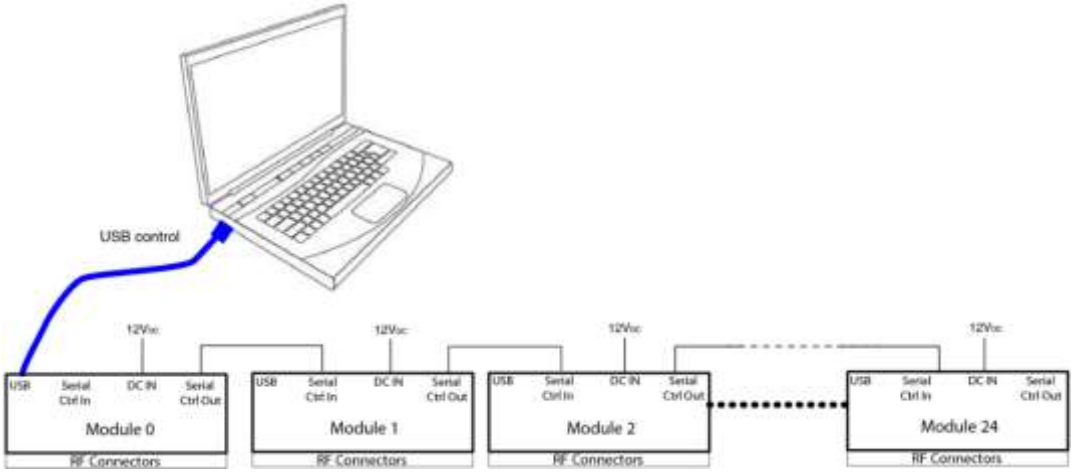
The recommended method for controlling the device is a series of ASCII text commands based on SCPI (Standard Commands for Programmable Instruments). These commands can be sent using any of the APIs detailed in this manual.

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. Daisy-Chain Control of SSG-44G-RC

SSG-44G-RC includes Mini-Circuits' dynamic daisy-chain control concept, to allow cost effective multi-channel signal generators to be configured quickly and without the need to re-write the control software each time.

Multiple signal generators can be cascaded or "daisy-chained" together via the respective Serial In and Out connections. The first generator in the chain is connected via its primary USB or Ethernet control interface and becomes the Master unit, with all daisy-chained modules receiving their instructions and issuing their replies through the Master.



3.1.1. ADDRESSING INDIVIDUAL DAISY-CHAINED MODULES

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

If new generators are subsequently added to the daisy-chain, or the order is changed, the Daisy-Chain - Assign Addresses and Daisy-Chain - Count Number of Slaves commands can be issued to refresh the addresses and check the number of connected modules.

The full list of SCPI control commands / queries for identifying and controlling the modules is summarized in the following sections. These commands can be directed to any specific generator 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 daisy-chained module (address 01)

The address component is not required where only 1 generator is controlled by Ethernet or USB.

3.1.2. ADDRESSING ALL DAISY-CHAINED MODULES

Two global addresses are supported to allow commands to be sent to all modules within the daisy-chain:

AL	Address all modules within the daisy-chain
SL	Address all modules within the daisy-chain except the Master

For example:

:AL:FREQ:2105MHz?	Set all signal generator modules with a CW frequency of 2105 MHz
:SL:FREQ:2105MHz?	Set all signal generator modules except the Master to a CW frequency of 2105 MHz

3.2. SCPI - Daisy-Chain Management

Applies To

Model Name	Requirements
SSG-44G(HP)-RC	All units

3.2.1. DAISY-CHAIN - ASSIGN ADDRESSES

:AssignAddresses

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

Applies To

Model Name	Requirements
SSG-44G(HP)-RC	All units

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:AssignAddresses	1

3.2.2. 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.

Applies To

Model Name	Requirements
SSG-44G(HP)-RC	All units

Return String

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

Examples

String to Send	String Returned
:NumberOfSlaves?	1

3.3. SCPI - Identification

3.3.1. GET MODEL NAME

:MN?

Return the Mini-Circuits model name.

Note: For SSG-30G(HP)-RC signal generators, only the model name of the core SSG-15G-RC signal source module will be returned. Use :FX30:CMD::MN? to identify the frequency extender / buffer module.

Return Value

MN=[model]

Value	Description
[model]	Model name of the connected device

Examples

String to Send	String Returned
:MN?	MN=SSG-15G-RC
:FX30:CMD::MN?	MN=FX-30GHP-RC

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

3.3.2. GET SERIAL NUMBER

:SN?

Returns the serial number.

Note: For SSG-30G(HP)-RC signal generators, only the model name of the core SSG-15G-RC signal source module will be returned. Use `:FX30:CMD::MN?` to identify the frequency extender / buffer module.

Return Value

`SN=[serial]`

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

Examples

String to Send	String Returned
`:SN?`	`SN=12208010025`
`:FX30:CMD::SN?`	`SN=12209030037`

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

3.3.3. GET FIRMWARE

FIRMWARE?

Returns the internal firmware version.

Note: For SSG-30G(HP)-RC signal generators, only the model name of the core SSG-15G-RC signal source module will be returned. Use `:FX30:CMD::MN?` to identify the frequency extender / buffer module.

Return Value

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

Examples

String to Send	String Returned
`FIRMWARE?`	`B3`
`:FX30:CMD::FIRMWARE?`	`A0`

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

3.3.4. CHECK INTERNAL TEMPERATURE

:TSENSOR?

Returns the temperature (degrees Celsius) from the generator's internal temperature sensor.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[temperature]	The internal temperature in degrees Celsius

Example

String to Send	String Returned
:TSENSOR?	+24.25

HTTP Implementation: http://10.10.10.10/:TSENSOR?

3.3.5. CHECK SSG-30G(HP)-RC OPERATION

:FX-30_CONNECTED?

SSG-30G-RC and SSG-30GHP-RC signal generators are constructed from an SSG-15G-RC signal source module with an FX-30G(HP)-RC frequency extender module which operate as a single 30 GHz signal generator when connected.

This query is used to confirm that the FX-30G(HP)-RC module is connected and that the system is therefore operating as SSG-30G(HP)-RC.

Note: Only the core SSG-15G-RC signal source module will respond to the standard identification queries (model name, serial number and firmware). The identification queries can be prepended with `:FX30:CMD:` to query the FX-30G(HP)-RC module directly.

Applies To

SSG-30G-RC & SSG-30GHP-RC

Return Value

Value	Description
0	FX-30G(HP)-RC module not found, therefore the signal generator is operating as SSG-15G-RC
1	FX-30G(HP)-RC module found, therefore the signal generator is operating as SSG-30G(HP)-RC

Examples

The sequence below would indicate that SSG-30GHP-RC is fully connected and operational:

String to Send	String Returned
:MN?	MN=SSG-15G-RC
:FX30:CMD::MN?	MN=FX-30GHP-RC
:FX-30_CONNECTED?	1

HTTP Implementation: http://10.10.10.10/:FX-30_CONNECTED?

3.3.6. SET USB ADDRESS

:ADD:[address]

Sets a numeric address (1-255) for the signal generator which can be used to identify and connect specific models in place of serial number when using the DLL for USB control. The default is 255 on all units.

Parameters

Value	Description
[address]	Numeric address value from 1 to 255

Return Value

Value	Description
0	Command failed (check the address is valid
1	Command completed successfully

Example

String to Send	String Returned
:ADD:10	1

HTTP Implementation: http://10.10.10.10/:ADD:10

3.3.7. GET USB ADDRESS

:ADD?

Returns the numeric address (1-255) for the signal generator. The address can be used to identify and connect specific models in place of serial number when using the DLL for USB control. The default is 255 on all units.

Return Value

Value	Description
[address]	Numeric address value from 1 to 255

Example

String to Send	String Returned
:ADD?	10

HTTP Implementation: http://10.10.10.10/:ADD?

3.4. SCPI – Operation Timers

3.4.1. GET ON/OFF COUNTER

:ONOFFCOUNTER?

Returns a timer value indicating the number of times that the signal generator has been powered on in its lifetime.

Return Value

Variable	Description
[count]	The power on count

Examples

String to Send	String Returned
:ONOFFCOUNTER?	150

HTTP Implementation: http://10.10.10.10/:ONOFFCOUNTER?

3.4.2. GET OPERATION TIMER

:OPERATIONTIME?

Returns a timer value indicating the number of minutes that the signal generator has been powered on in its lifetime.

Return Value

[timer] minutes

Variable	Description
[timer]	The total "on" time of the generator in minutes

Examples

String to Send	String Returned
:OPERATIONTIME?	7503 minutes

HTTP Implementation: http://10.10.10.10/:OPERATIONTIME?

3.5. SCPI – CW Output

3.5.1. SET RF OUTPUT ON/OFF

:PWR:RF:[state]

Enable or disable the RF output after setting the required frequency / power / pulse parameters.

Parameters

Value	Description
OFF	Disable the RF output
ON	Enable the RF output

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Example

String to Send	String Returned
:PWR:RF:OFF	1
:PWR:RF:ON	1

HTTP Implementation: http://10.10.10.10/:PWR:RF:OFF

3.5.2. QUERY RF OUTPUT STATE (ON/OFF)

:PWR:RF?

Query the RF output state.

Return Value

Value	Description
OFF	RF output is disabled
ON	RF output is enabled

Example

String to Send	String Returned
:PWR:RF?	OFF

HTTP Implementation: http://10.10.10.10/:PWR:RF?

3.5.3. CHECK EXTERNAL REFERENCE

:EXTREFDETECT?

Indicates whether a valid external reference signal has been detected in the Ref In port. The signal generator will automatically switch to using the external reference when available.

Return Value

Value	Description
0	External reference not detected; internal reference in use
1	External reference detected and in use

Example

String to Send	String Returned
:EXTREFDETECT?	0

HTTP Implementation: http://10.10.10.10/:EXTREFDETECT?

3.5.4. SET TRIGGER OUT

:TRIGGEROUT:STATE:[mode]

Sets the generator's Trigger Out port to logic low, logic high, or 5ms pulses.

Parameters

Value	Description
HIGH	Set Trigger Out to logic high
LOW	Set Trigger Out to logic low
PULSE	Pulse Trigger Out (5ms at logic high, 5ms at logic low)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:TRIGGEROUT:STATE:HIGH	1
:TRIGGEROUT:STATE:LOW	1
:TRIGGEROUT:STATE:PULSE	1

HTTP Implementation: http://10.10.10.10/:TRIGGEROUT:STATE:HIGH

3.5.5. GET TRIGGER IN

:TRIGGERIN:STATE?

Returns the logic state of the generator's Trigger In port (logic low or logic high).

Return Value

Value	Description
1	Trigger In is logic high
0	Trigger In is logic low

Examples

String to Send	String Returned
:TRIGGERIN:STATE?	1
:TRIGGERIN:STATE?	0

HTTP Implementation: http://10.10.10.10/:TRIGGERIN?

3.5.6. SET FREQUENCY

:FREQ:[freq][units]

Sets the CW output frequency.

Parameters

Variable	Value	Description
[freq]		The CW output frequency to set
[units]	Hz	Set the frequency in Hertz
	kHz	Set the frequency in kilohertz
	MHz	Set the frequency in megahertz
	GHz	Set the frequency in gigahertz

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FREQ:2.105GHz	Frequency Set
:FREQ:2105MHz	Frequency Set

HTTP Implementation: http://10.10.10.10/:FREQ:2.105GHz

See Also

Set RF Output On/Off

3.5.7. GET FREQUENCY

:FREQ?

Returns the CW output frequency setting.

Return Value

Value	Description
[freq]	CW output frequency of the generator in MHz

Examples

String to Send	String Returned
:FREQ?	2105.000000

HTTP Implementation: http://10.10.10.10/:FREQ?

See Also

Query RF Output State (On/Off)

3.5.8. QUERY FREQUENCY SPECIFICATION

:FREQ:[spec]?

Returns an output frequency specification of the signal generator; either the minimum / maximum limits, or minimum step size.

Parameters

Value	Description
MIN	Return the minimum output frequency specification
MAX	Return the maximum output frequency specification
STEP	Return the minimum step size specification

Return Value

Value	Description
[spec]	The relevant frequency specification including units

Examples

String to Send	String Returned
:FREQ:MAX?	6000.00 MHz
:FREQ:STEP	3.00 Hz

HTTP Implementation: http://10.10.10.10/:FREQ:MAX?

3.5.9. SET POWER

:PWR:[power]?

Sets the CW output power

Parameters

Value	Description
[power]	The CW output power in dBm

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Example

String to Send	String Returned
:PWR:-10.25	1

HTTP Implementation: http://10.10.10.10/:PWR:-10.25

See Also

Set RF Output On/Off

3.5.10. QUERY POWER

:PWR?

Returns the CW output power setting.

Return Value

Value	Description
[power]	CW output power of the generator in dBm

Examples

String to Send	String Returned
:PWR?	-10.25

HTTP Implementation: http://10.10.10.10/:PWR?

See Also

Query RF Output State (On/Off)

3.5.11. QUERY POWER SPECIFICATION

:PWR:[spec]?

Returns an output power specification of the signal generator, either the minimum or maximum guaranteed limit.
Note: Some models are capable of setting a power level slightly beyond the minimum / maximum guaranteed levels. Refer to the model datasheet for guidance on typical performance.

Parameters

Value	Description
MIN	Return the minimum output power specification
MAX	Return the maximum output power specification

Return Value

Value	Description
[spec]	The relevant power value in dBm

Examples

String to Send	String Returned
:PWR:MAX?	14.0
:PWR:STEP	-65.0

HTTP Implementation: http://10.10.10.10/:PWR:MAX?

3.6. SCPI - Power-Up Settings

These settings determine the initial output frequency, power level and state (on/off) that the signal generator will load when the DC power is connected.

The following models are supported:

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

3.6.1. SET POWER-UP MODE

:ONPOWERUP:CONFIG:[mode]

Sets the generator's power-up mode to determine the initial output frequency and power level to be loaded when powered on.
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Parameters

Value	Description
D	Factory default (maximum frequency and minimum power)
L	Last known frequency and power (these parameters are saved to permanent memory every 3 minutes, when the GUI application is closed, or when the Save Data command is issued)
U	User defined frequency and power

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ONPOWERUP:CONFIG:D	1
:ONPOWERUP:CONFIG:L	1
:ONPOWERUP:CONFIG:U	1

DLL Implementation: SCPI_Command(":ONPOWERUP:CONFIG:L")
Ethernet Implementation: :ONPOWERUP:CONFIG:L

3.6.2. GET POWER-UP MODE

:ONPOWERUP:CONFIG?

Returns the generator's power-up mode.
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Return Value

Value	Description
D	Factory default (maximum frequency and minimum power)
L	Last known frequency and power (these parameters are saved to permanent memory every 3 minutes, when the GUI application is closed, or when the Save Data command is issued)
U	User defined frequency and power

Examples

String to Send	String Returned
:ONPOWERUP:CONFIG?	D
:ONPOWERUP:CONFIG?	L
:ONPOWERUP:CONFIG?	U

DLL Implementation: SCPI_Query("ONPOWERUP:CONFIG?", RetSTR)
Ethernet Implementation: :ONPOWERUP:CONFIG?

3.6.3. SET POWER-UP FREQUENCY

:ONPOWERUP:FREQ:[freq]

Sets the initial output frequency to be loaded when the generator's DC power is connected. This only applies when the power-up mode is set to "U" (user defined).
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Parameters

Variable	Description
[freq]	Initial output frequency (MHz) to load when the generator is set to "user defined" power-up mode.

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ONPOWERUP:FREQ:1000.5	1

DLL Implementation: SCPI_Command(":ONPOWERUP:FREQ:1000.5")
Ethernet Implementation: :ONPOWERUP:FREQ:1000.5

3.6.4. GET POWER-UP FREQUENCY

:ONPOWERUP:FREQ?

Returns the initial output frequency that will be loaded when the generator's DC power is connected. This only applies when the power-up mode is set to "U" (user defined).
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Return Value

Variable	Description
[freq]	Initial output frequency (MHz) that will load when the generator is set to "user defined" power-up mode.

Examples

String to Send	String Returned
:ONPOWERUP:FREQ?	1000.5

DLL Implementation: SCPI_Query("ONPOWERUP:FREQ?", RetSTR)
Ethernet Implementation: :ONPOWERUP:FREQ?

3.6.5. SET POWER-UP POWER LEVEL

:ONPOWERUP:PWR:[power]

Sets the initial RF output power level to be loaded when the generator's DC power is connected. This only applies when the power-up mode is set to "U" (user defined).
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Parameters

Variable	Description
[power]	Initial output power level (dBm) to load when the generator is set to "user defined" power-up mode.

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ONPOWERUP:PWR:-3.25	1

DLL Implementation: SCPI_Command(":ONPOWERUP:PWR:-3.25")
Ethernet Implementation: :ONPOWERUP:PWR:-3.25

3.6.6. GET POWER-UP POWER LEVEL

:ONPOWERUP:PWR?

Returns the initial RF output power level that will be loaded when the generator's DC power is connected. This only applies when the power-up mode is set to "U" (user defined).
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Return Value

Variable	Description
[power]	Initial output power level (dBm) that will load when the generator is set to "user defined" power-up mode.

Examples

String to Send	String Returned
:ONPOWERUP:PWR?	-3.25

DLL Implementation: SCPI_Query("ONPOWERUP:PWR?", RetSTR)
Ethernet Implementation: :ONPOWERUP:PWR?

3.6.7. SET POWER-UP OUTPUT STATE

:ONPOWERUP:PWRSTATE:[state]

Sets the initial RF output state that applies when the generator's DC power is connected. The RF output can be enabled or disabled.
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Parameters

Value	Description
OFF	RF output will be disabled on power-up (recommended)
ON	RF output will be enabled on power up

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ONPOWERUP:PWRSTATE:OFF	1
:ONPOWERUP:PWRSTATE:ON	1

DLL Implementation: SCPI_Command(":ONPOWERUP:PWRSTATE:OFF")
Ethernet Implementation: :ONPOWERUP:PWRSTATE:OFF

3.6.8. GET POWER-UP OUTPUT STATE

:ONPOWERUP:PWRSTATE?

Returns the initial RF output state that applies when the generator's DC power is connected. The RF output can be enabled or disabled.
Note: By default, the RF output is disabled on power-up in order to avoid potential damage to the generator output and load in the event of excessive power or a mismatched load.

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Return Value

Value	Description
OFF	RF output will be disabled on power-up
ON	RF output will be enabled on power up

Examples

String to Send	String Returned
:ONPOWERUP:PWRSTATE?	OFF
:ONPOWERUP:PWRSTATE?	ON

DLL Implementation: SCPI_Query("ONPOWERUP:PWRSTATE?", RetSTR)
Ethernet Implementation: :ONPOWERUP:PWRSTATE?

3.6.9. SAVE STATE

:OPERATIONDATA:SAVE

Saves the latest CW output state (frequency and power) to permanent memory so that the settings can be recalled then next time the generator is powered on. This only applies when the power-up mode is set to "L" (last known).

Applies To

Model Name	Firmware Requirement
SSG-6000RC	Firmware A4 or later
SSG-15G-RC	Firmware B6 or later

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:OPERATIONDATA:SAVE	1

DLL Implementation: SCPI_Command(":OPERATIONDATA:SAVE")
Ethernet Implementation: :OPERATIONDATA:SAVE

3.7. SCPI - Frequency Sweep

The signal generator can be configured to produce a fast, automatic frequency sweep sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python

```
# Set sweep for 1000-6000MHz in 100MHz steps
MyPTE1.SCPI_Command(":FSWEEP:STARTFREQ:1000")
MyPTE1.SCPI_Command(":FSWEEP:STOPFREQ:6000")
MyPTE1.SCPI_Command(":FSWEEP:STEPSIZE:100")
# Set fixed 10dBm output power level and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":FSWEEP:POWER:10")
MyPTE1.SCPI_Command(":FSWEEP:DWELL:10")
MyPTE1.SCPI_Command(":PWR:RF:ON") # Enable RF & start the sweep
MyPTE1.SCPI_Command(":FSWEEP:MODE:ON")
```

Visual Basic

```
' Set sweep for 1000-6000MHz in 100MHz steps
MyPTE1.SCPI_Command(":FSWEEP:STARTFREQ:1000")
MyPTE1.SCPI_Command(":FSWEEP:STOPFREQ:6000")
MyPTE1.SCPI_Command(":FSWEEP:STEPSIZE:100")
' Set fixed 10dBm output power level and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":FSWEEP:POWER:10")
MyPTE1.SCPI_Command(":FSWEEP:DWELL:10")
MyPTE1.SCPI_Command(":PWR:RF:ON") ' Enable RF & start the sweep
MyPTE1.SCPI_Command(":FSWEEP:MODE:ON")
```

Visual C++

```
// Set sweep for 1000-6000MHz in 100MHz steps
MyPTE1->SCPI_Command(":FSWEEP:STARTFREQ:1000");
MyPTE1->SCPI_Command(":FSWEEP:STOPFREQ:6000");
MyPTE1->SCPI_Command(":FSWEEP:STEPSIZE:100");
// Set fixed 10dBm output power level and 10ms dwell time for the sweep
MyPTE1->SCPI_Command(":FSWEEP:POWER:10");
MyPTE1->SCPI_Command(":FSWEEP:DWELL:10");
MyPTE1->SCPI_Command(":PWR:RF:ON"); // Enable RF & start the sweep
MyPTE1->SCPI_Command(":FSWEEP:MODE:ON");
```

Visual C#

```
// Set sweep for 1000-6000MHz in 100MHz steps
MyPTE1.SCPI_Command(":FSWEEP:STARTFREQ:1000");
MyPTE1.SCPI_Command(":FSWEEP:STOPFREQ:6000");
MyPTE1.SCPI_Command(":FSWEEP:STEPSIZE:100");
// Set fixed 10dBm output power level and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":FSWEEP:POWER:10");
MyPTE1.SCPI_Command(":FSWEEP:DWELL:10");
MyPTE1.SCPI_Command(":PWR:RF:ON"); // Enable RF & start the sweep
MyPTE1.SCPI_Command(":FSWEEP:MODE:ON");
```

MatLab	% Set sweep for 1000-6000MHz in 100MHz steps MyPTE1.SCPI_Command(":FSWEEP:STARTFREQ:1000") MyPTE1.SCPI_Command(":FSWEEP:STOPFREQ:6000") MyPTE1.SCPI_Command(":FSWEEP:STEPSIZE:100") % Set fixed 10dBm output power level and 10ms dwell time for the sweep MyPTE1.SCPI_Command(":FSWEEP:POWER:10") MyPTE1.SCPI_Command(":FSWEEP:DWELL:10") MyPTE1.SCPI_Command(":PWR:RF:ON") % Enable RF & start the sweep MyPTE1.SCPI_Command(":FSWEEP:MODE:ON")

Example fast sequence using HTTP for Ethernet control:

- http://10.10.10.10/:FSWEEP:STARTFREQ:1000
- http://10.10.10.10/:FSWEEP:STOPFREQ:6000
- http://10.10.10.10/:FSWEEP:STEPSIZE:100
- http://10.10.10.10/:FSWEEP:POWER:10
- http://10.10.10.10/:FSWEEP:DWELL:10
- http://10.10.10.10/:PWR:RF:ON
- http://10.10.10.10/:FSWEEP:MODE:ON

3.7.1. FREQUENCY SWEEP – SET FREQUENCIES

:FSWEEP:[parameter]:[frequency]

This function sets the start, stop or step frequency for the sweep (in MHz).

Parameters

Variable	Value	Description
[parameter]	STARTFREQ	Set the start frequency for the sweep
	STOPFREQ	Set the stop frequency for the sweep
	STEPSIZE	Set the step frequency for the sweep
[frequency]		The frequency to set in MHz

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FSWEEP:STARTFREQ:10.50	1
:FSWEEP:STOPFREQ:5500	1
:FSWEEP:STEPSIZE:0.25	1

DLL Implementation: SCPI_Command(":FSWEEP:STARTFREQ:10.50")
HTTP Implementation: http://10.10.10.10/:FSWEEP:STARTFREQ:10.50

3.7.2. FREQUENCY SWEEP – GET FREQUENCIES

:FSWEEP:[parameter]?

This function returns the start, stop or step frequency for the sweep (in MHz).

Parameters

Value	Description
STARTFREQ	Return the start frequency for the sweep
STOPFREQ	Return the stop frequency for the sweep
STEPSIZE	Return the step frequency for the sweep

Return Value

Variable	Description
[freq]	The frequency in MHz

Examples

String to Send	String Returned
:FSWEEP:STARTFREQ?	1000.000000
:FSWEEP:STOPFREQ?	2000.000000
:FSWEEP:STEPSIZE?	25.000000

DLL Implementation: `SCPI_Query(":FSWEEP:STARTFREQ?", RetSTR)`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:STARTFREQ?`

3.7.3. FREQUENCY SWEEP – SET POWER

:FSWEEP:POWER:[power]

This function sets a constant power level (in dBm) for a frequency sweep.

Parameters

Variable	Description
[power]	The power to set in dBm

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FSWEEP:POWER:10.50	1

DLL Implementation: `SCPI_Command(":FSWEEP:POWER:10.50")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:POWER:10.50`

3.7.4. FREQUENCY SWEEP – GET POWER

:FSWEEP:POWER?

This function returns the constant output power level (in dBm) for a frequency sweep.

Return Value

Variable	Description
[power]	The output power level in dBm

Examples

String to Send	String Returned
:FSWEEP:POWER?	-10.5

DLL Implementation: `SCPI_Query(":FSWEEP:POWER?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:POWER?`

3.7.5. FREQUENCY SWEEP – SET DWELL TIME

:FSWEEP:DWELL:[time]

This function sets the dwell time to be used in a frequency sweep (the time in milliseconds for the generator to pause at each frequency point).

Parameters

Variable	Description
[time]	The dwell time in ms

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FSWEEP:DWELL:100	1

DLL Implementation: `SCPI_Command(":FSWEEP:DWELL:100")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:DWELL:100`

3.7.6. FREQUENCY SWEEP – GET DWELL TIME

:FSWEEP:DWELL?

This function returns the dwell time to be used in a frequency sweep (the time in milliseconds that the generator will pause at each frequency point).

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:FSWEEP:DWELL?	100

DLL Implementation: `SCPI_Query(":FSWEEP:DWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:DWELL?`

3.7.7. FREQUENCY SWEEP – GET DWELL TIME SPECIFICATION

:FSWEEP:[spec]?

This function returns the minimum or maximum dwell time specifications for the generator.

Parameters

Value	Description
MAXDWELL	Return the maximum specified dwell time
MINDWELL	Return the minimum specified dwell time

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:FSWEEP:MINDWELL?	20
:FSWEEP:MAXDWELL?	10000

DLL Implementation: `SCPI_Query(":FSWEEP:MINDWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:MINDWELL?`

3.7.8. FREQUENCY SWEEP – SET DIRECTION

:FSWEEP:DIRECTION:[mode]

This function sets the direction of a frequency sweep.

Parameters

Value	Description
0	Forward (sweep from start to stop frequency). This is the default.
1	Reverse (sweep from stop to start frequency)
2	Bi-directional (sweep from start to stop, then stop to start frequency)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:FSWEEP:DIRECTION:1`	`1`

DLL Implementation: `SCPI_Command(":FSWEEP:DIRECTION:1")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:DIRECTION:1`

3.7.9. FREQUENCY SWEEP – GET DIRECTION

:FSWEEP:DIRECTION?

This function returns the direction of a frequency sweep.

Return Value

Value	Description
0	Forward (sweep from start to stop frequency)
1	Reverse (sweep from stop to start frequency)
2	Bi-directional (sweep from start to stop, then stop to start frequency)

Examples

String to Send	String Returned
`:FSWEEP:DIRECTION?`	`0`

DLL Implementation: `SCPI_Query(":FSWEEP:DIRECTION?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:DIRECTION?`

3.7.10. FREQUENCY SWEEP – SET TRIGGER IN MODE

:FSWEEP:TRIGGERIN:[mode]

This function specifies how the frequency sweep should respond to an external trigger. The modes are:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each frequency point
2 – Wait for external trigger (Trigger In = logic 1) before starting each frequency sweep

Parameters

Value	Description
0	Ignore trigger input (default)
1	Wait for trigger before setting each frequency
2	Wait for trigger before starting each sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FSWEEP:TRIGGERIN:1	1

DLL Implementation: `SCPI_Command(":FSWEEP:TRIGGERIN:1")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:TRIGGERIN:1`

3.7.11. FREQUENCY SWEEP – SET TRIGGER OUT MODE

:FSWEEP:TRIGGEROUT:[mode]

This function specifies how the Trigger Out port will be used during the frequency sweep. The modes are:
0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) as each frequency point is set
2 – Provide a trigger output (Trigger Out = logic 1) as each frequency sweep is initiated

Parameters

Value	Description
0	Trigger output disabled (default)
1	Set Trigger Out on setting each frequency
2	Set Trigger Out on starting each sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:FSWEEP:TRIGGEROUT:1`	`1`

DLL Implementation: `SCPI_Command(":FSWEEP:TRIGGEROUT:1")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:TRIGGEROUT:1`

3.7.12. FREQUENCY SWEEP – GET TRIGGER IN MODE

:FSWEEP:TRIGGERIN?

This function returns a code to indicate how the frequency sweep will respond to an external trigger. The modes are:
0 – Trigger input ignored
1 – Wait for external trigger (Trigger In = logic 1) before setting each frequency point
2 – Wait for external trigger (Trigger In = logic 1) before starting each frequency sweep

Return Value

Value	Description
0	Ignore trigger input
1	Wait for trigger before setting each frequency
2	Wait for trigger before starting each sweep

Examples

String to Send	String Returned
:FSWEEP:TRIGGERIN?	0

DLL Implementation: SCPI_Query(":FSWEEP:TRIGGERIN?", RetStr)
HTTP Implementation: http://10.10.10.10/:FSWEEP:TRIGGERIN?

3.7.13. FREQUENCY SWEEP – GET TRIGGER OUT MODE

:FSWEEP:TRIGGEROUT?

This function returns a code to indicate how the Trigger Out port will be used during the frequency sweep. The modes are:
0 – Trigger output disabled
1 – Provide a trigger output (Trigger Out = logic 1) as each frequency point is set
2 – Provide a trigger output (Trigger Out = logic 1) as each frequency sweep is initiated

Return Value

Value	Description
0	Trigger output disabled
1	Set Trigger Out on setting each frequency
2	Set Trigger Out on starting each sweep

Examples

String to Send	String Returned
:FSWEEP:TRIGGEROUT?	0

DLL Implementation: SCPI_Query(":FSWEEP:TRIGGEROUT?", RetStr)
HTTP Implementation: http://10.10.10.10/:FSWEEP:TRIGGEROUT?

3.7.14. FREQUENCY SWEEP – START/STOP SWEEP

:FSWEEP:MODE:[mode]

This function starts or stops the frequency sweep using the previously defined parameters.

Parameters

Value	Description
ON	Start frequency sweep
OFF	Stop frequency sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:FSWEEP:MODE:ON	1

DLL Implementation: `SCPI_Command(":FSWEEP:MODE:ON")`
HTTP Implementation: `http://10.10.10.10/:FSWEEP:MODE:ON`

3.8. SCPI - Power Sweep

The signal generator can be configured to produce a fast, automatic power sweep sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python

```
# Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1.SCPI_Command(":PSWEEP:STARTPOWER:-20")
MyPTE1.SCPI_Command(":PSWEEP:STOPPOWER:20")
MyPTE1.SCPI_Command(":PSWEEP:STEPSIZE:0.5")
# Set fixed 1000MHz frequency and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":PSWEEP:FREQ:1000")
MyPTE1.SCPI_Command(":PSWEEP:DWELL:10")
MyPTE1.SCPI_Command(":PWR:RF:ON") # Enable the output & start the sweep
MyPTE1.SCPI_Command(":PSWEEP:MODE:ON")
```

Visual Basic

```
' Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1.SCPI_Command(":PSWEEP:STARTPOWER:-20")
MyPTE1.SCPI_Command(":PSWEEP:STOPPOWER:20")
MyPTE1.SCPI_Command(":PSWEEP:STEPSIZE:0.5")
' Set fixed 1000MHz frequency and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":PSWEEP:FREQ:1000")
MyPTE1.SCPI_Command(":PSWEEP:DWELL:10")
MyPTE1.SCPI_Command(":PWR:RF:ON") ' Enable the output & start the sweep
MyPTE1.SCPI_Command(":PSWEEP:MODE:ON")
```

Visual C++

```
// Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1->SCPI_Command(":PSWEEP:STARTPOWER:-20");
MyPTE1->SCPI_Command(":PSWEEP:STOPPOWER:20");
MyPTE1->SCPI_Command(":PSWEEP:STEPSIZE:0.5");
// Set fixed 1000MHz frequency and 10ms dwell time for the sweep
MyPTE1->SCPI_Command(":PSWEEP:FREQ:1000");
MyPTE1->SCPI_Command(":PSWEEP:DWELL:10");
MyPTE1->SCPI_Command(":PWR:RF:ON"); // Enable the output & start the sweep
MyPTE1->SCPI_Command(":PSWEEP:MODE:ON");
```

Visual C#

```
// Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1.SCPI_Command(":PSWEEP:STARTPOWER:-20");
MyPTE1.SCPI_Command(":PSWEEP:STOPPOWER:+20");
MyPTE1.SCPI_Command(":PSWEEP:STEPSIZE:0.5");
// Set fixed 1000MHz frequency and 10ms dwell time for the sweep
MyPTE1.SCPI_Command(":PSWEEP:FREQ:1000");
MyPTE1.SCPI_Command(":PSWEEP:DWELL:10");
MyPTE1.SCPI_Command(":PWR:RF:ON"); // Enable the output & start the sweep
MyPTE1.SCPI_Command(":PSWEEP:MODE:ON");
```

MatLab	% Set sweep for -20dBm to +20dBm in 0.5dB steps MyPTE1.SCPI_Command(":PSWEEP:STARTPOWER:-20") MyPTE1.SCPI_Command(":PSWEEP:STOPPOWER:20") MyPTE1.SCPI_Command(":PSWEEP:STEPSIZE:0.5") % Set fixed 1000MHz frequency and 10ms dwell time for the sweep MyPTE1.SCPI_Command(":PSWEEP:FREQ:1000") MyPTE1.SCPI_Command(":PSWEEP:DWELL:10") MyPTE1.SCPI_Command(":PWR:RF:ON") % Enable the output & start the sweep MyPTE1.SCPI_Command(":PSWEEP:MODE:ON")

Example fast sequence using HTTP for Ethernet control:

- http://10.10.10.10/:PSWEEP:STARTPOWER:-20
- http://10.10.10.10/:PSWEEP:STOPPOWER:20
- http://10.10.10.10/:PSWEEP:STEPSIZE:0.5
- http://10.10.10.10/:PSWEEP:FREQ:1000
- http://10.10.10.10/:PSWEEP:DWELL:10
- http://10.10.10.10/:PWR:RF:ON
- http://10.10.10.10/:PSWEEP:MODE:ON

3.8.1. POWER SWEEP – SET POWER

:PSWEEP:[parameter]:[power]

This function sets the start, stop or step power for the sweep (in dBm).

Parameters

Variable	Value	Description
[parameter]	STARTPOWER	Set the start power for the sweep
	STOPPOWER	Set the stop power for the sweep
	STEPSIZE	Set the power step for the sweep
[power]		The power to set in dBm

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:STARTPOWER:10.50	1
:PSWEEP:STOPPOWER:5500	1
:PSWEEP:STEPSIZE:0.25	1

DLL Implementation: SCPI_Command(":PSWEEP:STARTPOWER:10.50")
HTTP Implementation: http://10.10.10.10/:PSWEEP:STARTPOWER:10.50

3.8.2. POWER SWEEP – GET POWER

:PSWEEP:[parameter]?

This function returns the start, stop or step power for the sweep (in dBm).

Parameters

Value	Description
STARTPOWER	Return the start power for the sweep in dBm
STOPPOWER	Return the stop power for the sweep in dBm
STEPSIZE	Return the power step for the sweep in dBm

Return Value

Variable	Description
[power]	Power in dBm

Examples

String to Send	String Returned
:PSWEEP:STARTPOWER?	-20
:PSWEEP:STOPPOWER?	20
:PSWEEP:STEPSIZE?	0.5

DLL Implementation: `SCPI_Query(":PSWEEP:STARTPOWER?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:STARTPOWER?`

3.8.3. POWER SWEEP – SET FREQUENCY

:PSWEEP:FREQ:[frequency]

This function sets a constant frequency (in MHz) for a power sweep.

Parameters

Variable	Description
[frequency]	The frequency to set in MHz

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:FREQ:1050.5	1

DLL Implementation: SCPI_Command(":PSWEEP:FREQ:1050.5")
HTTP Implementation: http://10.10.10.10/:PSWEEP:FREQ:1050.5

3.8.4. POWER SWEEP – GET FREQUENCY

:PSWEEP:FREQ?

This function returns the constant frequency (in MHz) for a power sweep.

Return Value

Variable	Description
[frequency]	The frequency in MHz

Examples

String to Send	String Returned
:PSWEEP:FREQ?	1050.500000

DLL Implementation: SCPI_Command(":PSWEEP:FREQ?
HTTP Implementation: http://10.10.10.10/:PSWEEP:FREQ?

3.8.5. POWER SWEEP – SET DWELL TIME

:PSWEEP:DWELL:[time]

This function sets the dwell time to be used in a power sweep (the time in milliseconds for the generator to pause at each power point).

Parameters

Variable	Description
[time]	The dwell time in ms

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:DWELL:100	1

DLL Implementation: `SCPI_Command(":PSWEEP:DWELL:100")`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:DWELL:100`

3.8.6. POWER SWEEP – GET DWELL TIME

:PSWEEP:DWELL?

This function returns the dwell time to be used in a power sweep (the time in milliseconds for the generator to pause at each power point).

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:PSWEEP:DWELL?	20

DLL Implementation: `SCPI_Query(":PSWEEP:DWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:DWELL?`

3.8.7. POWER SWEEP – GET DWELL TIME SPECIFICATION

:PSWEEP:[spec]?

This function returns the minimum or maximum dwell time specifications for the generator.

Parameters

Value	Description
MAXDWELL	Return the maximum specified dwell time
MINDWELL	Return the minimum specified dwell time

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:PSWEEP:MINDWELL?	20
:PSWEEP:MAXDWELL?	10000

DLL Implementation: `SCPI_Query(":PSWEEP:MINDWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:MINDWELL?`

3.8.8. POWER SWEEP – SET DIRECTION

:PSWEEP:DIRECTION:[mode]

This function sets the direction of a power sweep.

Parameters

Value	Description
0	Forward (sweep from start to stop power). This is the default.
1	Reverse (sweep from stop to start power)
2	Bi-directional (sweep from start to stop, then stop to start power)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:PSWEEP:DIRECTION:1`	`1`

DLL Implementation: `SCPI_Command(":PSWEEP:DIRECTION:1")`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:DIRECTION:1`

3.8.9. POWER SWEEP – GET DIRECTION

:PSWEEP:DIRECTION?

This function returns the direction of a power sweep.

Return Value

Value	Description
0	Forward (sweep from start to stop power)
1	Reverse (sweep from stop to start power)
2	Bi-directional (sweep from start to stop, then stop to start power)

Examples

String to Send	String Returned
`:PSWEEP:DIRECTION?`	`0`

DLL Implementation: `SCPI_Query(":PSWEEP:DIRECTION?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:DIRECTION?`

3.8.10. POWER SWEEP – SET TRIGGER IN MODE

:PSWEEP:TRIGGERIN:[mode]

This function specifies how the power sweep should respond to an external trigger. The modes are:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each point
2 – Wait for external trigger (Trigger In = logic 1) before starting each sweep

Parameters

Value	Description
0	Ignore trigger input (default)
1	Wait for trigger before setting each power
2	Wait for trigger before starting each sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:TRIGGERIN:1	1

DLL Implementation: `SCPI_Command(":PSWEEP:TRIGGERIN:1")`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:TRIGGERIN:1`

3.8.11. POWER SWEEP – SET TRIGGER OUT MODE

:PSWEEP:TRIGGEROUT:[mode]

This function specifies how the Trigger Out port will be used during the power sweep. The modes are:
0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) as each power is set
2 – Provide a trigger output (Trigger Out = logic 1) as each power sweep is initiated

Parameters

Value	Description
0	Trigger output disabled (default)
1	Set Trigger Out on setting each power
2	Set Trigger Out on starting each sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:TRIGGEROUT:1	1

DLL Implementation: `SCPI_Command(":PSWEEP:TRIGGEROUT:1")`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:TRIGGEROUT:1`

3.8.12. POWER SWEEP – GET TRIGGER IN MODE

:PSWEEP:TRIGGERIN?

This function returns a code to indicate how the power sweep will respond to an external trigger. The modes are:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each point
2 – Wait for external trigger (Trigger In = logic 1) before starting each sweep

Return Value

Value	Description
0	Ignore trigger input
1	Wait for trigger before setting each power
2	Wait for trigger before starting each sweep

Examples

String to Send	String Returned
:PSWEEP:TRIGGERIN?	0

DLL Implementation: `SCPI_Query(":PSWEEP:TRIGGERIN?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:TRIGGERIN?`

3.8.13. POWER SWEEP – GET TRIGGER OUT MODE

:PSWEEP:TRIGGEROUT?

This function indicates how the Trigger Out port will be used during the power sweep. The modes are:
0 – Trigger output disabled
1 – Provide a trigger output (Trigger Out = logic 1) as each power is set
2 – Provide a trigger output (Trigger Out = logic 1) as each power sweep is initiated

Return Value

Value	Description
0	Trigger output disabled
1	Set Trigger Out on setting each power
2	Set Trigger Out on starting each sweep

Examples

String to Send	String Returned
:PSWEEP:TRIGGEROUT?	0

DLL Implementation: `SCPI_Query(":PSWEEP:TRIGGEROUT?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:TRIGGEROUT?`

3.8.14. POWER SWEEP – START/STOP SWEEP

:PSWEEP:MODE:[mode]

This function starts or stops the power sweep using the previously defined parameters.

Parameters

Value	Description
ON	Start power sweep
OFF	Stop power sweep

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PSWEEP:MODE:ON	1

DLL Implementation: `SCPI_Command(":PSWEEP:MODE:ON")`
HTTP Implementation: `http://10.10.10.10/:PSWEEP:MODE:ON`

3.9. SCPI – Frequency & Power Hop List

The signal generator can be configured to produce a fast, automatic frequency and power hop sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python

```
# Declare a sequence of 50 points, set dwell time of 10ms
MyPTE1.SCPI_Command(":HOP:POINTS:50")
MyPTE1.SCPI_Command(":HOP:DWELL:10")
# Index point 1 and set to 1000MHz, -10dBm
MyPTE1.SCPI_Command(":HOP:POINT:0")
MyPTE1.SCPI_Command(":HOP:FREQ:1000")
MyPTE1.SCPI_Command(":HOP:POWER:-10")
# Index point 2 and set to 1100MHz, -8dBm
MyPTE1.SCPI_Command(":HOP:POINT:1")
MyPTE1.SCPI_Command(":HOP:FREQ:1100")
MyPTE1.SCPI_Command(":HOP:POWER:-8")
# Index and set points 2 TO 49 in the same way
# Start the hop sequence
MyPTE1.SCPI_Command(":PWR:RF:ON")
MyPTE1.SCPI_Command(":HOP:MODE:ON")
```

Visual Basic

```
' Declare a sequence of 50 points, set dwell time of 10ms
MyPTE1.SCPI_Command(":HOP:POINTS:50")
MyPTE1.SCPI_Command(":HOP:DWELL:10")
' Index point 1 and set to 1000MHz, -10dBm
MyPTE1.SCPI_Command(":HOP:POINT:0")
MyPTE1.SCPI_Command(":HOP:FREQ:1000")
MyPTE1.SCPI_Command(":HOP:POWER:-10")
' Index point 2 and set to 1100MHz, -8dBm
MyPTE1.SCPI_Command(":HOP:POINT:1")
MyPTE1.SCPI_Command(":HOP:FREQ:1100")
MyPTE1.SCPI_Command(":HOP:POWER:-8")
' Index and set points 2 TO 49 in the same way
' Start the hop sequence
MyPTE1.SCPI_Command(":PWR:RF:ON")
MyPTE1.SCPI_Command(":HOP:MODE:ON")
```

Visual C++	`// Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1->SCPI_Command(":HOP:POINTS:50");`
	`MyPTE1->SCPI_Command(":HOP:DWELL:10");`
	`// Index point 1 and set to 1000MHz, -10dBm`
	`MyPTE1->SCPI_Command(":HOP:POINT:0");`
	`MyPTE1->SCPI_Command(":HOP:FREQ:1000");`
	`MyPTE1->SCPI_Command(":HOP:POWER:-10");`
	`// Index point 2 and set to 1100MHz, -8dBm`
	`MyPTE1->SCPI_Command(":HOP:POINT:1");`
	`MyPTE1->SCPI_Command(":HOP:FREQ:1100");`
	`MyPTE1->SCPI_Command(":HOP:POWER:-8");`
	`// Index and set points 2 to 49 in the same way`
	`// Start the hop sequence`
	`MyPTE1->SCPI_Command(":PWR:RF:ON");`
	`MyPTE1->SCPI_Command(":HOP:MODE:ON");`

Visual C#	`// Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.SCPI_Command(":HOP:POINTS:50");`
	`MyPTE1.SCPI_Command(":HOP:DWELL:10");`
	`// Index point 1 and set to 1000MHz, -10dBm`
	`MyPTE1.SCPI_Command(":HOP:POINT:0");`
	`MyPTE1.SCPI_Command(":HOP:FREQ:1000");`
	`MyPTE1.SCPI_Command(":HOP:POWER:-10");`
	`// Index point 2 and set to 1100MHz, -8dBm`
	`MyPTE1.SCPI_Command(":HOP:POINT:1");`
	`MyPTE1.SCPI_Command(":HOP:FREQ:1100");`
	`MyPTE1.SCPI_Command(":HOP:POWER:-8");`
	`// Index and set points 2 to 49 in the same way`
	`// Start the hop sequence`
	`MyPTE1.SCPI_Command(":PWR:RF:ON");`
	`MyPTE1.SCPI_Command(":HOP:MODE:ON");`

MatLab	`% Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.SCPI_Command(":HOP:POINTS:50")`
	`MyPTE1.SCPI_Command(":HOP:DWELL:10")`
	`% Index point 1 and set to 1000MHz, -10dBm`
	`MyPTE1.SCPI_Command(":HOP:POINT:0")`
	`MyPTE1.SCPI_Command(":HOP:FREQ:1000")`
	`MyPTE1.SCPI_Command(":HOP:POWER:-10")`
	`% Index point 2 and set to 1100MHz, -8dBm`
	`MyPTE1.SCPI_Command(":HOP:POINT:1")`
	`MyPTE1.SCPI_Command(":HOP:FREQ:1100")`
	`MyPTE1.SCPI_Command(":HOP:POWER:-8")`
	`% Index and set points 2 to 49 in the same way`
	`% Start the hop sequence`
	`MyPTE1.SCPI_Command(":PWR:RF:ON")`
	`MyPTE1.SCPI_Command(":HOP:MODE:ON")`

Example fast sequence using HTTP for Ethernet control:

- http://10.10.10.10/:HOP:POINTS:50
- http://10.10.10.10/:HOP:DWELL:10
- http://10.10.10.10/:HOP:POINT:0
- http://10.10.10.10/:HOP:FREQ:1000
- http://10.10.10.10/:HOP:POWER:-10
- http://10.10.10.10/:HOP:POINT:1
- http://10.10.10.10/:HOP:FREQ:1100
- http://10.10.10.10/:HOP:POWER:-8
- … (Declare points 2 to 49) …
- http://10.10.10.10/:PWR:RF:ON
- http://10.10.10.10/:HOP:MODE:ON

3.9.1. HOP – SET NUMBER OF POINTS

:HOP:POINTS:[number]

This function sets the number of points to be used in a frequency/power hop.

Parameters

Variable	Description
[number]	The number of points in the hop (maximum is 500)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:POINTS:50	1

DLL Implementation: SCPI_Command(":HOP:POINTS:50")
HTTP Implementation: http://10.10.10.10/:HOP:POINTS:50

3.9.2. HOP – GET NUMBER OF POINTS

This function returns the number of points to be used in the frequency/power hop.

:HOP:POINTS?

Return Value

Variable	Description
[number]	The number of points in the hop

Examples

String to Send	String Returned
:HOP:POINTS?	250

DLL Implementation: SCPI_Query(":HOP:POINTS?", RetStr)
HTTP Implementation: http://10.10.10.10/:HOP:POINTS?

3.9.3. HOP – GET MAXIMUM NUMBER OF POINTS

:HOP:MAXPOINTS?

This function returns the maximum number of points that can be used in a frequency/power hop.

Return Value

Variable	Description
[number]	The maximum number of points allowed in a hop sequence

Examples

String to Send	String Returned
:HOP:MAXPOINTS?	100

DLL Implementation: SCPI_Query(":HOP:MAXPOINTS?", RetStr)
HTTP Implementation: http://10.10.10.10/:HOP:MAXPOINTS?

3.9.4. HOP – SET INDEX POINT

:HOP:POINT:[index]

This function specifies which point in the hop sequence is to be indexed so that it's frequency/power can be set.

Parameters

Variable	Description
[index]	The point to be indexed.

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:POINT:1	1

DLL Implementation: `SCPI_Command(":HOP:POINT:1")`
HTTP Implementation: `http://10.10.10.10/:HOP:POINT:1`

3.9.5. HOP – GET INDEX POINT

:HOP:POINT?

This function returns the index number of the "active" point in the hop sequence, this is the point which is currently available for setting frequency and power.

Return Value

Variable	Description
[index]	The index number of the active point in the hop sequence

Examples

String to Send	String Returned
:HOP:POINT?	5

DLL Implementation: `SCPI_Query(":HOP:POINT?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:POINT?`

3.9.6. HOP – SET FREQUENCY OF INDEXED POINT

:HOP:FREQ:[frequency]

This function sets the frequency in MHz of the "active" point in the hop sequence (the point that is currently indexed). Hop – Set Index Point should be called first to specify which point in the hop sequence is indexed.

Parameters

Variable	Description
[frequency]	The frequency (MHz) to set for the indexed hop point

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:FREQ:1000.5	1

DLL Implementation: `SCPI_Command(":HOP:FREQ:1000.5")`
HTTP Implementation: `http://10.10.10.10/:HOP:FREQ:1000.5`

3.9.7. HOP – GET FREQUENCY OF INDEXED POINT

:HOP:FREQ?

This function returns the frequency in MHz of the "active" point in the hop sequence (the point that is currently indexed). Hop – Set Index Point should be called first to specify which point in the hop sequence is indexed.

Return Value

Variable	Description
[frequency]	The frequency (MHz) for the "active" (indexed) point in the hop sequence

Examples

String to Send	String Returned
:HOP:FREQ?	1500

DLL Implementation: `SCPI_Query(":HOP:FREQ?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:FREQ?`

3.9.8. HOP – SET POWER OF INDEXED POINT

:HOP:POWER:[power]

This function sets the power in dBm of the "active" point in the hop sequence (the point that is currently indexed). Hop – Set Index Point should be called first to specify which point in the hop sequence is indexed.

Parameters

Variable	Description
[power]	The power (dBm) to set for the indexed hop point

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:POWER:10	1

DLL Implementation: `SCPI_Command(":HOP:POWER:10")`
HTTP Implementation: `http://10.10.10.10/:HOP:POWER:10`

3.9.9. HOP – GET POWER OF INDEXED POINT

:HOP:POWER?

This function returns the power in dBm of the "active" point in the hop sequence (the point that is currently indexed). Hop – Set Index Point should be called first to specify which point in the hop sequence is indexed.

Return Value

Variable	Description
[power]	The power (dBm) for the "active" (indexed) point in the hop sequence

Examples

String to Send	String Returned
:HOP:POWER?	10

DLL Implementation: `SCPI_Query(":HOP:POWER?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:POWER?`

3.9.10. HOP – SET DWELL TIME

:HOP:DWELL:[time]

This function sets the dwell time to be used in a frequency/power hop sequence (the time in milliseconds for the generator to pause at each point).

Parameters

Variable	Description
[time]	The dwell time in ms

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:DWELL:100	1

DLL Implementation: `SCPI_Command(":HOP:DWELL:100")`
HTTP Implementation: `http://10.10.10.10/:HOP:DWELL:100`

3.9.11. HOP – GET DWELL TIME

:HOP:DWELL?

This function returns the dwell time to be used in a frequency/power hop sequence (the time in milliseconds for the generator to pause at each power point).

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:HOP:DWELL?	20

DLL Implementation: `SCPI_Query(":HOP:DWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:DWELL?`

3.9.12. HOP – GET DWELL TIME SPECIFICATION

:HOP:[spec]?

This function returns the minimum or maximum dwell time specifications for the generator.

Parameters

Value	Description
MAXDWELL	Return the maximum specified dwell time
MINDWELL	Return the minimum specified dwell time

Return Value

Variable	Description
[time]	The dwell time in ms

Examples

String to Send	String Returned
:HOP:MINDWELL?	10
:HOP:MAXDWELL?	1000

DLL Implementation: `SCPI_Query(":HOP:MINDWELL?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:MINDWELL?`

3.9.13. HOP – SET DIRECTION

:HOP:DIRECTION:[mode]

This function sets the direction of a frequency/power hop sequence.

Parameters

Value	Description
0	Forward (hop in sequence from the first point in list to the last). This is the default.
1	Reverse (hop in sequence from the last point in the list to the first)
2	Bi-directional (hop in sequence from first to last, then last to first)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:HOP:DIRECTION:1`	`1`

DLL Implementation: `SCPI_Command(":HOP:DIRECTION:1")`
HTTP Implementation: `http://10.10.10.10/:HOP:DIRECTION:1`

3.9.14. HOP – GET DIRECTION

:HOP:DIRECTION?

This function returns the direction of a frequency/power hop sequence.

Return Value

Value	Description
0	Forward (hop in sequence from the first point in list to the last)
1	Reverse (hop in sequence from the last point in list to the first)
2	Bi-directional (hop in sequence first to last, then last to first)

Examples

String to Send	String Returned
`:HOP:DIRECTION?`	`0`

DLL Implementation: `SCPI_Query(":HOP:DIRECTION?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:DIRECTION?`

3.9.15. HOP – SET TRIGGER IN MODE

:HOP:TRIGGERIN:[mode]

This function specifies how the hop sequence should respond to an external trigger.

Parameters

Value	Description
0	Ignore trigger input (default)
1	Wait for trigger signal (logic 1) before setting each point
2	Wait for trigger signal (logic 1) before starting each hop sequence

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:TRIGGERIN:1	1

DLL Implementation: `SCPI_Command(":HOP:TRIGGERIN:1")`
HTTP Implementation: `http://10.10.10.10/:HOP:TRIGGERIN:1`

3.9.16. HOP – SET TRIGGER OUT MODE

:HOP:TRIGGEROUT:[mode]

This function specifies how the Trigger Out port will be used during the hop sequence.

Parameters

Value	Description
0	Trigger output disabled (default)
1	Set Trigger Out (logic 1) on setting each point
2	Set Trigger Out (logic 1) on starting each hop sequence

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:TRIGGEROUT:1	1

DLL Implementation: `SCPI_Command(":HOP:TRIGGEROUT:1")`
HTTP Implementation: `http://10.10.10.10/:HOP:TRIGGEROUT:1`

3.9.17. HOP – GET TRIGGER IN MODE

:HOP:TRIGGERIN?

This function returns a code to indicate how the hop sequence will respond to an external trigger.

Return Value

Value	Description
0	Ignore trigger input
1	Wait for trigger input (logic 1) before setting each point
2	Wait for trigger input (logic 1) before starting each hop sequence

Examples

String to Send	String Returned
`:HOP:TRIGGERIN?`	`0`

DLL Implementation: `SCPI_Query(":HOP:TRIGGERIN?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:TRIGGERIN?`

3.9.18. HOP – GET TRIGGER OUT MODE

:HOP:TRIGGEROUT?

This function indicates how the Trigger Out port will be used during the hop sequence.

Return Value

Value	Description
0	Trigger output disabled
1	Set Trigger Out on setting each hop
2	Set Trigger Out on starting each hop sequence

Examples

String to Send	String Returned
`:HOP:TRIGGEROUT?`	`0`

DLL Implementation: `SCPI_Query(":HOP:TRIGGEROUT?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:HOP:TRIGGEROUT?`

3.9.19. HOP – START/STOP HOP SEQUENCE

:HOP:MODE:[mode]

This function starts or stops the hop sequence using the previously defined parameters.

Parameters

Value	Description
ON	Start hop sequence
OFF	Stop hop sequence

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:HOP:MODE:ON	1

DLL Implementation: `SCPI_Command(":HOP:MODE:ON")`
HTTP Implementation: `http://10.10.10.10/:HOP:MODE:ON`

3.10. SCPI - Regular Pulse Modulation

The signal generator can be configured to produce repetitive RF pulse sequences with fixed frequency and power, supporting internal or external modulation and input / output trigger options. The user stores the parameters of the pulse sequence in the generator's memory and can then enable / disable the output as required.

An example programming sequence to configure a pulse sequence using SCPI on a signal generator connected via the USB interface would be as follows:

Python

```
# Set generator to 6GHz and 0.5dBm output
MyPTE1.SCPI_Command(":FREQ:6GHz")
MyPTE1.SCPI_Command(":PWR:0.5")
# Configure a pulse of 50us duration, 500us off time
MyPTE1.SCPI_Command(":PULSE:TIMEUNITS=USEC")
MyPTE1.SCPI_Command(":PULSE:TIMEON:50")
MyPTE1.SCPI_Command(":PULSE:TIMEOFF:500")
# Enable the output (continuous pulses)
MyPTE1.SCPI_Command(":PWR:RF:ON")
MyPTE1.SCPI_Command(":PULSE:MODE:FREERUN:ON")
```

Visual Basic

```
' Set generator to 6GHz and 0.5dBm output
MyPTE1.SCPI_Command(":FREQ:6GHz")
MyPTE1.SCPI_Command(":PWR:0.5")
' Configure a pulse of 50us duration, 500us off time
MyPTE1.SCPI_Command(":PULSE:TIMEUNITS=USEC")
MyPTE1.SCPI_Command(":PULSE:TIMEON:50")
MyPTE1.SCPI_Command(":PULSE:TIMEOFF:500")
' Enable the output (continuous pulses)
MyPTE1.SCPI_Command(":PWR:RF:ON")
MyPTE1.SCPI_Command(":PULSE:MODE:FREERUN:ON")
```

Visual C++

```
// Set generator to 6GHz and 0.5dBm output
MyPTE1->SCPI_Command(":FREQ:6GHz");
MyPTE1->SCPI_Command(":PWR:0.5");
// Set fixed 10dBm output power level and 10ms dwell time for the sweep
MyPTE1->SCPI_Command(":PULSE:TIMEUNITS=USEC");
MyPTE1->SCPI_Command(":PULSE:TIMEON:50");
MyPTE1->SCPI_Command(":PULSE:TIMEOFF:500");
// Enable the output (continuous pulses)
MyPTE1->SCPI_Command(":PWR:RF:ON");
MyPTE1->SCPI_Command(":PULSE:MODE:FREERUN:ON");
```

Visual C#	`// Set generator to 6GHz and 0.5dBm output`
	`MyPTE1.SCPI_Command(":FREQ:6GHz");`
	`MyPTE1.SCPI_Command(":PWR:0.5");`
	`// Configure a pulse of 50us duration, 500us off time`
	`MyPTE1.SCPI_Command(":PULSE:TIMEUNITS=USEC");`
	`MyPTE1.SCPI_Command(":PULSE:TIMEON:50");`
	`MyPTE1.SCPI_Command(":PULSE:TIMEOFF:500");`
	`// Enable the output (continuous pulses)`
	`MyPTE1.SCPI_Command(":PWR:RF:ON");`
	`MyPTE1.SCPI_Command(":PULSE:MODE:FREERUN:ON");`

MatLab	`% Set generator to 6GHz and 0.5dBm output`
	`MyPTE1.SCPI_Command(":FREQ:6GHz")`
	`MyPTE1.SCPI_Command(":PWR:0.5")`
	`% Configure a pulse of 50us duration, 500us off time`
	`MyPTE1.SCPI_Command(":PULSE:TIMEUNITS=USEC")`
	`MyPTE1.SCPI_Command(":PULSE:TIMEON:50")`
	`MyPTE1.SCPI_Command(":PULSE:TIMEOFF:500")`
	`% Enable the output (continuous pulses)`
	`MyPTE1.SCPI_Command(":PWR:RF:ON")`
	`MyPTE1.SCPI_Command(":PULSE:MODE:FREERUN:ON")`

The same sequence could be sent as a series of HTTP commands as below if the signal generator is connected through the Ethernet interface:

- `http://10.10.10.10/:FREQ:6GHz`
- `http://10.10.10.10/:PWR:0.5`
- `http://10.10.10.10/:PULSE:TIMEUNITS=USEC`
- `http://10.10.10.10/:PULSE:TIMEON:50`
- `http://10.10.10.10/:PULSE:TIMEOFF:500`
- `http://10.10.10.10/:PWR:RF:ON`
- `http://10.10.10.10/:PULSE:MODE:FREERUN:ON`

3.10.1. PULSE MODULATION – SET PULSE PERIOD

:PULSE:[segment]:[time]

Sets the on and off times of the pulse. The time will be interpreted in milliseconds or microseconds, as specified by Pulse Modulation – Set Time Units (the default is milliseconds).

Parameters

Variable	Value	Description
[segment]	TIMEON	Set the pulse on period
	TIMEOFF	Set the pulse off period
[time]		Pulse period (units according to Pulse Modulation – Set Time Units, default is ms)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PULSE:TIMEON:10	1
:PULSE:TIMEOFF:100	1

DLL Implementation: SCPI_Command(":PULSE:TIMEON:10")
HTTP Implementation: http://10.10.10.10/:PULSE:TIMEON:10

3.10.2. PULSE MODULATION – SET TIME UNITS

:PULSE:TIMEUNITS=[units]

Sets the units to be used by the generator to interpret the pulse on and off times (the default is milliseconds).

Parameters

Value	Description
MSEC	Set the time units as milliseconds
USEC	Set the time units as microseconds

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PULSE:TIMEUNITS=MSEC	1
:PULSE:TIMEUNITS=USEC	1

DLL Implementation: SCPI_Command(":PULSE:TIMEUNITS=USEC")
HTTP Implementation: http://10.10.10.10/:PULSE:TIMEUNITS=USEC

3.10.3. PULSE MODULATION – ENABLE OUTPUT & TRIGGER MODE

:PULSE:MODE:[mode]:ON

This function sets the pulse modulation in to "free-running", "source triggered", or "externally modulated" mode and enables the RF output.

- Free-running mode will start a continuous series of pulses, at the previously specified CW frequency and power level, according to the previously specified pulse on and off times. The external trigger input is ignored in this mode.
- Source triggered mode will cause the generator to emit a single pulse, at the previously specified CW frequency and power level, every time an external trigger (logic high) is detected at the Trigger In port
- Externally modulated mode enables a pulsed output at the previously specified CW frequency and power level. The RF output will be enabled for as long as the Trigger In port is held at logic high and disabled when the Trigger In port is held at logic low.

Notes:

1. When pulsed output is enabled via a USB connection, any subsequent command sent to the generator will turn off the RF output and disable pulsed mode.
2. When pulsed output is enabled over an Ethernet, the generator cannot receive any further commands via HTTP or Telnet; a UDP "Magic Packet" must be sent in order to turn off the RF output and disable pulsed mode.

Parameters

Value	Description
FREERUN	Enable the generator in free-running mode (continuous pulsed output with no external trigger)
STRIGGER	Enable the generator in source triggered mode (single output pulse when an external trigger is received)
SEXTERNAL	Enable the generator in externally modulated mode (continuous RF output while Trigger In is held at logic high; RF output disabled while Trigger In is at logic low)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PULSE:MODE:FREERUN:ON	1
:PULSE:MODE:STRIGGER:ON	1
:PULSE:MODE:SEXTERNAL:ON	1

DLL Implementation: `SCPI_Command(":PULSE:MODE:FREERUN:ON")`
HTTP Implementation: `http://10.10.10.10/:PULSE:MODE:FREERUN:ON`

3.10.4. PULSE MODULATION – TURN OFF OUTPUT (USB MODE)

:PULSE:MODE:OFF

Disables the RF output in pulsed mode.

Note: This function can only be used in conjunction with the DLL over a USB connection. To turn off a pulsed output when communicating over Ethernet, a "Magic Packet" must be sent using UDP (User Datagram Protocol).

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:PULSE:MODE:OFF	1

DLL Implementation: SCPI_Command(":PULSE:MODE:OFF")

3.10.5. PULSE MODULATION – TURN OFF OUTPUT (ETHERNET MODE)

Disables the RF output in pulsed mode when operating over an Ethernet connection. In this mode of operation, the generator will not register incoming commands using HTTP or Telnet so a "Magic Packet" must be sent using UDP (User Datagram Protocol) to the signal generator's IP address.

The Magic Packet is made up of 6 bytes of decimal 255 (hex FF) followed by 16 repetitions of the signal generator's MAC address (1 byte per hex octet). The signal generator listens for UDP data on port 4950.

UDP does not offer a guarantee of service so it may be necessary to check that the generator's RF output was disabled.

Magic Packet

An example Magic Packet is represented below in hexadecimal notation for a signal generator with MAC address "D0-73-7F-86-5C-2B":

{FF FF FF FF FF FF D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B D0 73 7F 86 5C 2B}

Examples (Visual Basic.NET)

Visual Basic.Net

```
Private Sub Send_Magic_Packet()
  On Error GoTo err_Send_Magic_Packet
  ' Send a UDP Magic Packet to turn off the generator
  ' when pulsed output enabled
  ' Note: the generator will not accept HTTP/Telnet commands in this mode
  ' The generator's MAC and IP addresses are required
  Dim GenIPAddress As String = "192.168.9.59"
  Dim GenMacAddress As String = "D0-73-7F-86-5C-2B"
  Dim MacByte() As String = Split(GenMacAddress, "-")
  Dim intCounter As Integer = 0
  Dim sendBytes(0 To 101) As Byte

  ' Create the first 6 bytes of the magic packet (all decimal 255/hex FF)
  For i = 1 To 6
    sendBytes(intCounter) = &HFF
    intCounter += 1
  Next

  ' Create rest of packet, 16 repetitions of the MAC address (byte per octet)
  For i = 1 To 16
    For J = 0 To 5
      sendBytes(intCounter)              =              Byte.Parse(MacByte(J),
Globalization.NumberStyles.HexNumber)
      intCounter += 1
    Next
  Next
```

```
  Dim BCIP As System.Net.IPAddress
  Dim EP As System.Net.IPEndPoint
  Dim UDP As New System.Net.Sockets.UdpClient

  ' Send to the generator's IP address
  BCIP = System.Net.IPAddress.Parse(GenIPAddress)
  ' Create the IP end point (the generator listens on port 4950)
  EP = New System.Net.IPEndPoint(BCIP, 4950)
  ' Send the magic packet
  UDP.Send(sendBytes, sendBytes.Length, EP)
  UDP.Close()

  MsgBox("Generator output should be disabled." & vbCrLf & vbCrLf & _
          "Note: UDP does not offer guarantee of delivery.",,"Packet Sent")

exit_Send_Magic_Packet:
  UDP = Nothing
  Exit Sub

err_Send_Magic_Packet:
  MsgBox(Err.Description)
  Resume exit_Send_Magic_Packet
End Sub
```

3.11. SCPI - Dynamic Pulse Modulation

All models except SSG-6000RC & SSG-6001RC can be configured to create flexible RF pulse sequences with varying frequency, power, pulse width and interval per pulse. The user stores the parameters of the hop sequence in the generator's memory and can then enable/disable the output as required.

An example programming sequence to configure a dynamic pulse sequence could be sent as a series of HTTP commands as below if the signal generator is connected through the Ethernet interface:

- `--- Create a sequence of 3 pulses, with 100 repetitions ---`
- `http://10.10.10.10/:DFS:NoOfPulses:3`
- `http://10.10.10.10/:DFS:Cont:0`
- `http://10.10.10.10/:DFS:NoOfCycles:100`
- `--- Set point 0 ---`
- `http://10.10.10.10/:DFS:Pulse_IDX:0`
- `http://10.10.10.10/:DFS:RF:FREQ:1000`
- `http://10.10.10.10/:DFS:RF:Power:5`
- `http://10.10.10.10/:DFS:PulseWidth:1`
- `http://10.10.10.10/:DFS:Interval:100`
- `--- Set point 1 ---`
- `http://10.10.10.10/:DFS:Pulse_IDX:1`
- `http://10.10.10.10/:DFS:RF:FREQ:1100`
- `http://10.10.10.10/:DFS:RF:Power:5`
- `http://10.10.10.10/:DFS:PulseWidth:2`
- `http://10.10.10.10/:DFS:Interval:150`
- `--- Set point 2 ---`
- `http://10.10.10.10/:DFS:Pulse_IDX:2`
- `http://10.10.10.10/:DFS:RF:FREQ:1200`
- `http://10.10.10.10/:DFS:RF:Power:5`
- `http://10.10.10.10/:DFS:PulseWidth:3`
- `http://10.10.10.10/:DFS:Interval:200`
- `--- Start the sequence ---`
- `http://10.10.10.10/:PWR:RF:ON`
- `http://10.10.10.10/:DFS:MODE:ON`

3.11.1. DYNAMIC PULSES – SET NUMBER OF POINTS

DFS:NoOfPulses:[number]

Sets the number of points to be used in a dynamic pulse sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[number]	The number of points in the sequence (from 1 to 100)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:NoOfPulses:50	1

DLL Implementation: `SCPI_Command(":DFS:NoOfPulses:50")`
HTTP Implementation: `http://10.10.10.10/:DFS:NoOfPulses:50`

3.11.2. DYNAMIC PULSES – GET NUMBER OF POINTS

DFS:NoOfPulses?

Returns the number of points to be used in a dynamic pulse sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[number]	The number of points in the hop

Examples

String to Send	String Returned
:DFS:NoOfPulses?	50

DLL Implementation: `SCPI_Query(":DFS:NoOfPulses?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:NoOfPulses?`

3.11.3. DYNAMIC PULSES – SET INDEX POINT

:DFS:Pulse_IDX:[index]

Specifies which point in the pulse sequence is to be indexed so that its parameters can be set.

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[index]	The point to be indexed (from 0 to 99)

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:Pulse_IDX:1	1

DLL Implementation: `SCPI_Command(":DFS:Pulse_IDX:1")`
HTTP Implementation: `http://10.10.10.10/:DFS:Pulse_IDX:1`

3.11.4. DYNAMIC PULSES – GET INDEX POINT

:DFS:Pulse_IDX?

Returns the index number of the "active" point in the pulse sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[index]	The index number of the active point in the sequence

Examples

String to Send	String Returned
:DFS:Pulse_IDX?	1

DLL Implementation: `SCPI_Query(":DFS:Pulse_IDX?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:Pulse_IDX?`

3.11.5. DYNAMIC PULSES – SET FREQUENCY OF INDEXED POINT

:DFS:RF:FREQ:[frequency]

Sets the frequency in MHz of the "active" point in the pulse sequence (the point that is currently indexed).

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[frequency]	The frequency (MHz) to set for the indexed point

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:RF:FREQ:1000	1

DLL Implementation: `SCPI_Command(":DFS:RF:FREQ:1000.5")`
HTTP Implementation: `http://10.10.10.10/:DFS:RF:FREQ:1000.5`

3.11.6. DYNAMIC PULSES – GET FREQUENCY OF INDEXED POINT

:DFS:RF:FREQ?

Returns the frequency in MHz of the "active" point in the pulse sequence (the point that is currently indexed).

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[frequency]	The frequency (MHz) for the "active" (indexed) point in the sequence

Examples

String to Send	String Returned
:DFS:FREQ?	1000

DLL Implementation: `SCPI_Query(":DFS:RF:FREQ?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:RF:FREQ?`

3.11.7. DYNAMIC PULSES – SET POWER OF INDEXED POINT

:DFS:RF:POWER:[power]

Sets the power in dBm of the "active" point in the sequence (the point that is currently indexed).

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[power]	The power (dBm) to set for the indexed point

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:RF:POWER:10	1

DLL Implementation: `SCPI_Command(":DFS:RF:POWER:10")`
HTTP Implementation: `http://10.10.10.10/:DFS:RF:POWER:10`

3.11.8. DYNAMIC PULSES – GET POWER OF INDEXED POINT

:DFS:RF:POWER?

Returns the power in dBm of the "active" point in the sequence (the point that is currently indexed).

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[power]	The power (dBm) for the "active" (indexed) point in the sequence

Examples

String to Send	String Returned
:DFS:RF:POWER?	10

DLL Implementation: `SCPI_Query(":DFS:RF:POWER?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:RF:POWER?`

3.11.9. DYNAMIC PULSES – SET PULSE WIDTH

:DFS:PulseWidth:[time]

Sets the pulse width (“on” time) in microseconds for the indexed point in the sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[time]	The pulse width in microseconds

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:PulseWidth:100	1

DLL Implementation: `SCPI_Command(":DFS:PulseWidth:100")`
HTTP Implementation: `http://10.10.10.10/:DFS:PulseWidth:100`

3.11.10. DYNAMIC PULSES – GET PULSE WIDTH

:DFS:PulseWidth?

Returns the pulse width (“on” time) in microseconds for the indexed point in the sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[time]	The pulse width in microseconds

Examples

String to Send	String Returned
:DFS:PulseWidth?	100

DLL Implementation: `SCPI_Query(":DFS:PulseWidth?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:PulseWidth?`

3.11.11. DYNAMIC PULSES – SET PULSE INTERVAL

:DFS:Interval:[time]

Sets the pulse interval (delay before the next pulse) in microseconds for the indexed point in the sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[time]	The pulse interval in microseconds

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:Interval:200	1

DLL Implementation: SCPI_Command(":DFS:Interval:200")
HTTP Implementation: http://10.10.10.10/:DFS:Interval:200

3.11.12. DYNAMIC PULSES – GET PULSE INTERVAL

:DFS:Interval?

Returns the pulse interval (delay before the next pulse) in microseconds for the indexed point in the sequence.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Value	Description
[time]	The pulse interval in microseconds

Examples

String to Send	String Returned
:DFS:Interval?	200

DLL Implementation: SCPI_Query(":DFS:Interval?", RetStr)
HTTP Implementation: http://10.10.10.10/:DFS:Interval?

3.11.13. DYNAMIC PULSES – SET NUMBER OF CYCLES

:DFS:NoOfCycles:[time]

Sets the number of cycles for which the complete dynamic pulse sequence should be repeated. Continuous mode must be disabled in order for this setting to take effect.

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Variable	Description
[cycles]	The number of cycles

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:DFS:NoOfCycles:100	1

DLL Implementation: `SCPI_Command(":DFS:NoOfCycles:100")`
HTTP Implementation: `http://10.10.10.10/:DFS:NoOfCycles:100`

3.11.14. DYNAMIC PULSES – GET NUMBER OF CYCLES

:DFS:NoOfCycles?

Returns the number of cycles for which the complete dynamic pulse sequence should be repeated. Continuous mode must be disabled in order for this setting to take effect.

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Variable	Description
[cycles]	The number of cycles

Examples

String to Send	String Returned
:DFS:NoOfCycles?	100

DLL Implementation: `SCPI_Query(":DFS:NoOfCycles?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:NoOfCycles?`

3.11.15. DYNAMIC PULSES – SET CONTINUOUS MODE

:DFS:CONT:[mode]

Set the pulse sequence to run continuously or (if disabled) for the number of cycles specified in "Set Number of Cycles".

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Value	Description
0	Disable continuous mode
1	Enable continuous mode

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:DFS:CONT:1`	`1`

DLL Implementation: `SCPI_Command(":DFS:CONT:1")`
HTTP Implementation: `http://10.10.10.10/:DFS:CONT:1`

3.11.16. DYNAMIC PULSES – CHECK CONTINUOUS MODE

:DFS:CONT?

Check whether the pulse sequence is set to run continuously or (if disabled) for the number of cycles specified in "Set Number of Cycles".

Applies To

All models except SSG-6000RC & SSG-6001RC

Return Value

Value	Description
0	Continuous mode is disabled
1	Continuous mode is enabled

Examples

String to Send	String Returned
`:DFS:CONT?`	`1`

DLL Implementation: `SCPI_Query(":DFS:CONT?", RetStr)`
HTTP Implementation: `http://10.10.10.10/:DFS:CONT?`

3.11.17. DYNAMIC PULSES – START/STOP HOP SEQUENCE

:DFS:MODE:[mode]

Starts or stops the hop sequence using the previously defined parameters. The RF output should also be enabled (Set RF Output On/Off).

Applies To

All models except SSG-6000RC & SSG-6001RC

Parameters

Value	Description
ON	Start sequence
OFF	Stop sequence

Return Value

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
`:DFS:MODE:ON`	`1`

DLL Implementation: `SCPI_Command(":DFS:MODE:ON")`
HTTP Implementation: `http://10.10.10.10/:DFS:MODE:ON`

3.12. SCPI - Ethernet Configuration

The following functions provide a method to review and edit the Ethernet TCP / IP connection parameters. The Update Ethernet Settings command must be used to save settings and restart the controller. Refer to Ethernet Control API for further details on Ethernet control of the device.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

3.12.1. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?`

3.12.2. GET MAC ADDRESS

:ETHERNET:CONFIG:MAC?

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.3. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.4. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.5. SET DHCP HOST NAME

:ETHERNET:CONFIG:DHCPHOSTNAME:[hostname]

Sets the hostname which can be used in place of the IP address when DHCP is enabled.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	Firmware D2 or later
SSG-30G(HP)-RC	Firmware D2 or later
SSG-44G(HP)-RC	All units

Parameters

Variable	Description
[hostname]	Custom string host name, unique for this device

Return String

Value	Description
0	Command failed
1	Command completed successfully

Examples

String to Send	String Returned
:ETHERNET:CONFIG:DHCPHOSTNAME:MCLXXXXX	1

HTTP Implementation: http://10.10.10.10/:ETHERNET:CONFIG:DHCPHOSTNAME:MCLXXXXX

3.12.6. GET DHCP HOST NAME

:ETHERNET:CONFIG:DHCPHOSTNAME?

Returns the hostname which can be used in place of the IP address when DHCP is enabled.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	Firmware D2 or later
SSG-30G(HP)-RC	Firmware D2 or later
SSG-44G(HP)-RC	All units

Return String

Variable	Description
[hostname]	Custom string host name, unique for this device

Examples

String to Send	String Returned
:ETHERNET:CONFIG:DHCPHOSTNAME?	MCLXXXXX

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

3.12.7. GET STATIC IP ADDRESS

:ETHERNET:CONFIG:IP?

Returns the user-entered static IP address.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.8. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.9. GET STATIC NETWORK GATEWAY

:ETHERNET:CONFIG:NG?

Returns the user-entered network gateway IP address.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.10. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.11. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?`

3.12.12. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.13. GET HTTP PORT

:ETHERNET:CONFIG:HTPORT?

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.14. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.15. GET TELNET PORT

:ETHERNET:CONFIG:TELNETPORT?

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.16. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.17. GET SSH PORT

:ETHERNET:CONFIG:SSHPORT?

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.18. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.19. GET SSH LOGIN NAME

:ETHERNET:CONFIG:SSHLOGINNAME?

Returns the login name to be used for SSH communication.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.20. SAVE SSH LOGIN NAME

:ETHERNET:CONFIG:SSHLOGINNAME:[name]

Sets the login name to be used for SSH communication.

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.21. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.22. SET PASSWORD REQUIREMENT

:ETHERNET:CONFIG:PWDENABLED:[enabled]

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.23. GET PASSWORD

:ETHERNET:CONFIG:PWD?

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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?

3.12.24. SET PASSWORD

:ETHERNET:CONFIG:PWD:[pwd]

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

Applies To

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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

3.12.25. 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

Model Name	Requirements
SSG-6000RC / SSG-6001RC	Not supported
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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/sg.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_gen_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 (OBSOLETE)

Provided for support of systems with an older version of the .Net framework installed (prior to 4.5) and a Mini-Circuits signal generator purchased prior to 2024. This DLL is no longer maintained and is not guaranteed to support all models and features. Use of the .Net Framework 4.5 DLL or control via the Ethernet connection is recommended.

Filename: mcl_gen64.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_Gen.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:
   a. 32-bit PC: `\WINDOWS\System32\Regsvr32 \WINDOWS\System32\mcl_Gen.dll`
   b. 64-bit PC: `\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\mcl_Gen.dll`
4. Hit enter to confirm and a message box will appear to advise of successful registration.

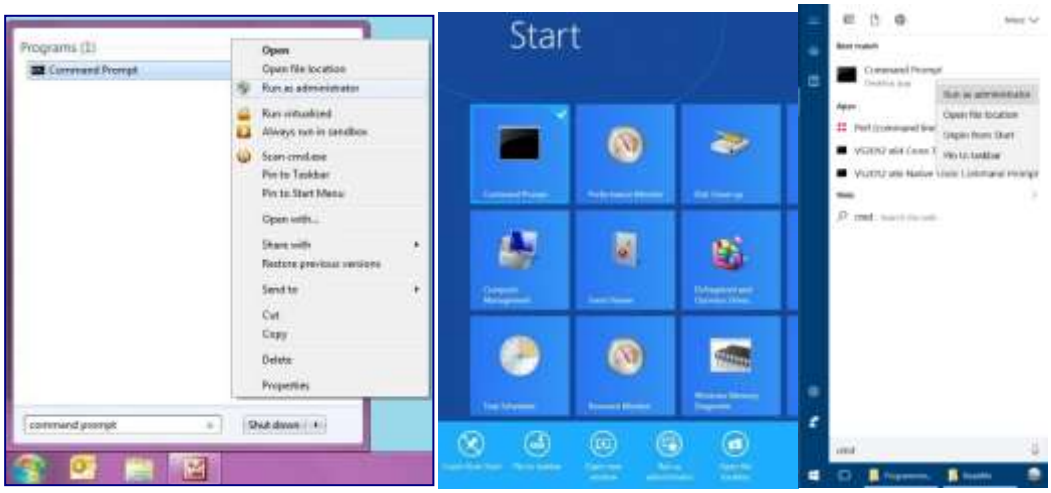


Fig 4.1-a: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

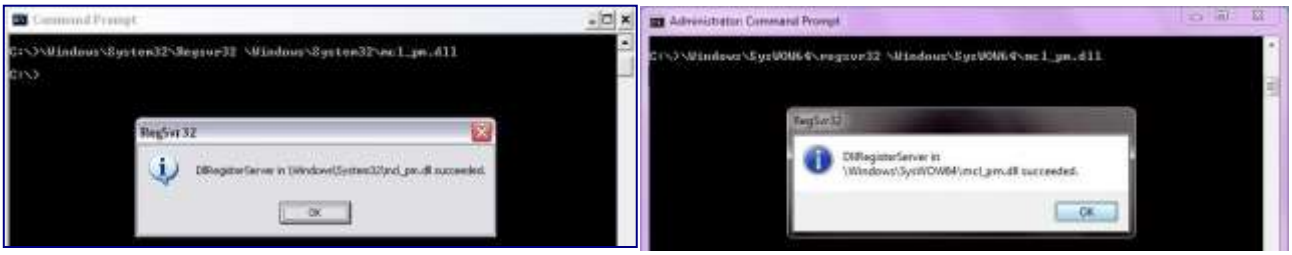


Fig 4.1-b: Registering the DLL in 32-bit (left) and 64-bit (right) environments

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

Examples Declarations Using the .NET 4.5 DLL (mcl_gen_NET45.dll)

*(For operation with the .Net 2.0 DLL, replace "*mcl_gen_NET45*" with "*mcl_gen64*")*

Python	`import clr  # Import the pythonnet CLR library`
	`clr.AddReference('mcl_gen_NET45')`
	`from mcl_gen_NET45 import usb_gen`
	`MyPTE1 = usb_gen()`
	`MyPTE2 = usb_gen()`

Visual Basic	`Public MyPTE1 As New mcl_gen_NET45.usb_gen` `Public MyPTE2 As New mcl_gen_NET45.usb_gen`
Visual C++	`mcl_gen_NET45::usb_gen ^MyPTE1 = gcnew mcl_gen_NET45::usb_gen();` `mcl_gen_NET45::usb_gen ^MyPTE2 = gcnew mcl_gen_NET45::usb_gen();`
Visual C#	`mcl_gen_NET45.usb_gen MyPTE1 = new mcl_gen_NET45.usb_gen();` `mcl_gen_NET45.usb_gen MyPTE2 = new mcl_gen_NET45.usb_gen();`

MatLab	`MCL_GEN = NET.addAssembly('C:\Windows\SysWOW64\mcl_gen_NET45.dll')`
	`MyPTE1 = mcl_gen_NET45.usb_gen`
	`MyPTE2 = mcl_gen_NET45.usb_gen`

Examples Declarations using the ActiveX DLL (MCL_Gen.dll)

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

4.3. Additional DLL Considerations

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

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

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

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

Definition	**int Read_SN(ByRef string SN)**

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

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

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

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

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

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

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

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

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

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

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

4.4. DLL – Connect & Identify

4.4.1. CONNECT

Short Connect(Optional String SN)

This function is called to initialize the USB connection to a signal generator. If multiple generators are connected via USB to the same computer then the serial number should be included, otherwise this can be omitted. The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the generator is no longer needed. The generator should be disconnected on completion of the program using the Disconnect function.

Parameters

Variable	Description
SN	Optional. A string containing the serial number of the signal generator. Can be omitted if only one generator is connected but must be included otherwise.

Return Values

Value	Description
0	No connection was possible
1	Connection successfully established
2	Device already connected

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);`

4.4.2. CONNECT BY ADDRESS

Short ConnectByAddress(Optional Short Address)

This function is called to initialize the USB connection to a signal generator by referring to a user defined address. The address is an integer number from 1 to 255 which can be assigned using the Set_Address function (the factory default is 255). The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the generator is no longer needed. The generator should be disconnected on completion of the program using the Disconnect function.

Parameters

Variable	Description
Address	Optional. A short containing the address of the signal generator. Can be omitted if only one signal generator is connected but must be included otherwise.

Return Values

Value	Description
0	No connection was possible
1	Connection successfully established
2	Device already connected

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);`

4.4.3. DISCONNECT

Void Disconnect()

This function is called to close the USB connection to the signal generator. It is strongly recommended that this function be used prior to ending the program. Failure to do so may result in a connection failure with the generator. Should this occur, terminate the program, unplug and shut down the signal generator before reconnecting and restarting.

Examples

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

4.4.4. READ MODEL NAME

Short Read_ModelName(String ModelName)

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

Parameters

Variable	Description
ModelName	Required. A string variable that will be updated with the Mini-Circuits model name for the signal generator.

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`

4.4.5. READ SERIAL NUMBER

Short Read_SN(String SN)

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

Parameters

Variable	Description
SN	Required. A string variable that will be updated with the Mini-Circuits serial number for the signal generator.

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`

4.4.6. GET FIRMWARE

Short GetExtFirmware(Short A0, Short A1, Short A2, String Firmware)

This function returns the internal firmware version of the signal generator along with three reserved variables for factory use.

Parameters

Variable	Description
A0	Numeric variable (factory use)
A1	Numeric variable (factory use)
A2	Numeric variable (factory use)
Firmware	String variable to be updated with the current firmware version, for example "B3"

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetExtFirmware(A0, A1, A2, Firmware)`
	`if status[0] > 0:`
	`    Firmware = str(status[4])`
	`    print("Firmware Version:", Firmware)`

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

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

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

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

4.4.7. GET FIRMWARE VERSION (ANTIQUATED)

Short GetFirmware()

This function is antiquated, GetExtFirmware should be used instead.

4.4.8. SET USB ADDRESS

Short Set_Address(Short Address)

This function allows the internal address of the connected signal generator to be changed from the factory default of 255. This allows the user to connect by a short address rather than serial number in future.

Parameters

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

Return Values

Value	Description
0	Command failed
>0	Command completed successfully

Examples

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);`

4.4.9. GET USB ADDRESS

Short Get_Address()

This function returns the address of the connected signal generator.

Return Values

Value	Description
0	Command failed
1-255	Address of the signal generator

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();`

4.4.10. GET LIST OF CONNECTED SERIAL NUMBERS

Short Get_Available_SN_List(String SN_List)

This function takes a user defined variable and updates it with a list of serial numbers for all signal generators currently connected via USB.

Parameters

Variable	Description
SN_List	Required. String variable which the function will update with a list of all available serial numbers, separated by a single space character, for example "11110001 11110002 11110003".

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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`

4.4.11. GET LIST OF AVAILABLE ADDRESSES

Short Get_Available_Address_List(String Add_List)

This function takes a user defined variable and updates it with a list of addresses of all signal generators currently connected via USB.

Parameters

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

Return Values

Value	Description
0	Command failed
>0	The number of signal generators connected

Examples

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`

4.4.12. GET TEMPERATURE

Float GetDeviceTemperature()

This function returns the internal temperature of the signal generator.

Return Values

Value	Description
Temperature	The device internal temperature in degrees Celsius

Examples

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

4.4.13. CHECK CONNECTION

Short Check_Connection()

This function checks whether the USB connection to the signal generator is active.

Return Values

Value	Description
0	No connection
1	USB connection to signal generator is active

Examples

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

4.4.14. GET STATUS (ANTIQUATED)

Short GetStatus()

This function is antiquated, please use Check Connection instead.

4.5. DLL - SCPI Communication

4.5.1. SEND SCPI QUERY

Short SCPI_Query(String QuerySTR, ByRef String RetSTR)

Sends a SCPI command or query to the generator and returns the response.

Parameters

Variable	Description
QuerySTR	The SCPI command or query to send to the device.
RetSTR	Variable passed by reference, to be updated with the generator's response to the command / query.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.SCPI_Query(":REF?", "")`
	`RetStr = Status[1]`
	`' Check which reference is in use.  For example:`
	`' RetSTR equals "EXT:10MHz" if external 10MHz ref in use`

Visual Basic	`Status = MyPTE1.SCPI_Query(":REF?", RetSTR)`
	`' Check which reference is in use.  For example:`
	`' RetSTR equals "EXT:10MHz" if external 10MHz ref in use`

Visual C++	`Status = MyPTE1->SCPI_Query(":REF?", RetSTR);`
	`// Check which reference is in use.  For example:`
	`// RetSTR equals "EXT:10MHz" if external 10MHz ref in use`

Visual C#	`Status = MyPTE1.SCPI_Query(":REF?", RetSTR);`
	`// Check which reference is in use.  For example:`
	`// RetSTR equals "EXT:10MHz" if external 10MHz ref in use`

MatLab	`[Status, RetSTR] = MyPTE1.SCPI_Query(":REF?", RetSTR)`
	`% Check which reference is in use.  For example:`
	`% RetSTR equals "EXT:10MHz" if external 10MHz ref in use`

4.5.2. SEND SCPI COMMAND

Short SCPI_Command(String CommandSTR)

Sends a SCPI command to the generator without looking for a response.

Parameters

Variable	Description
CommandSTR	Required. The SCPI command to send to the device.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.SCPI_Command(":PWR:RF:ON")` `# Enable RF output`
Visual Basic	`Status = MyPTE1.SCPI_Command(":PWR:RF:ON")` `' Enable RF output`
Visual C++	`Status = MyPTE1->SCPI_Command(":PWR:RF:ON");` `// Enable RF output`
Visual C#	`Status = MyPTE1.SCPI_Command(":PWR:RF:ON");` `// Enable RF output`
MatLab	`Status = MyPTE1.SCPI_Command(":PWR:RF:ON")` `% Enable RF output`

4.6. DLL - CW Output

These common functions apply to all models in the Mini-Circuits SSG signal generator series unless otherwise stated; providing the basic means to configure a CW RF output.

4.6.1. TURN ON RF OUTPUT

Short Set_Power_ON()

This function enables the RF output from the signal generator.

Return Values

Value	Description
0	Command failed
>0	Command completed successfully

Examples

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

4.6.2. TURN OFF RF OUTPUT

Short Set_Power_OFF()

This function disables the RF output from the signal generator.

Return Values

Value	Description
0	Command failed
>0	Command completed successfully

Examples

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

4.6.3. SET OUTPUT FREQUENCY AND POWER

Short SetFreqAndPower(Double Fr, Float Pr, Short TriggerOut)

This function sets the RF output frequency and power level of the signal generator and enables or disables the "trigger out" function.

Parameters

Variable	Description
Fr	Required. The frequency in MHz.
Pr	Required. The power in dBm.
TriggerOut	Required. An integer variable to determine whether the "trigger out" function should be enabled. 1 enables trigger out, 0 disables it.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetFreqAndPower(Freq, Power, 0)`
Visual Basic	`status = MyPTE1.SetFreqAndPower(Freq, Power, 0)`
Visual C++	`status = MyPTE1->SetFreqAndPower(Freq, Power, 0);`
Visual C#	`status = MyPTE1.SetFreqAndPower(Freq, Power, 0);`
MatLab	`status = MyPTE1.SetFreqAndPower(Freq, Power, 0);`

4.6.4. SET OUTPUT FREQUENCY

Short SetFreq(Double Fr, Short TriggerOut)

This function sets the RF output frequency of the signal generator and enables or disables the "trigger out" function. The output power of the signal generator will not be changed.

Parameters

Variable	Description
Fr	Required. The frequency in MHz.
TriggerOut	Required. An integer variable to determine whether the "trigger out" function should be enabled. 1 enables trigger out, 0 disables it.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.6.5. SET OUTPUT POWER

Short SetPower(Float Pr, Short TriggerOut)

This function sets the RF output power of the signal generator and enables or disables the "trigger out" function. The output frequency of the signal generator will not be changed.

Parameters

Variable	Description
Pr	Required. The power in dBm.
TriggerOut	Required. An integer variable to determine whether the "trigger out" function should be enabled. 1 enables trigger out, 0 disables it.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.6.6. GET GENERATOR OUTPUT STATUS

Short GetGenStatus(Byte Locked, Short PowerIsOn, Double Fr, Float Pr, Short UnLevelHigh, Short UnLevelLow)

This function returns the current status of the signal generator RF output in a series of user defined variables. The following parameters are checked:

- Generator lock status (locked/unlocked)
- RF output status (on/off)
- Current output frequency
- Current output power
- Current output power relative to user requested level

Parameters

Variable	Description
Locked	Byte variable which will be set to 1 if the frequency is locked or 0 otherwise.
PowerIsOn	Numeric variable which will be set to 1 if the RF output power is enabled or 0 otherwise.
Fr	Numeric variable which will be updated with the generator output frequency in MHz.
Pr	Numeric variable which will be updated with the generator output power in dBm.
UnLevelHigh	Numeric variable that will be set to 1 if the user requested a higher power level than the generator can achieve. The variable is set to 0 if the output power is at the correct level. See model datasheets for output power specifications.
UnLevelLow	Numeric variable that will be set to 1 if the user requested a lower power level than the generator can achieve. The variable is set to 0 if the output power is at the correct level. See model datasheets for output power specifications.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.GetGenStatus(Locked, PowerIsOn, Freq, Power, UnHigh, UnLow)`
Visual Basic	`status = MyPTE1.GetGenStatus(Locked, PowerIsOn, Freq, Power, UnHigh, UnLow)`
Visual C++	`status = MyPTE1->GetGenStatus(Locked, PowerIsOn, Freq, Power, UnHigh, UnLow);`
Visual C#	`status = MyPTE1.GetGenStatus(Locked, PowerIsOn, Freq, Power, UnHigh, UnLow);`
MatLab	`status = MyPTE1.GetGenStatus(Locked, PowerIsOn, Freq, Power, UnHigh, UnLow);`

4.6.7. CHECK EXTERNAL REFERENCE

Short ExtRefDetected()

This function checks whether an external 10MHz reference is connected to the generator. An external reference will be used automatically if it is present, otherwise the internal reference is used. The generator will assume that any signal on the Ref In port is a valid 10MHz source.

Return Values

Value	Description
0	External reference is not connected
1	External reference is connected

Examples

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

4.6.8. GET REFERENCE SOURCE

String GetGenRef()

This function reports the whether the generator is currently operating with an external or internal reference source.

Return Values

Value	Description
INT	Generator is using the internal reference
EXT	Generator is using an external reference source

Examples

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

4.6.9. GET TRIGGER IN STATUS

Short GetTriggerIn_Status()

This function indicates whether the generator's trigger input is at logic level low or high.

Return Values

Value	Description
0	Trigger input is at logic low
1	Trigger input is at logic high

Examples

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

4.6.10. SET TRIGGER OUT

Short SetTrigger()

This function sets the generator's trigger output to logic high.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.6.11. CLEAR TRIGGER

Short Clear_Trigger()

This function clears the generator's trigger output (resets to logic low).

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.6.12. GET STEP SIZE SPEC

Float GetGenStepFreq()

This function reports the generator's minimum step size in KHz.

Return Values

Value	Description
Frequency	Generator step size in KHz

Examples

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

4.6.13. GET MAXIMUM FREQUENCY SPEC

Float GetGenMaxFreq()

This function reports the maximum output frequency in MHz that the generator is capable of providing.

Return Values

Value	Description
Frequency	Maximum output frequency in MHz

Examples

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

4.6.14. GET MINIMUM FREQUENCY SPEC

Float GetGenMinFreq()

This function reports the minimum output frequency in MHz that the generator is capable of providing.

Return Values

Value	Description
Frequency	Minimum output frequency in MHz

Examples

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

4.6.15. GET MAXIMUM POWER SPEC

Float GetGenMaxPower()

This function reports the maximum output power specification in dBm for the active generator. The maximum output power achievable by the generator is guaranteed to be at least as high as this specified level across the full operating frequency and will be even higher in some frequency bands.

Return Values

Value	Description
Power	Maximum output power in dBm

Examples

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

4.6.16. GET MINIMUM POWER SPEC

Float GetGenMinPower()

This function reports the minimum output power specification in dBm for the active generator. The minimum output power achievable by the generator is guaranteed to be at least as low as this specified level across the full operating frequency and will be even lower in some frequency bands.

Return Values

Value	Description
Power	Minimum output power in dBm

Examples

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

4.7. DLL – Operation Timers

These functions provide access to calibration information for the signal generators.

4.7.1. GET CALIBRATION REMINDER DATE

String GetCalReminderDate()

Returns the date on which a calibration reminder should be displayed.

Return Values

Value	Description
RemDate	A string representing the calibration reminder date, in the format "yymmdd", eg: "140625" for 25th June 2014.

Examples

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

4.7.2. SET CALIBRATION REMINDER DATE

Short SetCalReminderDate(String RemDate)

Sets the date on which a calibration reminder should be displayed.

Parameters

Value	Description
RemDate	A string representing the calibration reminder date, in the format "yymmdd", eg: "140625" for 25th June 2014.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetCalReminderDate("230625")`
Visual Basic	`status = MyPTE1.SetCalReminderDate("230625")`
Visual C++	`status = MyPTE1->SetCalReminderDate("230625");`
Visual C#	`status = MyPTE1.SetCalReminderDate("230625");`
MatLab	`status = MyPTE1.SetCalReminderDate("230625");`

4.7.3. GET CALIBRATION REMINDER DATE STATUS

Short GetCalReminderDateIsRequired()

Indicates whether a calibration reminder should be displayed from a given date.

Return Values

Value	Description
0	Calibration date reminders disabled
1	Calibration date reminders enabled

Examples

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

4.7.4. SET CALIBRATION REMINDER DATE STATUS

Short SetCALReminderDateIsRequired(Short DateRequired)

Sets whether a calibration reminder date is required.

Parameters

Value	Description
0	Calibration date reminders disabled
1	Calibration date reminders enabled

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.7.5. GET CALIBRATION REMINDER OPERATING TIME

Short GetCALReminderOTVal()

Returns the operating time after which a calibration reminder should be displayed. The operating time is measured in hours from the last calibration.

Return Values

Value	Description
OpTime	The operating time in hours at which a calibration reminder is due

Examples

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

4.7.6. SET CALIBRATION REMINDER OPERATING TIME

Short SetCALReminderOTVal(Short OpTime)

Sets the operating time after which a calibration reminder should be displayed. The operating time is measured in hours from the last calibration.

Parameters

Value	Description
OpTime	The operating time in hours at which a calibration reminder is due

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetCALReminderOTVal(120)`
Visual Basic	`status = MyPTE1.SetCALReminderOTVal(120)`
Visual C++	`status = MyPTE1->SetCALReminderOTVal(120);`
Visual C#	`status = MyPTE1.SetCALReminderOTVal(120);`
MatLab	`status = MyPTE1.SetCALReminderOTVal(120);`

4.7.7. GET CALIBRATION REMINDER OPERATING TIME STATUS

Short GetCALReminderOTValIsRequired()

Indicates whether a calibration reminder should be displayed after a specified operating time (in hours from last calibration).

Return Values

Value	Description
0	Calibration reminders after predefined number of operating hours are disabled
1	Calibration reminders after predefined number of operating hours are enabled

Examples

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

4.7.8. SET CALIBRATION REMINDER OPERATING TIME STATUS

Short SetCALReminderOTValIsRequired(Short OTRequired)

Sets whether a calibration reminder should be displayed after a specified operating time.

Parameters

Value	Description
0	Calibration reminders after predefined number of operating hours are disabled
1	Calibration reminders after predefined number of operating hours are enabled

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.7.9. GET GENERATOR OPERATING TIME

Short GetGenOperationTime()

Returns the total time in hours that the signal generator has been powered on since the last calibration.

Return Values

Value	Description
OpTime	Operating time in hours since last calibration

Examples

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

4.8. DLL – Pulse Modulation

The signal generator can be configured to produce a pulsed output, either in response to an external trigger or continuously using the generator's internal timing systems. The user stores the parameters of the pulse sequence in the generator's memory and can then enable/disable the output as required.

An example programming sequence to configure a pulse sequence using a signal generator connected via the USB interface would be as follows:

Python	`MyPTE1.SetFreqAndPower(6000, 0.5, 0)`	`# 6GHz and 0.5 dBm CW output`
	`MyPTE1.Set_PulseMode(50, 500, 0)`	`# 50us duration, 500us off time`
	`MyPTE1.SetPowerON`	`# Enable the output (continuous pulses)`
Visual Basic	`MyPTE1.SetFreqAndPower(6000, 0.5, 0)`	`' 6GHz and 0.5 dBm CW output`
	`MyPTE1.Set_PulseMode(50, 500, 0)`	`' 50us duration, 500us off time`
	`MyPTE1.SetPowerON`	`' Enable the output (continuous pulses)`
Visual C++	`MyPTE1->SetFreqAndPower(6000, 0.5, 0);`	`// 6GHz and 0.5 dBm CW output`
	`MyPTE1->Set_PulseMode(50, 500, 0);`	`// 50us duration, 500us off time`
	`MyPTE1->SetPowerON;`	`// Enable the output (continuous pulses)`
Visual C#	`MyPTE1.SetFreqAndPower(6000, 0.5, 0);`	`// 6GHz and 0.5 dBm CW output`
	`MyPTE1.Set_PulseMode(50, 500, 0);`	`// 50us duration, 500us off time`
	`MyPTE1.SetPowerON;`	`// Enable the output (continuous pulses)`
MatLab	`MyPTE1.SetFreqAndPower(6000, 0.5, 0)`	`% 6GHz and 0.5 dBm CW output`
	`MyPTE1.Set_PulseMode(50, 500, 0)`	`% 50us duration, 500us off time`
	`MyPTE1.SetPowerON`	`% Enable the output (continuous pulses)`

4.8.1. SET PULSE MODE

Short Set_PulseMode(Short T_OFF, Short T_ON, Short Tunit)

This function creates a pulsed output with a user specified pulse duration and time period. The output during the pulse "on" period is a CW signal with a frequency and power level which should be set by the user in advance (see Set Output Frequency and Power). The pulse period will repeat indefinitely until any other function is called by the user's program.

Parameters

Variable	Description
T_OFF	Pulse interval / "off" time
T_ON	Pulse width / "on" time
Tunit	Time units for the T_OFF and T_ON time periods; 0 for microseconds or 1 for milliseconds.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`MyPTE1.Set_PulseMode(50, 500, 0)`	`# 50us duration, 500us off time`
Visual Basic	`MyPTE1.Set_PulseMode(50, 500, 0)`	`' 50us duration, 500us off time`
Visual C++	`MyPTE1->Set_PulseMode(50, 500, 0);`	`// 50us duration, 500us off time`
Visual C#	`MyPTE1.Set_PulseMode(50, 500, 0);`	`// 50us duration, 500us off time`
MatLab	`MyPTE1.Set_PulseMode(50, 500, 0)`	`% 50us duration, 500us off time`

4.8.2. SET TRIGGERED PULSE MODE

Short Set_PulseMode_Trigger(Short TriggerType, Short T_ON, Short Tunit)

This function creates a pulsed output with a user specified pulse duration that will start when an external trigger is received at the "Trigger In" input. The output during the pulse "on" period is a CW signal with a frequency and power level which should be set by the user in advance (see Set Output Frequency and Power). The pulsed output will be enabled until any other function is called by the user's program.

Parameters

Variable	Description
TriggerType	Required. The trigger input sequence that will trigger the pulsed output; 0 for Trigger In = on then off, or 1 for Trigger In = off then on.
T_ON	Required. The pulse "on" duration.
Tunit	Required. The units for the T_ON time period; 0 for microseconds or 1 for milliseconds.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1)`
	`# Set 2ms pulses of 1000MHz, 10dBm CW`
	`# Start the pulse when a "on, off" is received at Trigger In`

Visual Basic	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1)`
	`' Set 2ms pulses of 1000MHz, 10dBm CW`
	`' Start the pulse when a "on, off" is received at Trigger In`

Visual C++	`status = MyPTE1->SetFreqAndPower(1000, 10, 0);`
	`status = MyPTE1->Set_PulseMode_Trigger(0, 2, 1);`
	`// Set 2ms pulses of 1000MHz, 10dBm CW`
	`// Start the pulse when a "on, off" is received at Trigger In`

Visual C#	`status = MyPTE1.SetFreqAndPower(1000, 10, 0);`
	`status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1);`
	`// Set 2ms pulses of 1000MHz, 10dBm CW`
	`// Start the pulse when a "on, off" is received at Trigger In`

MatLab	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1)`
	`% Set 2ms pulses of 1000MHz, 10dBm CW`
	`% Start the pulse when a "on, off" is received at Trigger In`

4.8.3. SET EXTERNAL PULSE MODULATION

Short Set_ExtPulseMod()

This function enables a pulsed output in response to the generator's Trigger In port. The generator's CW output will be enabled with the specified frequency and power while the Trigger In port is held high. The output will be disabled while the Trigger In port is held low. The CW frequency and power for the "on" period should be set in advance (see Set Output Frequency and Power). The external pulse modulation mode will be disabled when any other command is sent to the generator.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_ExtPulseMod()`
	`# Enable externally modulated pulses of 1000MHz, 10dBm`

Visual Basic	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_ExtPulseMod()`
	`' Enable externally modulated pulses of 1000MHz, 10dBm`

Visual C++	`status = MyPTE1->SetFreqAndPower(1000, 10, 0);`
	`status = MyPTE1->Set_ExtPulseMod();`
	`// Enable externally modulated pulses of 1000MHz, 10dBm`

Visual C#	`status = MyPTE1.SetFreqAndPower(1000, 10, 0);`
	`status = MyPTE1.Set_ExtPulseMod();`
	`// Enable externally modulated pulses of 1000MHz, 10dBm`

MatLab	`status = MyPTE1.SetFreqAndPower(1000, 10, 0)`
	`status = MyPTE1.Set_ExtPulseMod()`
	`% Enable externally modulated pulses of 1000MHz, 10dBm`

4.9. DLL - Frequency Sweep

The signal generator can be configured to produce a fast, automatic frequency sweep sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python	`# Set sweep for 1000-6000MHz in 100MHz steps`
	`MyPTE1.FSweep_SetStartFreq(1000)`
	`MyPTE1.FSweep_SetStopFreq(6000)`
	`MyPTE1.FSweep_SetStepSize(100)`
	`# Set fixed 10dBm output power level and 10ms dwell time for the sweep`
	`MyPTE1.FSweep_SetDwell(10)`
	`MyPTE1.FSweep_SetPower(10)`
	`# Start the sweep`
	`MyPTE1.FSweep_SetMode(1)`

Visual Basic	`' Set sweep for 1000-6000MHz in 100MHz steps`
	`MyPTE1.FSweep_SetStartFreq(1000)`
	`MyPTE1.FSweep_SetStopFreq(6000)`
	`MyPTE1.FSweep_SetStepSize(100)`
	`' Set fixed 10dBm output power level and 10ms dwell time for the sweep`
	`MyPTE1.FSweep_SetDwell(10)`
	`MyPTE1.FSweep_SetPower(10)`
	`' Start the sweep`
	`MyPTE1.FSweep_SetMode(1)`

Visual C++	`// Set sweep for 1000-6000MHz in 100MHz steps`
	`MyPTE1->FSweep_SetStartFreq(1000);`
	`MyPTE1->FSweep_SetStopFreq(6000);`
	`MyPTE1->FSweep_SetStepSize(100);`
	`// Set fixed 10dBm output power level and 10ms dwell time for the sweep`
	`MyPTE1->FSweep_SetDwell(10);`
	`MyPTE1->FSweep_SetPower(10);`
	`// Start the sweep`
	`MyPTE1->FSweep_SetMode(1);`

Visual C#	// Set sweep for 1000-6000MHz in 100MHz steps
	MyPTE1.FSweep_SetStartFreq(1000);
	MyPTE1.FSweep_SetStopFreq(6000);
	MyPTE1.FSweep_SetStepSize(100);
	// Set fixed 10dBm output power level and 10ms dwell time for the sweep
	MyPTE1.FSweep_SetDwell(10);
	MyPTE1.FSweep_SetPower(10);
	// Start the sweep
	MyPTE1.FSweep_SetMode(1);

MatLab	% Set sweep for 1000-6000MHz in 100MHz steps MyPTE1.FSweep_SetStartFreq(1000) MyPTE1.FSweep_SetStopFreq(6000) MyPTE1.FSweep_SetStepSize(100) % Set fixed 10dBm output power level and 10ms dwell time for the sweep MyPTE1.FSweep_SetDwell(10) MyPTE1.FSweep_SetPower(10) % Start the sweep MyPTE1.FSweep_SetMode(1)

4.9.1. FREQUENCY SWEEP – GET DIRECTION

Short FSweep_GetDirection()

This function returns the current frequency sweep direction. The possible settings are:
0 – Increasing from start to stop frequency
1 – Decreasing from stop to start frequency
2 – Increasing from start to stop, before decreasing from stop to start frequency

Return Values

Value	Description
0	Increasing frequency sweep from start to stop frequency
1	Decreasing frequency sweep from stop to start frequency
2	Increasing then decreasing frequency sweep (from start to stop to start frequency)

Examples

Python	Sweep = MyPTE1.FSweep_GetDirection
Visual Basic	Sweep = MyPTE1.FSweep_GetDirection
Visual C++	Sweep = MyPTE1->FSweep_GetDirection();
Visual C#	Sweep = MyPTE1.FSweep_GetDirection();
MatLab	Sweep = MyPTE1.FSweep_GetDirection;

4.9.2. FREQUENCY SWEEP – GET DWELL TIME

Short FSweep_GetDwell()

This function returns the current dwell time setting in milliseconds; this is the length of time that the generator will pause at each frequency point.

Return Values

Value	Description
0	Command failed
Time	Dwell time in milliseconds

Examples

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

4.9.3. FREQUENCY SWEEP – GET MAXIMUM DWELL TIME

Short FSweep_GetMaxDwell()

This function returns the maximum allowed dwell time in milliseconds. Dwell time is the length of time that the generator will pause at each frequency point.

Return Values

Value	Description
0	Command failed
Time	Maximum dwell time in milliseconds

Examples

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

4.9.4. FREQUENCY SWEEP – GET MINIMUM DWELL TIME

Short FSweep_GetMinDwell()

This function returns the minimum allowed dwell time in milliseconds. Dwell time is the length of time that the generator will pause at each frequency point.

Return Values

Value	Description
0	Command failed
Time	Minimum dwell time in milliseconds

Examples

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

4.9.5. FREQUENCY SWEEP – GET POWER

Float FSweep_GetPower()

This function returns the current output power setting of the frequency sweep in dBm.

Return Values

Value	Description
-999	Command failed
Power	Output power in dBm

Examples

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

4.9.6. FREQUENCY SWEEP – GET START FREQUENCY

Double FSweep_GetStartFreq()

This function returns the start frequency in MHz of the current frequency sweep.

Return Values

Value	Description
0	Command failed
Freq	Start frequency in MHz

Examples

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

4.9.7. FREQUENCY SWEEP – GET STOP FREQUENCY

Double FSweep_GetStopFreq()

This function returns the stop frequency in MHz of the current frequency sweep.

Return Values

Value	Description
0	Command failed
Freq	Stop frequency in MHz

Examples

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

4.9.8. FREQUENCY SWEEP – GET STEP SIZE

Double FSweep_GetStepSize()

This function returns the step size in MHz of the current frequency sweep.

Return Values

Value	Description
0	Command failed
Freq	Step size in MHz

Examples

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

4.9.9. FREQUENCY SWEEP – GET TRIGGER IN MODE

Short FSweep_GetTriggerIn()

This function returns the Trigger Input mode for the frequency sweep, this dictates how the generator will respond to an external trigger:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each frequency point
2 – Wait for external trigger (Trigger In = logic 1) before starting each frequency

Return Values

Value	Description
0	Ignore Trigger In
1	Wait for Trigger In for each frequency point
2	Wait for Trigger In before starting each sweep

Examples

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

4.9.10. FREQUENCY SWEEP – GET TRIGGER OUT MODE

Short FSweep_GetTriggerOut()

This function returns Trigger Output mode for the frequency sweep, this dictates how the Trigger Out port will be used during the frequency sweep:
0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) as each frequency point is set
2 – Provide a trigger output (Trigger Out = logic 1) as each frequency sweep is initiated

Return Values

Value	Description
0	Trigger Out disabled
1	Trigger Out set at each frequency point
2	Trigger Out set as each sweep is initialized

Examples

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

4.9.11. FREQUENCY SWEEP – SET DIRECTION

Short FSweep_SetDirection(Short SweepDirection)

This function sets the direction of the frequency sweep.

Parameters

Value	Description
0	Increasing frequency sweep from start to stop frequency
1	Decreasing frequency sweep from stop to start frequency
2	Increasing then decreasing frequency sweep (from start to stop to start frequency)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.9.12. FREQUENCY SWEEP – SET DWELL TIME

Short FSweep_SetDwell(Short dwell_msec)

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each frequency point.

Parameters

Variable	Description
dwell_msec	Required. The dwell time in milliseconds.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Sweep = MyPTE1.FSweep_SetDwell(15)`
Visual Basic	`Sweep = MyPTE1.FSweep_SetDwell(15)`
Visual C++	`Sweep = MyPTE1->FSweep_SetDwell(15);`
Visual C#	`Sweep = MyPTE1.FSweep_SetDwell(15);`
MatLab	`Sweep = MyPTE1.FSweep_SetDwell(15);`

4.9.13. FREQUENCY SWEEP – START/STOP SWEEP

Short FSweep_SetMode(Short onoff)

This function starts or stops the frequency sweep using the previously defined parameters.
Note: The frequency sweep will stop automatically if any other command is sent.

Parameters

Variable	Description
1	Start frequency sweep
0	Stop frequency sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.9.14. FREQUENCY SWEEP – SET POWER

Short FSweep_SetPower(Float Pr)

This function sets the output power level in dBm to be used for the frequency sweep in.

Parameters

Variable	Description
Pr	Required. The fixed power level in dBm to be used for the frequency sweep.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.FSweep_SetPower(-10.5)`
Visual Basic	`Status = MyPTE1.FSweep_SetPower(-10.5)`
Visual C++	`Status = MyPTE1->FSweep_SetPower(-10.5);`
Visual C#	`Status = MyPTE1.FSweep_SetPower(-10.5);`
MatLab	`Status = MyPTE1.FSweep_SetPower(-10.5)`

4.9.15. FREQUENCY SWEEP – SET START FREQUENCY

Short FSweep_SetStartFreq(Double Fr)

This function sets the start frequency in MHz for the sweep.

Parameters

Variable	Description
Fr	Required. The start frequency in MHz.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.FSweep_SetStartFreq(250)`
Visual Basic	`Status = MyPTE1.FSweep_SetStartFreq(250)`
Visual C++	`Status = MyPTE1->FSweep_SetStartFreq(250);`
Visual C#	`Status = MyPTE1.FSweep_SetStartFreq(250);`
MatLab	`Status = MyPTE1.FSweep_SetStartFreq(250)`

4.9.16. FREQUENCY SWEEP – SET STOP FREQUENCY

Short FSweep_SetStopFreq(Double Fr)

This function sets the stop frequency in MHz for the sweep.

Parameters

Variable	Description
Fr	Required. The stop frequency in MHz.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.FSweep_SetStopFreq(5500)`
Visual Basic	`Status = MyPTE1.FSweep_SetStopFreq(5500)`
Visual C++	`Status = MyPTE1->FSweep_SetStopFreq(5500);`
Visual C#	`Status = MyPTE1.FSweep_SetStopFreq(5500);`
MatLab	`Status = MyPTE1.FSweep_SetStopFreq(5500)`

4.9.17. FREQUENCY SWEEP – SET STEP SIZE

Short FSweep_SetStepSize(Double Fr)

This function sets the step size in MHz to be used in the frequency sweep.

Parameters

Variable	Description
Fr	Required. The step size in MHz.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.FSweep_SetStepSize(0.1)`
Visual Basic	`Status = MyPTE1.FSweep_SetStepSize(0.1)`
Visual C++	`Status = MyPTE1->FSweep_SetStepSize(0.1);`
Visual C#	`Status = MyPTE1.FSweep_SetStepSize(0.1);`
MatLab	`Status = MyPTE1.FSweep_SetStepSize(0.1)`

4.9.18. FREQUENCY SWEEP – SET TRIGGER IN MODE

Short FSweep_SetTriggerIn(Short SweepTriggerIn)

This function specifies how the frequency sweep should respond to an external trigger.

Parameters

Variable	Description
0	Ignore Trigger In
1	Wait for Trigger In before each frequency point
2	Wait for Trigger In before commencing sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.9.19. FREQUENCY SWEEP – SET TRIGGER OUT MODE

Short FSweep_SetTriggerOut(Short SweepTriggerOut)

This function specifies how the Trigger Out port will be used during the frequency sweep.

Parameters

Variable	Description
0	Trigger Out disabled
1	Set Trigger Out (logic 1) at each frequency point
2	Set Trigger Out (logic 1) on commencing the sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.10. DLL - Power Sweep

The signal generator can be configured to produce a fast, automatic power sweep sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python

```
# Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1.PSweep_SetStartPower(-20)
MyPTE1.PSweep_SetStopPower(20)
MyPTE1.PSweep_SetStepSize(0.5)
# Set fixed 1000MHz output and 10ms dwell time for the sweep
MyPTE1.PSweep_SetDwell(10)
MyPTE1.PSweep_SetFreq(1000)
# Start the sweep
MyPTE1.PSweep_SetMode(1)
```

Visual Basic

```
' Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1.PSweep_SetStartPower(-20)
MyPTE1.PSweep_SetStopPower(20)
MyPTE1.PSweep_SetStepSize(0.5)
' Set fixed 1000MHz output and 10ms dwell time for the sweep
MyPTE1.PSweep_SetDwell(10)
MyPTE1.PSweep_SetFreq(1000)
' Start the sweep
MyPTE1.PSweep_SetMode(1)
```

Visual C++

```
// Set sweep for -20dBm to +20dBm in 0.5dB steps
MyPTE1->PSweep_SetStartPower(-20);
MyPTE1->PSweep_SetStopPower(20);
MyPTE1->PSweep_SetStepSize(0.5);
// Set fixed 1000MHz output and 10ms dwell time for the sweep
MyPTE1->PSweep_SetDwell(10);
MyPTE1->PSweep_SetFreq(1000);
// Start the sweep
MyPTE1->PSweep_SetMode(1);
```

Visual C#	// Set sweep for -20dBm to +20dBm in 0.5dB steps
	MyPTE1.PSweep_SetStartPower(-20);
	MyPTE1.PSweep_SetStopPower(20);
	MyPTE1.PSweep_SetStepSize(0.5);
	// Set fixed 1000MHz output and 10ms dwell time for the sweep
	MyPTE1.PSweep_SetDwell(10);
	MyPTE1.PSweep_SetFreq(1000);
	// Start the sweep
	MyPTE1.PSweep_SetMode(1);

MatLab	% Set sweep for -20dBm to +20dBm in 0.5dB steps MyPTE1.PSweep_SetStartPower(-20) MyPTE1.PSweep_SetStopPower(20) MyPTE1.PSweep_SetStepSize(0.5) % Set fixed 1000MHz output and 10ms dwell time for the sweep MyPTE1.PSweep_SetDwell(10) MyPTE1.PSweep_SetFreq(1000) % Start the sweep MyPTE1.PSweep_SetMode(1)

4.10.1. POWER SWEEP – GET DIRECTION

Short PSweep_GetDirection()

This function returns the current power sweep direction.

Return Values

Value	Description
0	Ascending power sweep from start to stop
1	Descending power sweep from stop to start
2	Ascending then descending power sweep

Examples

Python	Sweep = MyPTE1.PSweep_GetDirection
Visual Basic	Sweep = MyPTE1.PSweep_GetDirection
Visual C++	Sweep = MyPTE1->PSweep_GetDirection();
Visual C#	Sweep = MyPTE1.PSweep_GetDirection();
MatLab	Sweep = MyPTE1.PSweep_GetDirection;

4.10.2. POWER SWEEP – GET DWELL TIME

Short PSweep_GetDwell()

This function returns the current dwell time setting in milliseconds; this is the length of time that the generator will pause at each power setting.

Return Values

Value	Description
0	Command failed
Time	Dwell time in milliseconds

Examples

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

4.10.3. POWER SWEEP – GET MAXIMUM DWELL TIME

Short PSweep_GetMaxDwell()

This function returns the maximum allowed dwell time in milliseconds. Dwell time is the length of time that the generator will pause at each power setting.

Return Values

Value	Description
0	Command failed
Time	Maximum dwell time in milliseconds

Examples

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

4.10.4. POWER SWEEP – GET MINIMUM DWELL TIME

Short PSweep_GetMinDwell()

This function returns the minimum allowed dwell time in milliseconds. Dwell time is the length of time that the generator will pause at each power setting.

Return Values

Value	Description
0	Command failed
Time	Minimum dwell time in milliseconds

Examples

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

4.10.5. POWER SWEEP – GET FREQUENCY

Float PSweep_GetFreq()

This function returns the current frequency setting of the power sweep in MHz.

Return Values

Value	Description
0	Command failed
Frequency	Frequency in MHz

Examples

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

4.10.6. POWER SWEEP – GET START POWER

Double PSweep_GetStartPower()

This function returns the start power of the current power sweep in dBm.

Return Values

Value	Description
0	Command failed
Power	Start power in dBm

Examples

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

4.10.7. POWER SWEEP – GET STOP POWER

Double PSweep_GetStopPower()

This function returns the stop power of the current power sweep in dBm.

Return Values

Value	Description
0	Command failed
Power	Stop power in dBm

Examples

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

4.10.8. POWER SWEEP – GET STEP SIZE

Double PSweep_GetStepSize()

This function returns the step size in dBm of the current power sweep.

Return Values

Value	Description
0	Command failed
Power	Step size in dBm

Examples

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

4.10.9. POWER SWEEP – GET TRIGGER IN MODE

Short PSweep_GetTriggerIn()

This function returns the Trigger Input mode for the power sweep, this dictates how the generator will respond to an external trigger.

Return Values

Value	Description
0	Ignore Trigger In
1	Wait for Trigger In for each power setting
2	Wait for Trigger In before commencing each sweep

Examples

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

4.10.10. POWER SWEEP – GET TRIGGER OUT MODE

Short PSweep_GetTriggerOut()

This function returns Trigger Output mode for the power sweep, this dictates how the Trigger Out port will be used during the power sweep.

Return Values

Value	Description
0	Trigger Out disabled
1	Trigger Out set at each power
2	Trigger Out set as each sweep is initiated

Examples

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

4.10.11. POWER SWEEP – SET DIRECTION

Short PSweep_SetDirection(Short SweepDirection)

This function sets the direction of the power sweep.

Parameters

Variable	Description
0	Ascending power from start to stop
1	Descending power from stop to start
2	Ascending, then descending power

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.10.12. POWER SWEEP – SET DWELL TIME

Short PSweep_SetDwell(Short dwell_msec)

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each power setting.

Parameters

Variable	Description
dwell_msec	Required. The dwell time in milliseconds.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.PSweep_SetDwell(15)`
Visual Basic	`Status = MyPTE1.PSweep_SetDwell(15)`
Visual C++	`Status = MyPTE1->PSweep_SetDwell(15);`
Visual C#	`Status = MyPTE1.PSweep_SetDwell(15);`
MatLab	`Status = MyPTE1.PSweep_SetDwell(15);`

4.10.13. POWER SWEEP – START/STOP SWEEP

Short PSweep_SetMode(Short onoff)

This function starts or stops the power sweep using the previously defined parameters.

Note: The power sweep will stop automatically if any other command is sent.

Parameters

Variable	Description
1	Start power sweep
0	Stop power sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.10.14. POWER SWEEP – SET FREQUENCY

Short PSweep_SetFreq(Float Fr)

This function sets the output frequency in MHz to be used for the power sweep.

Parameters

Variable	Description
Fr	Required. The fixed frequency in MHz to be used for the power sweep.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.PSweep_SetFreq(1000)`
Visual Basic	`Status = MyPTE1.PSweep_SetFreq(1000)`
Visual C++	`Status = MyPTE1->PSweep_SetFreq(1000);`
Visual C#	`Status = MyPTE1.PSweep_SetFreq(1000);`
MatLab	`Status = MyPTE1.PSweep_SetFreq(1000);`

4.10.15. POWER SWEEP – SET START POWER

Short PSweep_SetStartPower(Float Pr)

This function sets the start power in dBm for the sweep.

Parameters

Variable	Description
Pr	Required. The start power in dBm.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.PSweep_SetStartPower(-10)`
Visual Basic	`Status = MyPTE1.PSweep_SetStartPower(-10)`
Visual C++	`Status = MyPTE1->PSweep_SetStartPower(-10);`
Visual C#	`Status = MyPTE1.PSweep_SetStartPower(-10);`
MatLab	`Status = MyPTE1.PSweep_SetStartPower(-10);`

4.10.16. POWER SWEEP – SET STOP POWER

Short PSweep_SetStopPower(Float Pr)

This function sets the stop power in dBm for the sweep.

Parameters

Variable	Description
Pr	Required. The stop power in dBm.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.PSweep_SetStopPower(10)`
Visual Basic	`Status = MyPTE1.PSweep_SetStopPower(10)`
Visual C++	`Status = MyPTE1->PSweep_SetStopPower(10);`
Visual C#	`Status = MyPTE1.PSweep_SetStopPower(10);`
MatLab	`Status = MyPTE1.PSweep_SetStopPower(10);`

4.10.17. POWER SWEEP – SET STEP SIZE

Short PSweep_SetStepSize(Float Pr)

This function sets the step size in dBm to be used in the power sweep.

Parameters

Variable	Description
Pr	Required. The step size in dBm.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Status = MyPTE1.PSweep_SetStepSize(0.5)`
Visual Basic	`Status = MyPTE1.PSweep_SetStepSize(0.5)`
Visual C++	`Status = MyPTE1->PSweep_SetStepSize(0.5);`
Visual C#	`Status = MyPTE1.PSweep_SetStepSize(0.5);`
MatLab	`Status = MyPTE1.PSweep_SetStepSize(0.5);`

4.10.18. POWER SWEEP – SET TRIGGER IN MODE

Short PSweep_SetTriggerIn(Short SweepTriggerIn)

This function specifies how the power sweep should respond to an external trigger. The modes are:

Parameters

Variable	Description
0	Ignore Trigger In
1	Wait for Trigger In before each power
2	Wait for Trigger In before commencing each sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.10.19. POWER SWEEP – SET TRIGGER OUT MODE

Short PSweep_SetTriggerOut(Short SweepTriggerOut)

This function specified how the Trigger Out port will be used during the power sweep.

Parameters

Variable	Description
0	Trigger Out disabled
1	Set Trigger Out (logic 1) at each power
2	Set Trigger Out (logic 1) on commencing each sweep

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.11. DLL – Frequency & Power Hop List

The signal generator can be configured to produce a fast, automatic frequency and power hop sequence to run unaided using the generator's internal memory and timing. This method allows sequences with very short dwell times (<1 ms) to be executed without the additional delays of USB / Ethernet communication at each step. The tradeoff with this method is that it can be more difficult to track the sweep progress from the control program.

For longer dwell times where the USB / Ethernet communication time is less significant (in the order of 5 ms or longer), the recommended method is to handle the loop and timing in the control program and simply issue commands to set the next state at the appropriate intervals. This method gives the user full control and monitoring of the sequence.

Example fast sequences using the DLL for USB control:

Python	`# Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.Hop_SetNoOfPoints(50)`
	`MyPTE1.Hop_SetDwell(10)`
	`# Set point 1 to 1000MHz, -10dBm`
	`MyPTE1.Hop_SetPoint (1, 1000, -10)`
	`# Set point 2 to 1100MHz, -8dBm`
	`MyPTE1.Hop_SetPoint (2, 1100, -8)`
	`# Set points 3 to 50 in the same way`
	`# Start the hop sequence`
	`MyPTE1.Hop_SetMode(1)`

Visual Basic	`' Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.Hop_SetNoOfPoints(50)`
	`MyPTE1.Hop_SetDwell(10)`
	`' Set point 1 to 1000MHz, -10dBm`
	`MyPTE1.Hop_SetPoint (1, 1000, -10)`
	`' Set point 2 to 1100MHz, -8dBm`
	`MyPTE1.Hop_SetPoint (2, 1100, -8)`
	`' Set points 3 to 50 in the same way`
	`' Start the hop sequence`
	`MyPTE1.Hop_SetMode(1)`

Visual C++	`// Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1->Hop_SetNoOfPoints(50);`
	`MyPTE1->Hop_SetDwell(10);`
	`// Set point 1 to 1000MHz, -10dBm`
	`MyPTE1->Hop_SetPoint (1, 1000, -10);`
	`// Set point 2 to 1100MHz, -8dBm`
	`MyPTE1->Hop_SetPoint (2, 1100, -8);`
	`// Index and set points 3 to 50 in the same way`
	`// Start the hop sequence`
	`MyPTE1->Hop_SetMode(1);`

Visual C#	`// Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.Hop_SetNoOfPoints(50);`
	`MyPTE1.Hop_SetDwell(10);`
	`// Set point 1 to 1000MHz, -10dBm`
	`MyPTE1.Hop_SetPoint (1, 1000, -10);`
	`// Set point 2 to 1100MHz, -8dBm`
	`MyPTE1.Hop_SetPoint (2, 1100, -8);`
	`// Index and set points 3 to 50 in the same way`
	`// Start the hop sequence`
	`MyPTE1.Hop_SetMode(1);`

MatLab	`% Declare a sequence of 50 points, set dwell time of 10ms`
	`MyPTE1.Hop_SetNoOfPoints(50)`
	`MyPTE1.Hop_SetDwell(10)`
	`% Set point 1 to 1000MHz, -10dBm`
	`MyPTE1.Hop_SetPoint (1, 1000, -10)`
	`% Set point 2 to 1100MHz, -8dBm`
	`MyPTE1.Hop_SetPoint (2, 1100, -8)`
	`% Index and set points 3 to 50 in the same way`
	`% Start the hop sequence`
	`MyPTE1.Hop_SetMode(1)`

4.11.1. FREQUENCY/POWER HOP – GET DIRECTION

Short Hop_GetDirection()

This function returns the direction setting for the current hop sequence.

Return Values

Value	Description
0	Ascending frequency from lowest to highest
1	Descending frequency from highest to lowest
2	Ascending frequency from lowest to highest, then descending to lowest

Examples

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

4.11.2. FREQUENCY/POWER HOP – GET DWELL TIME

Short Hop_GetDwell()

This function returns the dwell time setting in milliseconds for the current hop sequence.

Return Values

Value	Description
0	Command failed
Dwell Time	Dwell time setting in milliseconds

Examples

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

4.11.3. FREQUENCY/POWER HOP – GET MAXIMUM DWELL TIME

Short Hop_GetMaxDwell()

This function returns the maximum allowed dwell time in milliseconds for any point in a hop sequence.

Return Values

Value	Description
0	Command failed
Dwell Time	Maximum allowed dwell time in milliseconds

Examples

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

4.11.4. FREQUENCY/POWER HOP – GET MINIMUM DWELL TIME

Short Hop_GetMinDwell()

This function returns the minimum allowed dwell time in milliseconds for any point in a hop sequence.

Return Values

Value	Description
0	Command failed
Dwell Time	Minimum allowed dwell time in milliseconds

Examples

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

4.11.5. FREQUENCY/POWER HOP – GET NUMBER OF POINTS

Short Hop_GetNoOfPoints()

This function returns the number of points set for the current hop sequence.

Return Values

Value	Description
0	Command failed
Hops	Number of frequency hop points set

Examples

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

4.11.6. FREQUENCY/POWER HOP – GET MAXIMUM NUMBER OF POINTS

Short Hop_GetMaxNoOfPoints()

This function returns the maximum allowed number of points in a hop sequence.

Return Values

Value	Description
0	Command failed
Max Hops	Maximum number of frequency hop points

Examples

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

4.11.7. FREQUENCY/POWER HOP – GET HOP POINT

Short Hop_GetPoint(Short PointNo, ByRef Double HopFreq, ByRef Float HopPower)

This function returns the frequency and power settings for a specific point in a hop sequence, from 1 to the maximum allowed number of points (device specific, see Frequency/Power Hop – Get Maximum Number of Points).

Parameters

Variable	Description
PointNo	The point number; from 0 to (n - 1) where n equals the number of hop points in the sequence.
HopFreq	User defined variable to be updated with the frequency in MHz of the specified hop point.
HopPower	User defined variable to be updated with the power in dBm of the specified hop point.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`status = MyPTE1.Hop_GetPoint(PointNo, 0, 0)`
	`HopFreq = status[1]`
	`HopPower = status[2]`

Visual Basic	`status = MyPTE1.Hop_GetPoint(PointNo, HopFreq, HopPower)`
Visual C++	`status = MyPTE1->Hop_GetPoint(PointNo, HopFreq, HopPower);`
Visual C#	`status = MyPTE1.Hop_GetPoint(PointNo, ref(HopFreq), ref(HopPower));`
MatLab	`[HopFreq, HopPower] = MyPTE1.Hop_GetPoint(PointNo, HopFreq, HopPower)`

4.11.8. FREQUENCY/POWER HOP – GET TRIGGER IN MODE

Short Hop_GetTriggerIn()

This function returns the Trigger Input mode for the hop sequence, this dictates how the generator will respond to an external trigger.

Return Values

Value	Description
0	Ignore Trigger In
1	Wait for Trigger In before hopping to next point
2	Wait for Trigger In before starting hop sequence

Examples

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

4.11.9. FREQUENCY/POWER HOP – GET TRIGGER OUT MODE

Short Hop_GetTriggerOut()

This function returns the Trigger Output mode for the hop sequence, this dictates how the Trigger Out port will be used during the frequency sweep.

Return Values

Value	Description
0	Trigger Out disabled
1	Trigger Out (logic 1) set at each point
2	Trigger Out (logic 1) set as the hop is initiated

Examples

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

4.11.10. FREQUENCY/POWER HOP – SET DIRECTION

Short Hop_SetDirection(Short HopDirection)

This function sets the direction of the hop sequence.

Parameters

Variable	Description
0	Ascending from first to last
1	Descending from last to first
2	Ascending from first to last, then descending from last to first

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.11.11. FREQUENCY/POWER HOP – SET DWELL TIME

Short Hop_SetDwell(Short dwell_msec)

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each point in the hop sequence.

Parameters

Variable	Description
dwell_msec	Required. The dwell time in milliseconds.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Result = MyPTE1.Hop_SetDwell(15)`
Visual Basic	`Result = MyPTE1.Hop_SetDwell(15)`
Visual C++	`Result = MyPTE1->Hop_SetDwell(15);`
Visual C#	`Result = MyPTE1.Hop_SetDwell(15);`
MatLab	`Result = MyPTE1.Hop_SetDwell(15);`

4.11.12. FREQUENCY/POWER HOP – START/STOP HOP SEQUENCE

Short Hop_SetMode(Short onoff)

This function starts or stops the hop sequence using the previously defined parameters.

Note: The hop sequence will stop automatically if any other command is sent.

Parameters

Variable	Description
1	Start the hop sequence
0	Stop the hop sequence

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.11.13. FREQUENCY/POWER HOP – SET NUMBER OF POINTS

Short Hop_SetNoOfPoints(Short NoOfPoints)

This function sets the number of points to be used in the hop sequence.

Parameters

Variable	Description
NoOfPoints	Required. The number of points to hop

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Result = MyPTE1.Hop_SetNoOfPoints(10)`
Visual Basic	`Result = MyPTE1.Hop_SetNoOfPoints(10)`
Visual C++	`Result = MyPTE1->Hop_SetNoOfPoints(10);`
Visual C#	`Result = MyPTE1.Hop_SetNoOfPoints(10);`
MatLab	`Result = MyPTE1.Hop_SetNoOfPoints(10);`

4.11.14. FREQUENCY/POWER HOP – SET HOP POINT

Short Hop_SetPoint(Short PointNo, Double HopFreq, Float HopPower)

This function sets the frequency and power for a specific point in the hop sequence.

Parameters

Variable	Description
PointNo	Required. The point number; 0 for the first point in the sequence, 1 for the second, up (n - 1) for the final point where n equals the maximum number of points.
HopFreq	Required. The frequency in MHz.
HopPower	Required. The power in dBm.

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python	`Result = MyPTE1.Hop_SetPoint(3, 1000, 10)`
Visual Basic	`Result = MyPTE1.Hop_SetPoint(3, 1000, 10)`
Visual C++	`Result = MyPTE1->Hop_SetPoint(3, 1000, 10);`
Visual C#	`Result = MyPTE1.Hop_SetPoint(3, 1000, 10);`
MatLab	`Result = MyPTE1.Hop_SetPoint(3, 1000, 10);`

4.11.15. FREQUENCY/POWER HOP – SET TRIGGER IN MODE

Short Hop_SetTriggerIn(Short HopTriggerIn)

This function specifies how the hop sequence should respond to an external trigger.

Parameters

Variable	Description
0	Ignore Trigger In
1	Wait for Trigger In before each hop
2	Wait for Trigger In before starting each hop sequence

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.11.16. FREQUENCY/POWER HOP – SET TRIGGER OUT MODE

Short Hop_SetTriggerOut(Short HopTriggerOut)

This function specified how the Trigger Out port will be used during the hop sequence.

Parameters

Variable	Description
0	Trigger Out disabled
1	Set Trigger Out (logic 1) at each point
2	Set Trigger Out (logic 1) on starting each hop sequence

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.12. DLL - Ethernet Configuration

The following functions provide a method to review and edit the Ethernet TCP / IP connection parameters. Refer to Ethernet Control API for further details on Ethernet control of the device and default settings.

4.12.1. GET ETHERNET CONFIGURATION

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

Python

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

Visual Basic

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

Visual C++

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

Visual C#

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

MatLab

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

4.12.2. GET DHCP STATUS

int GetEthernet_UseDHCP()

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

Return Values

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

Examples

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

4.12.3. USE DHCP

int SaveEthernet_UseDHCP(int UseDHCP)

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

Parameters

Variable	Description
0	DHCP disabled (static IP settings used)
1	DHCP enabled (IP setting assigned by network)

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.12.4. GET IP ADDRESS

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

Returns the user-entered static IP address.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.12.5. SAVE IP *ADDRESS*

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.12.6. GET MAC ADDRESS

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.12.7. GET NETWORK GATEWAY

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

Returns the user-entered network gateway IP address.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.12.8. SAVE NETWORK GATEWAY

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.12.9. GET SUBNET MASK

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

Returns the user-entered subnet mask.

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

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

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

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

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

4.12.10. SAVE SUBNET MASK

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

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

Parameters

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

Return Values

Value	Description
0	Command failed
1	Command completed successfully

Examples

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

4.12.11. GET TCP/IP PORT

int GetEthernet_TCPIPPort(ByRef int port)

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

Parameters

Variable	Description
port	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`

4.12.12. 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.

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);`

4.12.13. GET TELNET PORT

int GetEthernet_TelnetPort(ByRef int port)

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

Parameters

Variable	Description
port	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`

4.12.14. 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.

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);`

4.12.15. 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

Model Name	Requirements
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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`

4.12.16. 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

Model Name	Requirements
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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);`

4.12.17. GET SSH LOGIN NAME

int GetEthernet_SSHLoginName(ByRef string SSHLoginName)

Returns the login name for SSH communication.

Applies To

Model Name	Requirements
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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`

4.12.18. SAVE SSH LOGIN NAME

int SaveEthernet_SSHLoginName(string SSHLoginName)

Sets the login name for SSH communication.

Applies To

Model Name	Requirements
SSG-15G-RC	All units
SSG-30G(HP)-RC	All units
SSG-44G(HP)-RC	All units

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');`

4.12.19. GET PASSWORD REQUIREMENT

int GetEthernet_UsePWD()

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

Return Values

Value	Description
0	Password not required
1	Password required

Examples

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

4.12.20. SET PASSWORD REQUIREMENT

int SaveEthernet_UsePWD(int UsePwd)

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

Parameters

Variable	Description
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);`

4.12.21. GET PASSWORD

int GetEthernet_PWD(ByRef string Pwd)

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

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`

4.12.22. 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.

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");`

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 Mini-Circuits RF switch matrix series, the Vendor ID and Product ID are required:

- Vendor ID: 0x20CE
- Product ID: 0x12

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

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

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

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

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. Common Commands

5.2.1. GET DEVICE MODEL NAME

Description

Returns the full Mini-Circuits part number of the signal generator.

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

Examples

The following array would be returned for Mini-Circuits' SSG-4000HP signal generator. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5
Value	40	83	83	71	45	52
ASCII Character	N/A	S	S	G	-	4

Byte	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
Description	Char 6	Char 7	Char 8	Char 9	Char 10	End Marker
Value	48	48	48	72	80	0
ASCII Character	0	0	0	H	P	N/A

5.2.2. GET DEVICE SERIAL NUMBER

Description

Returns the serial number of the connected signal generator.

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

Examples

The following example indicates that the current signal generator has serial number 1100040023. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5
Value	41	49	49	48	48	48
ASCII Character	N/A	1	1	0	0	0

Byte	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
Description	Char 6	Char 7	Char 8	Char 9	Char 10	End Marker
Value	52	48	48	50	51	0
ASCII Character	4	0	0	2	3	N/A

5.2.3. SET FREQUENCY AND POWER (SSG-6000 SERIES)

Sets the RF output frequency and power level of the signal generator and enables or disables the "trigger out" function.

Applies To

SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	103	Interrupt code for Set Frequency and Power
1	Freq_1	Frequency (Hz), split over 5 bytes Freq_1 = INT (Freq / 256^4)
2	Freq_2	Frequency (Hz), split over 5 bytes: REMAINDER1 = Freq - Freq_1* (256^4) Freq_2 = INT (REMAINDER1 / 256^3)
3	Freq_3	Frequency (Hz), split over 5 bytes: REMAINDER2 = REMAINDER1 - Freq_2* (256^3) Freq_3 = INT (REMAINDER2 / 256^2)
4	Freq_4	Frequency (Hz), split over 5 bytes REMAINDER3 = REMAINDER2 - Freq_3* (256^2) Freq_4 = INT (REMAINDER3 / 256)
5	Freq_5	Frequency (Hz), split over 5 bytes Freq_5 = INT (REMAINDER3 - Freq_4* 256)
6	Power (+/-)	Power polarity (0 = positive; 1 = negative)
7	Power_1	Power magnitude (dBm), split over 2 bytes: Power_1 = INT (Magnitude * 100 / 256)
8	Power_2	Power magnitude (dBm), split over 2 bytes: Power_2 = Magnitude * 100 - (Power_1* 256)
9	Trigger_Out	Trigger Out status (0 = disabled; 1 = enabled)

Returned Array

Byte	Data	Description
0	103	Interrupt code for Set Frequency and Power
1 - 63	Not significant	

Examples

The following transmit array would set the generator output to 5501.56MHz with a power level of +4.5dBm and enable the Trigger Out:

Byte	Data	Description	
0	103	Interrupt code for Set Frequency and Power	
1	1	Freq_1	= INT (5,501,560,000 / 256^4) = INT (1.2809) = 1
2	71	REMAINDER1 Freq_2	= 5,501,560,000 - 1 * (256^4) = 1,206,592,704 = INT (1,206,592,704 / 256^3) = 71
3	235	REMAINDER2 Freq_3	= 1,206,592,704 - 71 * (256^3) = 60,196 = INT (60,196 / 256^2) = 235
4	36	REMAINDER3 Freq_4	= 1,206,592,704 - 71 * (256^2) = 9,408 = INT (9,408 / 256) = 36
5	192	Freq_5	= INT (9,408 - (36 * 256)) = 192
6	0	Power level is positive (dBm)	
7	1	BYTE7	= INT ((4.5 * 100) / 256) = INT (1.76) = 1
8	194	BYTE8	= (4.5 *100) - (1 * 256) = 194
9	1	Enable Trigger Out	

5.2.4. SET FREQUENCY AND POWER (SSG-XG SERIES)

Sets the RF output frequency and power level of the signal generator .

Applies To

All models except SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	103	Set frequency and power
1 to 19	Frequency	The frequency in MHz as an ASCII string, using the ASCII code for each character, 1 per byte. The final byte in the sequence should have value 0 and then any subsequent bytes up to 19 are not considered.
20 to (n-1)	Power	The power in dBm as an ASCII string, using the ASCII code for each character, 1 per byte.
n	0	Indicates the end of the ASCII power string

Returned Array

Byte	Data	Description
0	103	Set frequency and power
1 - 63	Not significant	

Examples

The following transmit array would set the generator output to 4100.55MHz with a power level of +4.5dBm:

Byte	Data	Description
0	103	Set frequency and power
1	52	ASCII character code for "4"
2	49	ASCII character code for "1"
3	48	ASCII character code for "0"
4	48	ASCII character code for "0"
5	46	ASCII character code for "."
6	53	ASCII character code for "5"
7	53	ASCII character code for "5"
8	0	Indicates the end of the ASCII frequency string
9	0	Not considered
10	0	Not considered
11	0	Not considered
12	0	Not considered
13	0	Not considered
14	0	Not considered
15	0	Not considered
16	0	Not considered
17	0	Not considered
18	0	Not considered
19	0	Not considered
20	43	ASCII character code for "+"
21	52	ASCII character code for "4"
22	46	ASCII character code for "."
23	53	ASCII character code for "5"
24	0	Indicates the end of the ASCII power string

5.2.5. SET FREQUENCY (SSG-6000 SERIES)

Sets the RF output frequency of the signal generator and enables or disables the "trigger out" function.

Applies To

SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	101	Set frequency in MHz
1	Freq_1	Frequency (Hz), split over 5 bytes Freq_1 = INT (Frequency / 256^4)
2	Freq_2	Frequency (Hz), split over 5 bytes: REMAINDER1 = Frequency - Freq_1 * (256^4) Freq_2 = INT (REMAINDER1 / 256^3)
3	Freq_3	Frequency (Hz), split over 5 bytes: REMAINDER2 = REMAINDER1 - Freq_2 * (256^3) Freq_3 = INT (REMAINDER2 / 256^2)
4	Freq_4	Frequency (Hz), split over 5 bytes REMAINDER3 = REMAINDER2 - Freq_3 * (256^2) Freq_4 = INT (REMAINDER3 / 256)
5	Freq_5	Frequency (Hz), split over 5 bytes Freq_5 = INT (REMAINDER3 - Freq_4 * 256)
6	Trigger_Out	Trigger Out status: 0 = Disable trigger output 1 = Enable trigger output

Returned Array

Byte	Data	Description
0	101	Set frequency in MHz
1-63	Not significant	

Example

The following transmit array would set the generator output to 4100.55MHz and enable Trigger Out:

Byte	Data	Description	
0	101	Set frequency in MHz	
1	0	BYTE1	= INTEGER (4,100,550,000 / 256 ^ 4) = INTEGER (0.9547) = 0
2	244	REMAINDER1 BYTE2	= 4,100,550,000 - 0 * (256 ^ 4) = 4,100,550,000 = INTEGER (4,100,550,000 / 256 ^ 3) = 244
3	105	REMAINDER2 BYTE3	= 4,100,550,000 - 244 * (256 ^ 3) = 6,909,296 = INTEGER (6,909,296 / 256 ^ 2) = 105
4	109	REMAINDER3 BYTE4	= 6,909,296 - 105 * (256 ^ 2) = 28,016 = INTEGER (28,016 / 256) = 109
5	112	BYTE5	= INTEGER (28,016 - (109 * 256)) = 112
6	1	Enable Trigger Out	

5.2.6. SET FREQUENCY (SSG-XG SERIES)

Sets the RF output frequency of the signal generator.

Applies To

All models except SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	101	Set frequency in MHz
1 to (n-1)	Frequency	The frequency in MHz as an ASCII string, using the ASCII code for each character, 1 per byte.
n	0	Indicates the end of the ASCII string

Returned Array

Byte	Data	Description
0	101	Set frequency in MHz
1-63	Not significant	

Example

The following transmit array would set the generator output to 4100.55MHz:

Byte	Data	Description
0	101	Set frequency in MHz
1	52	ASCII character code for "4"
2	49	ASCII character code for "1"
3	48	ASCII character code for "0"
4	48	ASCII character code for "0"
5	46	ASCII character code for "."
6	53	ASCII character code for "5"
7	53	ASCII character code for "5"
8	0	Indicate the end of the ASCII string

5.2.7. SET POWER

Sets the RF output power of the signal generator in dBm and enables or disables the "trigger out" function.

Transmit Array

Byte	Data	Description
0	102	Interrupt code for Set Power
1	Power (+/-)	Power polarity: Positive (Power = 1 * Power) Negative (Power = -1 * Power)
2	Power_1	Power magnitude (dBm), split over 2 bytes: BYTE2 = INT (Magnitude * 100 / 256)
3	Power_2	Power magnitude (dBm), split over 2 bytes: BYTE3 = Magnitude * 100 - (BYTE2 * 256)
4	Trigger_Out	Trigger Out status: Disable trigger output Enable trigger output

Returned Array

Byte	Data	Description
0	102	Interrupt code for Set Power
1 - 63	Not significant	

Examples

To following transmit array would set the generator output power to -5.5dBm and enable the Trigger Out:

Byte	Data	Description
0	102	Interrupt code for Set Power
1	1	Power value is negative
2	2	BYTE2 = INT ((5.5 * 100) / 256) = INT (2.15) = 2
3	38	BYTE3 = (5.5 *100) - (2 * 256) = 38
4	1	Enable Trigger Out

5.2.8. SET RF POWER ON/OFF

Description

This function enables or disables the RF output of the signal generator.

Send code 104 in BYTE0 of the transmit array with BYTE1 as 1 to enable or 0 to disable the RF output. BYTE2 to BYTE63 are "don't care" bytes and can be any value.

The returned array contains 104 in BYTE0. BYTE1 to BYTE63 are "don't care" bytes and could be any value.

Transmit Array

Byte	Data	Description
0	104	Interrupt code for Set RF Power On/Off
1	Status	RF output status: Disable RF output Enable RF output
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	104	Interrupt code for Set RF Power On/Off

Examples

The below transmit array enables the RF output with the previously defined frequency and power levels:

Byte	Data	Description
0	104	Interrupt code for Set RF Power On/Off
1	1	Enable the RF output

The below transmit array disables the RF output:

Byte	Data	Description
0	104	Interrupt code for Set RF Power On/Off
1	0	Disable the RF output

5.2.9. GET GENERATOR OUTPUT STATUS

Description

Returns the current output status of the signal generator, covering the following parameters:

Generator lock status (locked/unlocked)

- RF output status (on/off)
- Current output frequency
- Current output power
- Current output power relative to user requested level

Transmit Array

Byte	Data	Description
0	105	Interrupt code for Get Generator Output Status
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	105	Interrupt code for Get Generator Output Status
1	Output Status	RF output is disabled RF output is enabled
2	Lock Status	Frequency is not locked Frequency is locked
3	Freq_1	Frequency (Hz), split over 5 bytes: Frequency = (256^4) * BYTE3 + (256^3) * BYTE4 + (256^2) * BYTE5 + 256 * BYTE6 + BYTE7
4	Freq_2	Frequency (Hz), split over 5 bytes
5	Freq_3	Frequency (Hz), split over 5 bytes
6	Freq_4	Frequency (Hz), split over 5 bytes
7	Freq_5	Frequency (Hz), split over 5 bytes
8	Power (+/-)	Power polarity: Positive (Power = 1 * Magnitude) Negative (Power = -1 * Magnitude)
9	Power_1	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * BYTE9 + BYTE10) / 100
10	Power_2	Power magnitude (dBm), split over 2 bytes
11	Unlevel_High	High power request warning: User requested power level within generator capability User requested power level is higher than the generator can achieve
12	Unlevel_Low	Low power request warning: User requested power level within generator capability

		User requested power level is lower than the generator can achieve
13 - 63	Not used	"Don't care" bytes, could be any value

Examples

The following returned array indicates that the generator was set with the output enabled at 4980.50MHz, +5.5dBm and the power level is within the generator's capability:

Byte	Data	Description
0	105	Interrupt code for Get Generator Output Status
1	1	RF output enabled
2	1	Frequency is locked
3	1	Frequency (Hz), split over 5 bytes: Frequency = 256^4 * 1 + 256^3 * 40 + 256^2 * 220 + 256 * 102 + 32 = 751,250,000 Hz = 751.25 MHz
4	40	Frequency (Hz), split over 5 bytes
5	220	Frequency (Hz), split over 5 bytes
6	102	Frequency (Hz), split over 5 bytes
7	32	Frequency (Hz), split over 5 bytes
8	0	Power value is positive
9	2	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * 2 + 38) / 100 = 5.5dBm
10	38	Power magnitude (dBm), split over 2 bytes
11	0	Power level within range
12	0	Power level within range

5.2.10. GET GENERATOR MINIMUM FREQUENCY

Description

Returns the signal generator minimum frequency specification in Hz.

Transmit Array

Byte	Data	Description
0	42	Interrupt code for Get Generator Minimum Frequency
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	42	Interrupt code for Get Generator Minimum Frequency
1	Freq_1	Frequency (Hz), split over 4 bytes: Frequency = (256^3) * BYTE1 + (256^2) * BYTE2 + 256 * BYTE3 + BYTE4
2	Freq_2	Frequency (Hz), split over 4 bytes
3	Freq_3	Frequency (Hz), split over 4 bytes
4	Freq_4	Frequency (Hz), split over 4 bytes
5 - 63	Not Used	"Don't care" bytes, could be any value

Examples

The following array would be returned for SSG-4000HP:

Byte	Data	Description
0	42	Interrupt code for Get Generator Minimum Frequency
1	14	Frequency (Hz), split over 4 bytes: FREQUENCY = 256^3 * 14 + 256^2 * 230 + 256 * 178 + 128 = 250,000,000 Hz = 250 MHz
2	220	Frequency (Hz), split over 4 bytes
3	178	Frequency (Hz), split over 4 bytes
4	128	Frequency (Hz), split over 4 bytes

5.2.11. GET GENERATOR MAXIMUM FREQUENCY

Description

Returns the maximum frequency specification in Hz.

Transmit Array

Byte	Data	Description
0	43	Interrupt code for Get Generator Maximum Frequency
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	43	Interrupt code for Get Generator Maximum Frequency
1	Freq_1	Frequency (Hz), split over 5 bytes: Frequency = 256^4 * BYTE1 + 256^3 + BYTE2 + 256^2 * BYTE3 + 256 * BYTE4 + BYTE5
2	Freq_2	Frequency (Hz), split over 5 bytes
3	Freq_3	Frequency (Hz), split over 5 bytes
4	Freq_4	Frequency (Hz), split over 5 bytes
5	Freq_5	Frequency (Hz), split over 5 bytes
6 - 63	Not Used	"Don't care" bytes, could be any value

Examples

The following array would be returned for SSG-6001RC:

Byte	Data	Description
0	43	Interrupt code for Get Generator Maximum Frequency
1	1	Frequency (Hz), split over 5 bytes: Frequency = 256^4 * 1 + 256^3 * 101 + 256^2 * 160 + 256 * 188 + 0 = 6,000,000,000 Hz = 6,000 MHz
2	101	Frequency (Hz), split over 5 bytes
3	160	Frequency (Hz), split over 5 bytes
4	188	Frequency (Hz), split over 5 bytes
5	0	Frequency (Hz), split over 5 bytes

5.2.12. GET GENERATOR STEP SIZE

Description

Returns the signal generator's minimum step size in Hz.

Transmit Array

Byte	Data	Description
0	44	Interrupt code for Get Generator Step Size
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	44	Interrupt code for Get Generator Step Size
1	Freq_1	Frequency (Hz), split over 4 bytes: Frequency = (256^3) * BYTE1 + (256^2) * BYTE2 + 256 * BYTE3 + BYTE4
2	Freq_2	Frequency (Hz), split over 4 bytes
3	Freq_3	Frequency (Hz), split over 4 bytes
4	Freq_4	Frequency (Hz), split over 4 bytes
5 - 63	Not Used	"Don't care" bytes, could be any value

Examples

The following array would be returned for SSG-4000HP:

Byte	Data	Description
0	44	Interrupt code for Get Generator Step Size
1	0	Frequency (Hz), split over 4 bytes: FREQUENCY = 256^3 * 0 + 256^2 * 0 + 256 * 19 + 136 = 5,000 Hz = 5 KHz
2	0	Frequency (Hz), split over 4 bytes
3	19	Frequency (Hz), split over 4 bytes
4	136	Frequency (Hz), split over 4 bytes

5.2.13. GET GENERATOR MINIMUM POWER

Description

Returns the generator's minimum output power specification in dBm. The minimum output power achievable by the generator is guaranteed to be at least as low as this specified level across the full operating frequency and will be even lower in some frequency bands.

Transmit Array

Byte	Data	Description
0	45	Interrupt code for Get Generator Minimum Power
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	45	Interrupt code for Get Generator Minimum Power
1	Power (+/-)	Power polarity: Positive (Power = 1 * Magnitude) Negative (Power = -1 * Magnitude)
2	Power_1	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * BYTE2 + BYTE3) / 100
3	Power_2	Power magnitude (dBm), split over 2 bytes
4 - 63	Not used	"Don't care" bytes, could be any value

Examples

The following array would be returned for SSG-4000HP:

Byte	Data	Description
0	45	Interrupt code for Get Generator Minimum Power
1	1	Power value is negative
2	19	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * 19 + 136) / 100 = 50 Power = -50 dBm
3	136	Power magnitude (dBm), split over 2 bytes

5.2.14. GET GENERATOR MAXIMUM POWER

Description

Returns the generator's maximum output power specification in dBm. The maximum output power achievable by the generator is guaranteed to be at least as high as this specified level across the full operating frequency and will be even higher in some frequency bands.

Transmit Array

Byte	Data	Description
0	46	Interrupt code for Get Generator Maximum Power
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	46	Interrupt code for Get Generator Maximum Power
1	Power (+/-)	Power polarity: Positive (Power = 1 * Magnitude) Negative (Power = -1 * Magnitude)
2	Power_1	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * BYTE2 + BYTE3) / 100
3	Power_2	Power magnitude (dBm), split over 2 bytes
4 - 63	Not used	"Don't care" bytes, could be any value

Examples

The following array would be returned for SSG-4000HP:

Byte	Data	Description
0	46	Interrupt code for Get Generator Maximum Power
1	0	Power value is negative
2	7	Power magnitude (dBm), split over 2 bytes: Magnitude = (256 * 7 + 208) / 100 = 20 Power = +20dBm
3	208	Power magnitude (dBm), split over 2 bytes

5.2.15. CHECK EXTERNAL REFERENCE

Description

Indicates which reference source is currently in use. The signal generator will automatically switch from internal to external reference if a valid signal is detected at the Ref In port.

Transmit Array

Byte	Data	Description
0	47	Interrupt code for Check External Reference
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	47	Interrupt code for Check External Reference
1	Ref_Source	The reference source: Internal reference in use External reference in use

Examples

The below returned array indicates an external reference source is connected at the signal generator's Ref In port and is currently in use:

Byte	Data	Description
0	47	Interrupt code for Check External Reference
1	1	External reference in use

5.2.16. GET FIRMWARE

Description

Returns the internal firmware version of the signal generator.

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

Examples

The following returned array indicates that the signal generator 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

5.2.17. SEND SCPI COMMAND (SSG-6000 SERIES)

Sends a SCPI command to the signal generator and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products.

Applies To

SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	121	Interrupt code for Send SCPI Command
1	SCPI Length	The length (number of ASCII characters) of the SCPI string to send
2 to 63	SCPI String	The SCPI command to be sent represented as a series of ASCII character codes, one character code per byte

Returned Array

Byte	Data	Description
0	121	Interrupt code for Send SCPI Command
1	SCPI Length	The length (number of ASCII characters) of the SCPI command sent in the transmit array
2 to 7	SCPI String	Bytes 2 to 7 of the transmit array repeated
8 to (n-1)	SCPI Response	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	

Examples

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 2 to 5 of the transmit array as follows:

Byte	Data	Description
0	121	Interrupt code for Send SCPI Command
1	4	Length of the SCPI command (four ASCII characters)
2	49	ASCII character code for :
3	77	ASCII character code for M
4	78	ASCII character code for N
5	63	ASCII character code for ?

The returned array for SSG-6001RC would be as follows:

Byte	Data	Description
0	121	Interrupt code for Send SCPI Command
1	4	Length of the SCPI command (four ASCII characters) from the transmit array
2	49	Repeated from the transmit array (ASCII character code)
3	77	Repeated from the transmit array (ASCII character code)
4	78	Repeated from the transmit array (ASCII character code)
5	63	Repeated from the transmit array (ASCII character code)
6	0	Repeated from the transmit array (unused byte)
7	0	Repeated from the transmit array (unused byte)
8	83	ASCII character code for S
9	83	ASCII character code for S
10	81	ASCII character code for G
11	45	ASCII character code for -
12	54	ASCII character code for 6
13	48	ASCII character code for 0
14	48	ASCII character code for 0
15	49	ASCII character code for 1
16	82	ASCII character code for R
17	67	ASCII character code for C
18	0	Zero value byte to indicate end of string

5.2.18. SEND SCPI COMMAND (SSG-XG SERIES)

Sends a SCPI command to the signal generator and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products.

Applies To

All models except SSG-6000RC & SSG-6001RC

Transmit Array

Byte	Data	Description
0	42	Send SCPI command
1 to 63	SCPI String	The SCPI command to be sent represented as a series of ASCII character codes, one character code per byte

Returned Array

Byte	Data	Description
0	42	Send SCPI command
8 to (n-1)	SCPI Response	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	

Examples

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	42	Send SCPI command
1	49	ASCII character code for :
2	77	ASCII character code for M
3	78	ASCII character code for N
4	63	ASCII character code for ?

The returned array for SSG-15G-RC would be as follows:

Byte	Data	Description
0	42	Send SCPI command
8	83	ASCII character code for S
9	83	ASCII character code for S
10	81	ASCII character code for G
11	45	ASCII character code for -
12	49	ASCII character code for 1
13	53	ASCII character code for 5
14	81	ASCII character code for G
15	45	ASCII character code for -
16	82	ASCII character code for R
17	67	ASCII character code for C
18	0	Zero value byte to indicate end of string

5.3. Pulse Modulation Functions

The signal generator can be configured to produce a pulsed output, either in response to an external trigger or continuously using the generator's internal timing systems.

Full details of the commands for configuring a pulsed output are covered in the following sections.

5.3.1. SET PULSE MODE

Description

Creates a pulsed output with a user specified pulse duration and time period. The output during the pulse "on" period is a CW signal with a frequency and power level which should be set by the user in advance. The pulse period will repeat indefinitely until any other command is received by the signal generator.

Transmit Array

Byte	Data	Description
0	117	Interrupt code for Set Pulse Mode
1	Off_Time	The pulse off time split over 2 bytes: BYTE1 = INT (Off_Time / 256)
2	Off_Time	The pulse off time split over 2 bytes: BYTE2 = Off_Time - (BYTE1 * 256)
3	On_Time	The pulse duration (on time) split over 2 bytes: BYTE3 = INT (On_Time / 256)
4	On_Time	The pulse duration (on time) split over 2 bytes: BYTE4 = On_Time - (BYTE3 * 256)
5	Time_Units	Units for On_Time and Off_Time: microseconds (µs) milliseconds (ms)
6 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	117	Interrupt code for Set Pulse Mode

Examples

After configuring the frequency and power of the CW signal to be used during the pulse "on" time, to enable the pulsed output with an on period of 10µs and off period of 1000µs, use the below transmit array:

Byte	Data	Description
0	117	Interrupt code for Set Pulse Mode
1	3	Off_Time = 1000µs BYTE1 = INT (1000 / 256) = 3
2	232	Off_Time = 1000µs BYTE2 = 1000 - (3 * 256) = 232
3	0	On_Time = 10µs BYTE3 = INT (10 / 256) = 0
4	10	On_Time = 10µs BYTE4 = 10 - (0 * 256) = 10
5	0	On_Time and Off_Time are expressed in microseconds (µs)

5.3.2. SET TRIGGERED PULSE MODE

Description

Configures a pulsed output with user specified pulse duration that will start when an external trigger is received at the "Trigger In" input. The output during the pulse "on" period is a CW signal with a frequency and power level which should be set by the user in advance. The pulsed output will be enabled until any other command is received by the signal generator.

Transmit Array

Byte	Data	Description
0	118	Interrupt code for Set Triggered Pulse Mode
1	Trigger_Type	The trigger type to use: 0 = Trigger on the falling edge 1 = Trigger on the rising edge
2	0	Zero value byte
3	On_Time (MSB)	The pulse duration (on time) split over 2 bytes: BYTE3 = INT (On_Time / 256)
4	On_Time (LSB)	The pulse duration (on time) split over 2 bytes: BYTE4 = On_Time - (BYTE3 * 256)
5	Time_Units	Units for On_Time and Off_Time: microseconds (µs) milliseconds (ms)
6 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	118	Interrupt code for Set Triggered Pulse Mode

Examples

The below transmit array will enable a pulsed output with 25µs, to be triggered at the falling edge of an external trigger:

Byte	Data	Description
0	118	Interrupt code for Set Triggered Pulse Mode
1	0	Trigger on the falling edge
2	0	Zero value byte
3	0	On_Time = 25µs BYTE3 = INT (25 / 256) = 0
4	25	On_Time = 25µs BYTE4 = 25 - (0 * 256) = 25
5	0	On_Time and Off_Time are expressed in microseconds (µs)

5.3.3. SET EXTERNAL PULSE MODULATION MODE

Description

Enables a pulsed output in response to the generator's Trigger In port. The generator's CW output will be enabled with the specified frequency and power while the Trigger In port is held high. The output will be disabled while the Trigger In port is held low. The CW frequency and power for the "on" period should be set in advance and the external pulse modulation mode will be disabled when any other command is sent to the generator.

Transmit Array

Byte	Data	Description
0	128	Interrupt code for Set External Pulse Modulation Mode
1 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	128	Interrupt code for Set External Pulse Modulation Mode

Examples

After configuring the frequency and power of the CW signal to be used during the pulse "on" time, external modulation is enable with the below transmit array:

Byte	Data	Description
0	128	Interrupt code for Set External Pulse Modulation Mode

5.4. Frequency/Power Hop Functions

These functions define the frequency and power hop capabilities of the generators. The signal generator can be configured to automatically hop through a series of user defined frequency and power outputs using the generator's internal timing systems. The user stores the parameters of the hop sequence in the generator's memory and can then enable/disable the output as required.

Full details of the specific commands are covered in the following sections.

5.4.1. FREQUENCY/POWER HOP - GET HOP DIRECTION

Description

Returns the direction that the generator will run through the list of hop values.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	7	Code for Get Hop Direction
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_Direction	The direction set for the current hop sequence: 0 = From first to last in the list 1 = From last to first in the list 2 = From first to last in the list, then back from last to first

Examples

The below returned array indicates that the signal generator will hop bi-directionally through the list, from start to finish, then back to start:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	2	The generator will set the frequency/power values in the hop list from first to last, then last to first

5.4.2. FREQUENCY/POWER HOP - GET HOP DWELL TIME

Description

Returns the dwell time to be used by the generator between each frequency/power hop point in the sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	4	Code for Get Hop Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_Dwell	The dwell time in milliseconds split over 2 bytes: Hop_Dwell = (256 * BYTE1) + BYTE2
2	Hop_Dwell	The dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's dwell time is set to 300ms:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	1	The dwell time in milliseconds split over 2 bytes: Hop_Dwell = (256 * 1) + 44 = 300 ms
2	44	The dwell time in milliseconds split over 2 bytes

5.4.3. FREQUENCY/POWER HOP - GET MAXIMUM HOP DWELL TIME

Description

Returns the maximum allowed dwell time in milliseconds for any point in a hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	5	Code for Get Maximum Hop Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes: Hop_MaxDwell = (256 * BYTE1) + BYTE2
2	Hop_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's maximum dwell time is 10,000ms (10s):

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	39	The maximum dwell time in milliseconds split over 2 bytes: Hop_MaxDwell = (256 * 39) + 16 = 10,000 ms
2	16	The maximum dwell time in milliseconds split over 2 bytes

5.4.4. FREQUENCY/POWER HOP - GET MINIMUM HOP DWELL TIME

Description

Returns the minimum allowed dwell time in milliseconds for any point in a hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	6	Code for Get Minimum Hop Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_MinDwell	The minimum dwell time in milliseconds split over 2 bytes: Hop_MinDwell = (256 * BYTE1) + BYTE2
2	Hop_MinDwell	The minimum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's minimum dwell time is 20ms:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	0	The minimum dwell time in milliseconds split over 2 bytes: Hop_MinDwell = (256 * 0) + 20 = 20 ms
2	20	The minimum dwell time in milliseconds split over 2 bytes

5.4.5. FREQUENCY/POWER HOP - GET MAXIMUM NUMBER OF HOP POINTS

Description

Returns the maximum number of points allowed in the hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	10	Code for Get Maximum Number of Hop Points
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_MaxPoints	The maximum number of points allowed in the hop sequence

Examples

The below returned array indicates that the maximum number of hop points allowed is 60,000:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	60000	The maximum number of hop points allowed is 60,000

5.4.6. FREQUENCY/POWER HOP - GET NUMBER OF HOP POINTS

Description

Returns the number of points specified for the current the hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	0	Code for Get Number of Hop Points
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Hop_Points	The number of points in the current hop sequence

Examples

The below returned array indicates that the current hop is configured for 100 points:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	100	The current hop sequence has 100 points

5.4.7. FREQUENCY/POWER HOP - GET SPECIFIC HOP SETTING

Description

Returns the frequency and power setting for a specified point within the hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	1	Code for Get Specific Hop Setting
2	Hop_Point	Index number of the hop point, from 0 to (n - 1) where n equals the number of points specified for the hop
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Freq (1st Byte)	Frequency (Hz) of the hop point split over 5 bytes: Freq = (256^4) * BYTE1 + (256^3) * BYTE2 + (256^2) * BYTE3 + (256) * BYTE4 + BYTE5
2	Freq (2nd Byte)	Frequency (Hz) of the hop point split over 5 bytes
3	Freq (3rd Byte)	Frequency (Hz) of the hop point split over 5 bytes
4	Freq (4th Byte)	Frequency (Hz) of the hop point split over 5 bytes
5	Freq (5th Byte)	Frequency (Hz) of the hop point split over 5 bytes
6	Power (+/-)	Power polarity of the hop point: 0 = Positive (Power = 1 * Power) 1 = Negative (Power = -1 * Power)
7	Power_Mag (1st Byte)	Power magnitude (dBm) of the hop point split over 2 bytes: Power_Mag = (256 * BYTE7 + BYTE8) / 100
8	Power_Mag (2nd Byte)	Power magnitude (dBm) of the hop point split over 2 bytes

Examples

Send the below transmit array to query the frequency/power settings of point 3 in the hop list:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	1	Code for Get Specific Hop Setting
2	3	Query point 3 in the hop list

The below example array is returned:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	0	Frequency (Hz) of the hop point split over 5 bytes: Freq = $(256^4) * 0$ $+ (256^3) * 203$ $+ (256^2) * 100$ $+ (256) * 250$ $+ 45$ = 3,412,392,493 Hz
2	203	Frequency (Hz) of the hop point split over 5 bytes
3	100	Frequency (Hz) of the hop point split over 5 bytes
4	250	Frequency (Hz) of the hop point split over 5 bytes
5	45	Frequency (Hz) of the hop point split over 5 bytes
6	1	Power value is negative
7	10	Power magnitude (dBm) of the hop point split over 2 bytes: Power_Mag = (256 * 10 + 105) / 100 = 26.65 dBm Power = (-1) * Power_Mag = -26.65 dBm
8	105	Power magnitude (dBm) of the hop point split over 2 bytes

The above example indicates that hop point 3 is set for 3,412,392,493 Hz (3412.392493 MHz) at -26.65 dBm.

5.4.8. FREQUENCY/POWER HOP - GET HOP TRIGGER-IN MODE

Description

Returns the trigger-in mode which specifies how the generator will respond to an external trigger during the hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	8	Code for Get Hop Trigger-In Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each hop point Wait for external trigger (Trigger In = logic 1) only at the start of the hop sequence

Examples

The below returned array indicates that the generator is configured to wait for an external trigger input before setting each point in the hop sequence:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	1	Wait for Trigger-In at each hop point

5.4.9. FREQUENCY/POWER HOP - GET HOP TRIGGER-OUT MODE

Description

Returns the trigger-out mode which specifies when the generator will provide an external trigger signal during the hop sequence.

Transmit Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	9	Code for Get Hop Trigger-Out Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each hop point Set trigger output (logic 1) only at the start of the hop sequence

Examples

The below returned array indicates that the generator is configured to produce an external trigger output at the beginning of the hop sequence:

Byte	Data	Description
0	205	Interrupt code for Get Hop Parameter
1	2	Set Trigger-Out at start of hop sequence

5.4.10. FREQUENCY/POWER HOP - SET HOP DIRECTION

Description

Sets the direction that the generator will run through the list of hop values.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	5	Code for Set Hop Direction
2	Hop_Direction	The direction to execute the current hop sequence: 0 = From first to last in the list 1 = From last to first in the list 2 = From first to last in the list, then back from last to first
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

The below transmit array will set the generator to hop through the list from the first value to the last:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	0	Execute the hop list from first to last

5.4.11. FREQUENCY/POWER HOP - SET HOP DWELL TIME

Description

Sets the dwell time to be used by the generator between each frequency/power hop point in the sequence.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	4	Code for Set Hop Dwell Time
2	Hop_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (Hop_Dwell / 256)
3	Hop_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE3 = Hop_Dwell - (BYTE2 * 256)
4 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

The below transmit array sets the signal generator's dwell time to 300 ms:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	4	Code for Set Hop Dwell Time
2	1	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (300 / 256) = 1
3	44	The dwell time in milliseconds split into 2 bytes: BYTE3 = 300 - (1 * 256) = 44

5.4.12. FREQUENCY/POWER HOP - START/STOP HOP SEQUENCE

Description

Starts or stops the hop sequence using the previously defined parameters. The hop sequence will stop automatically if any other command is sent.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	8	Code for Start/Stop Hop Sequence
2	Hop_Mode	Set the hop mode: Hop sequence is disabled Start hop sequence (the sequence will continue until Hop_Mode is set to 0 or any other command is sent)
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

The below transmit array enables the hop sequence (all hop parameters must be set first):

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	8	Code for Start/Stop Hop Sequence
2	1	Start the hop sequence

5.4.13. FREQUENCY/POWER HOP - SET NUMBER OF HOP POINTS

Description

Sets the number of points to be used in the hop sequence.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	0	Code for Set Number of Hop Points
2	Hop_Points	The number of points to configure in the hop sequence
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

The below transmit array configures a hop sequence with 3 points:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	0	Code for Set Number of Hop Points
2	3	Enable 3 points in the hop sequence

Frequency/Power Hop - Set Specific Hop Parameters

Description

Sets the frequency and power for a specified point within the hop sequence.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	1	Code for Set Specific Hop Parameters
2	Hop_Point	Index number of the hop point, from 0 to (n -1) where n equals the maximum allowed number of points
3	Freq	Frequency (Hz) of the hop point split over 5 bytes: BYTE3 = INT (Freq / 256^4)
4	Freq	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER1 = Freq - BYTE3 * 256^4 BYTE4 = INT (REMAINDER1 / 256^3)
5	Freq	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER2 = REMAINDER1 - BYTE4 * 256^3 BYTE5 = INT (REMAINDER2 / 256^2)
6	Freq	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER3 = REMAINDER2 - BYTE5 * 256^2 BYTE6 = INT (REMAINDER3 / 256)
7	Freq	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER4 = REMAINDER3 - BYTE6 * 256 If REMAINDER4 is greater than 0 then: BYTE7 = INT (REMAINDER4) If REMAINDER4 is less than or equal to 0 then: BYTE7 = 0
8	Power (+/-)	Power polarity of the hop point: 0 = Positive (Power = 1 * Power) 1 = Negative (Power = -1 * Power)
9	Power_Mag	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE9 = INT (Power_Mag *100 / 256)
10	Power_Mag	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE10 = Power_Mag * 100 - (BYTE9 * 256)
11 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

To set point 3 in the hop sequence to 3500.25 MHz (3,500,250,000 Hz), -15.5 dBm, send the following transmit array:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	1	Code for Set Specific Hop Parameters
2	3	Set point 3 in the hop sequence
3	0	Frequency (Hz) of the hop point split over 5 bytes: BYTE3 = INT (3500250000 / 256^4) = 0
4	208	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER1 = 3500250000 - 0 * 256^4 = 3500250000 BYTE4 = INT (3500250000 / 256^3) = 208
5	161	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER2 = 3500250000 - 208 * 256^3 = 10589072 BYTE5 = INT (10589072 / 256^2) = 161
6	147	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER3 = 10589072 - 161 * 256^2 = 37776 BYTE6 = INT (37776 / 256) = 147
7	144	Frequency (Hz) of the hop point split over 5 bytes: REMAINDER4 = 37776 - 147 * 256 = 144 REMAINDER4 is greater than 0, therefore: BYTE7 = INT (144) = 144
8	1	Power value is negative
9	6	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE9 = INT (15.5 *100 / 256) = 6
10	14	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE10 = 15.5 * 100 - (6 * 256) = 14
11 - 63	Not used	"Don't care" bytes, can be any value

5.4.14. FREQUENCY/POWER HOP - SET HOP TRIGGER-IN MODE

Description

Sets the trigger-in mode, specifying how the generator will respond to an external trigger during the hop sequence.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	6	Code for Set Hop Trigger-In Mode
2	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each hop point Wait for external trigger (Trigger In = logic 1) only at the start of the hop sequence
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter

Examples

The below transmit array will set the generator to wait for an external trigger input before setting each point in the hop sequence:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	6	Code for Set Hop Trigger-In Mode
2	1	Wait for Trigger-In at each hop point

5.4.15. FREQUENCY/POWER HOP - SET HOP TRIGGER-OUT MODE

Description

Sets the trigger-out mode, specifying when the generator will provide an external trigger signal during the hop sequence.

Transmit Array

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	7	Code for Set Hop Trigger-Out Mode
2	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each hop point Set trigger output (logic 1) only at the start of the hop sequence
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	205	Interrupt code for Set Hop Parameter

Examples

The below transmit array will set the generator to produce an external trigger output at the beginning of the hop sequence:

Byte	Data	Description
0	204	Interrupt code for Set Hop Parameter
1	7	Code for Set Hop Trigger-Out Mode
2	2	Set Trigger-Out at start of hop sequence

5.5. Frequency Sweep Functions

These functions define the frequency sweep capabilities of the generators. The signal generator can be configured to produce an automatic, swept frequency output, using the generator's internal timing systems. The user stores the parameters of the sweep in the generator's memory and can then enable/disable the sweep as required.

Full details of the commands for configuring a frequency sweep sequence are covered in the following sections.

5.5.1. FREQUENCY SWEEP - GET SWEEP DIRECTION

Description

Returns the direction in which the generator will execute the frequency sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	7	Code for Get Sweep Direction
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Sweep_Direction	The direction set for the current frequency sweep: 0 = From start value to stop value 1 = From stop value to start value 2 = From start to stop, then back from stop to start

Examples

The below returned array indicates that the signal generator will sweep bi-directionally, from start to finish, then back to start:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	2	The generator will sweep from start to stop, then back to start

5.5.2. FREQUENCY SWEEP - GET SWEEP DWELL TIME

Description

Returns the dwell time to be used by the generator between each frequency in the sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	4	Code for Get Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Sweep_Dwell	The dwell time in milliseconds split over 2 bytes: Sweep_Dwell = (256 * BYTE1) + BYTE2
2	Sweep_Dwell	The dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's dwell time is set to 300ms:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	1	The dwell time in milliseconds split over 2 bytes: Sweep_Dwell = (256 * 1) + 44 = 300 ms
2	44	The dwell time in milliseconds split over 2 bytes

5.5.3. FREQUENCY SWEEP - GET MAXIMUM SWEEP DWELL TIME

Description

Returns the maximum allowed dwell time in milliseconds for each frequency in the sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	5	Code for Get Maximum Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Sweep_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes: Sweep_MaxDwell = (256 * BYTE1) + BYTE2
2	Sweep_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's maximum dwell time is 10,000ms (10s):

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	39	The maximum dwell time in milliseconds split over 2 bytes: Sweep_MaxDwell = (256 * 39) + 16 = 10,000 ms
2	16	The maximum dwell time in milliseconds split over 2 bytes

5.5.4. FREQUENCY SWEEP - GET MINIMUM SWEEP DWELL TIME

Description

Returns the minimum allowed dwell time in milliseconds for all points in the frequency sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	6	Code for Get Minimum Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Sweep_MinDwell	The minimum dwell time in milliseconds split over 2 bytes: Sweep_MinDwell = (256 * BYTE1) + BYTE2
2	Sweep_MinDwell	The minimum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's minimum dwell time is 20ms:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	0	The minimum dwell time in milliseconds split over 2 bytes: Sweep_MinDwell = (256 * 0) + 20 = 20 ms
2	20	The minimum dwell time in milliseconds split over 2 bytes

5.5.5. FREQUENCY SWEEP - GET SWEEP POWER

Description

Returns the constant power level for the frequency sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	3	Code for Get Sweep Power
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Power (+/-)	Power polarity of the sweep: 0 = Positive (Power = 1 * Power_Mag) 1 = Negative (Power = -1 * Power_Mag)
2	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 * BYTE2 + BYTE3) / 100
3	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes

Examples

The below example returned array indicates that the constant output level set for the current frequency sweep is -12.25 dBm:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	1	Power value is negative
2	4	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 * 4 + 201) / 100 = 12.25 dBm Power = (-1) * Power_Mag = -12.25 dBm
3	201	Power magnitude (dBm) of the sweep, split over 2 bytes

5.5.6. FREQUENCY SWEEP - GET SWEEP START FREQUENCY

Description

Returns the start frequency for the current sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	0	Code for Get Sweep Start Frequency
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Freq	Start frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * BYTE1 + (256^3) * BYTE2 + (256^2) * BYTE3 + (256) * BYTE4 + BYTE5
2	Freq	Start frequency (Hz) of the sweep split over 5 bytes
3	Freq	Start frequency (Hz) of the sweep split over 5 bytes
4	Freq	Start frequency (Hz) of the sweep split over 5 bytes
5	Freq	Start frequency (Hz) of the sweep split over 5 bytes

Examples

The below returned array indicates that the start frequency for the sweep is 1000 MHz:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	0	Start frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * 0 + (256^3) * 59 + (256^2) * 154 + (256) * 202 + 0 = 1,000,000,000 Hz = 1,000 MHz
2	59	Start frequency (Hz) of the sweep split over 5 bytes
3	154	Start frequency (Hz) of the sweep split over 5 bytes
4	202	Start frequency (Hz) of the sweep split over 5 bytes
5	0	Start frequency (Hz) of the sweep split over 5 bytes

5.5.7. FREQUENCY SWEEP - GET SWEEP STOP FREQUENCY

Description

Returns the stop frequency for the current sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	1	Code for Get Sweep Stop Frequency
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Freq	Stop frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * BYTE1 + (256^3) * BYTE2 + (256^2) * BYTE3 + (256) * BYTE4 + BYTE5
2	Freq	Stop frequency (Hz) of the sweep split over 5 bytes
3	Freq	Stop frequency (Hz) of the sweep split over 5 bytes
4	Freq	Stop frequency (Hz) of the sweep split over 5 bytes
5	Freq	Stop frequency (Hz) of the sweep split over 5 bytes

Examples

The below example returned array indicates that the stop frequency for the sweep is 1,999.999999 MHz:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	0	Stop frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * 0 + (256^3) * 119 + (256^2) * 53 + (256) * 147 + 255 = 1,999,999,999 Hz = 1,999.999999 MHz
2	119	Stop frequency (Hz) of the sweep split over 5 bytes
3	53	Stop frequency (Hz) of the sweep split over 5 bytes
4	147	Stop frequency (Hz) of the sweep split over 5 bytes
5	255	Stop frequency (Hz) of the sweep split over 5 bytes

5.5.8. FREQUENCY SWEEP - GET SWEEP STEP SIZE

Description

Returns the frequency step size for the current sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	2	Code for Get Sweep Step Size
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Freq	Step frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * BYTE1 + (256^3) * BYTE2 + (256^2) * BYTE3 + (256) * BYTE4 + BYTE5
2	Freq	Step frequency (Hz) of the sweep split over 5 bytes
3	Freq	Step frequency (Hz) of the sweep split over 5 bytes
4	Freq	Step frequency (Hz) of the sweep split over 5 bytes
5	Freq	Step frequency (Hz) of the sweep split over 5 bytes

Examples

The below example returned array indicates that the step size for the sweep is 100 MHz:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	0	Step frequency (Hz) of the sweep split over 5 bytes: Freq = (256^4) * 0 + (256^3) * 5 + (256^2) * 245 + (256) * 225 + 0 = 100,000,000 Hz = 100 MHz
2	5	Step frequency (Hz) of the sweep split over 5 bytes
3	245	Step frequency (Hz) of the sweep split over 5 bytes
4	225	Step frequency (Hz) of the sweep split over 5 bytes
5	0	Step frequency (Hz) of the sweep split over 5 bytes

5.5.9. FREQUENCY SWEEP - GET SWEEP TRIGGER-IN MODE

Description

Returns the trigger-in mode which specifies how the generator will respond to an external trigger during the frequency sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	8	Code for Get Sweep Trigger-In Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each frequency Wait for external trigger (Trigger In = logic 1) only at the start of the sweep

Examples

The below returned array indicates that the generator is configured to wait for an external trigger input before setting each frequency in the sweep:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	1	Wait for Trigger-In before each frequency point

5.5.10. FREQUENCY SWEEP - GET SWEEP TRIGGER-OUT MODE

Description

Returns the trigger-out mode which specifies when the generator will provide an external trigger signal during the frequency sweep.

Transmit Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	9	Code for Get Sweep Trigger-Out Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each frequency Set trigger output (logic 1) only at the start of the sweep

Examples

The below returned array indicates that the generator is configured to produce an external trigger output at the beginning of the frequency sweep:

Byte	Data	Description
0	201	Interrupt code for Get Frequency Sweep Parameter
1	2	Set Trigger-Out at start of sweep

5.5.11. FREQUENCY SWEEP - SET SWEEP DIRECTION

Description

Sets the direction in which the generator will execute the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	5	Code for Set Sweep Direction
2	Sweep_Direction	The direction to execute the sweep: 0 = From start to stop frequency 1 = From stop to start frequency 2 = From start to stop, then back to start
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The below transmit array will set the generator to sweep backwards, from stop frequency to start frequency:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	1	Execute the sweep backwards

5.5.12. FREQUENCY SWEEP - SET SWEEP DWELL TIME

Description

Sets the dwell time to be used by the generator between setting each frequency in the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	4	Code for Set Sweep Dwell Time
2	Sweep_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (Sweep_Dwell / 256)
3	Sweep_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE3 = Sweep_Dwell - (BYTE2 * 256)
4 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The below transmit array sets the signal generator's dwell time to 300 ms:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	4	Code for Set Sweep Dwell Time
2	1	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (300 / 256) = 1
3	44	The dwell time in milliseconds split into 2 bytes: BYTE3 = 300 - (1 * 256) = 44

5.5.13. FREQUENCY SWEEP - START/STOP SWEEP SEQUENCE

Description

Starts or stops the frequency sweep using the previously defined parameters. The sweep will stop automatically if any other command is sent.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	8	Code for Start/Stop Sweep Sequence
2	Sweep_Mode	Set the hop mode: Sweep sequence is disabled Start sweep (the sequence will continue until Sweep_Mode is set to 0 or any other command is sent)
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The below transmit array enables the frequency sweep (all sweep parameters must be set first):

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	8	Code for Start/Stop Sweep Sequence
2	1	Start the sweep

5.5.14. FREQUENCY SWEEP - SET SWEEP POWER

Description

Sets the constant power level for the frequency sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	3	Code for Set Sweep Power
2	Power (+/-)	Power polarity of the sweep: 0 = Positive (Power = 1 * Power) 1 = Negative (Power = -1 * Power)
3	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes: BYTE3 = INT (Power_Mag *100 / 256)
4	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes: BYTE4 = Power_Mag * 100 - (BYTE3 * 256)
5 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The following transmit array sets a constant output power level of 5.75dBm to be used for the frequency sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	3	Code for Set Sweep Power
2	0	Power value is positive
3	2	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE3 = INT (5.75 *100 / 256) = 2
4	63	Power magnitude (dBm) of the hop point split over 2 bytes: BYTE4 = 5.75 * 100 - (2 * 256) = 63
5 - 63	Not used	"Don't care" bytes, can be any value

5.5.15. FREQUENCY SWEEP - SET SWEEP START FREQUENCY

Description

Sets the start frequency for the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	0	Code for Set Sweep Start Frequency
2	Freq	Start frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (Freq / 256^4)
3	Freq	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = Freq - BYTE2 * 256^4 BYTE3 = INT (REMAINDER1 / 256^3)
4	Freq	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = REMAINDER1 - BYTE3 * 256^3 BYTE4 = INT (REMAINDER2 / 256^2)
5	Freq	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = REMAINDER2 - BYTE4 * 256^2 BYTE5 = INT (REMAINDER3 / 256)
6	Freq	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = REMAINDER3 - BYTE5 * 256 If REMAINDER4 is greater than 0 then: BYTE6 = INT (REMAINDER4) If REMAINDER4 is less than or equal to 0 then: BYTE6 = 0
7 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The following transmit array sets 1000 MHz as the start frequency for the sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	0	Code for Set Sweep Start Frequency
2	0	Start frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (1000000000 / 256^4) = 0
3	59	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = 1000000000 - 0 * 256^4 = 1000000000 BYTE3 = INT (1000000000 / 256^3) = 59
4	154	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = 1000000000 - 59 * 256^3 = 10144256 BYTE4 = INT (10144256 / 256^2) = 154
5	202	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = 10144256 - 154 * 256^2 = 51712 BYTE5 = INT (51712 / 256) = 202
6	0	Start frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = 51712 - 202 * 256 = 0 BYTE6 = INT (0) = 0
7 - 63	Not used	"Don't care" bytes, can be any value

5.5.16. FREQUENCY SWEEP - SET SWEEP STOP FREQUENCY

Description

Sets the stop frequency for the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	1	Code for Set Sweep Stop Frequency
2	Freq	Stop frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (Freq / 256^4)
3	Freq	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = Freq - BYTE2 * 256^4 BYTE3 = INT (REMAINDER1 / 256^3)
4	Freq	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = REMAINDER1 - BYTE3 * 256^3 BYTE4 = INT (REMAINDER2 / 256^2)
5	Freq	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = REMAINDER2 - BYTE4 * 256^2 BYTE5 = INT (REMAINDER3 / 256)
6	Freq	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = REMAINDER3 - BYTE5 * 256 If REMAINDER4 is greater than 0 then: BYTE6 = INT (REMAINDER4) If REMAINDER4 is less than or equal to 0 then: BYTE6 = 0
7 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The following transmit array sets 1,999,999,999 Hz (1999.999999 MHz) as the stop frequency for the sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	1	Code for Set Sweep Stop Frequency
2	0	Stop frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (1999999999 / 256^4) = 0
3	119	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = 1999999999 - 0 * 256^4 = 1999999999 BYTE3 = INT (1999999999 / 256^3) = 119
4	53	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = 1999999999 - 119 * 256^3 = 3511295 BYTE4 = INT (3511295 / 256^2) = 53
5	147	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = 3511295 - 53 * 256^2 = 37887 BYTE5 = INT (37887 / 256) = 147
6	255	Stop frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = 37887 - 147 * 256 = 255 BYTE6 = INT (255) = 255
7 - 63	Not used	"Don't care" bytes, can be any value

5.5.17. FREQUENCY SWEEP - SET SWEEP STEP SIZE

Description

Sets the frequency step size for the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	2	Code for Set Sweep Step Size
2	Freq	Step frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (Freq / 256^4)
3	Freq	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = Freq - BYTE2 * 256^4 BYTE3 = INT (REMAINDER1 / 256^3)
4	Freq	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = REMAINDER1 - BYTE3 * 256^3 BYTE4 = INT (REMAINDER2 / 256^2)
5	Freq	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = REMAINDER2 - BYTE4 * 256^2 BYTE5 = INT (REMAINDER3 / 256)
6	Freq	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = REMAINDER3 - BYTE5 * 256 If REMAINDER4 is greater than 0 then: BYTE6 = INT (REMAINDER4) If REMAINDER4 is less than or equal to 0 then: BYTE6 = 0
7 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The following transmit array sets a step size of 100 MHz for the frequency sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	2	Code for Set Sweep Step Size
2	0	Step frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (100000000 / 256^4) = 0
3	5	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = 100000000 - 0 * 256^4 = 100000000 BYTE3 = INT (100000000 / 256^3) = 5
4	245	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = 100000000- 5 * 256^3 = 16113920 BYTE4 = INT (16113920 / 256^2) = 245
5	225	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = 16113920 - 245 * 256^2 = 57600 BYTE5 = INT (57600 / 256) = 225
6	0	Step frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = 57600 - 225 * 256 = 0 BYTE6 = 0
7 - 63	Not used	"Don't care" bytes, can be any value

5.5.18. FREQUENCY SWEEP - SET SWEEP TRIGGER-IN MODE

Description

Sets the trigger-in mode, specifying how the generator will respond to an external trigger during the frequency sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	6	Code for Set Sweep Trigger-In Mode
2	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each frequency Wait for external trigger (Trigger In = logic 1) only at the start of the sweep
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The below transmit array will set the generator to wait for an external trigger input before setting each frequency in the sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	6	Code for Set Sweep Trigger-In Mode
2	1	Wait for Trigger-In before each frequency is set

5.5.19. FREQUENCY SWEEP - SET SWEEP TRIGGER-OUT MODE

Description

Sets the trigger-out mode, specifying when the generator will provide an external trigger signal during the sweep.

Transmit Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	7	Code for Set Sweep Trigger-Out Mode
2	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each frequency Set trigger output (logic 1) only at the start of the sweep
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter

Examples

The below transmit array will set the generator to produce an external trigger output at the beginning of the frequency sweep:

Byte	Data	Description
0	200	Interrupt code for Set Frequency Sweep Parameter
1	7	Code for Set Sweep Trigger-Out Mode
2	2	Set Trigger-Out at start of sweep

5.6. Power Sweep Functions

These functions define the power sweep capabilities of the generators. The signal generator can be configured to produce an automatic, swept power output, using the generator's internal timing systems. The user stores the parameters of the sweep in the generator's memory and can then enable/disable the sweep as required.

Full details of the commands for configuring a power sweep sequence are covered in the following sections.

5.6.1. POWER SWEEP - GET SWEEP DIRECTION

Description

Returns the direction in which the generator will execute the power sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	7	Code for Get Sweep Direction
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Sweep_Direction	The direction set for the current power sweep: 0 = From start value to stop value 1 = From stop value to start value 2 = From start to stop, then back from stop to start

Examples

The below returned array indicates that the signal generator will sweep bi-directionally, from start to finish, then back to start:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	2	The generator will sweep from start to stop, then back to start

5.6.2. POWER SWEEP - GET SWEEP DWELL TIME

Description

Returns the dwell time to be used by the generator between each power level in the sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	4	Code for Get Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Sweep_Dwell	The dwell time in milliseconds split over 2 bytes: Sweep_Dwell = (256 * BYTE1) + BYTE2
2	Sweep_Dwell	The dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's dwell time is set to 300ms:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	1	The dwell time in milliseconds split over 2 bytes: Sweep_Dwell = (256 * 1) + 44 = 300 ms
2	44	The dwell time in milliseconds split over 2 bytes

5.6.3. POWER SWEEP - GET MAXIMUM SWEEP DWELL TIME

Description

Returns the maximum allowed dwell time in milliseconds for each power level in the sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	5	Code for Get Maximum Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Sweep_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes: Sweep_MaxDwell = (256 * BYTE1) + BYTE2
2	Sweep_MaxDwell	The maximum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's maximum dwell time is 10,000ms (10s):

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	39	The maximum dwell time in milliseconds split over 2 bytes: Sweep_MaxDwell = (256 * 39) + 16 = 10,000 ms
2	16	The maximum dwell time in milliseconds split over 2 bytes

5.6.4. POWER SWEEP - GET MINIMUM SWEEP DWELL TIME

Description

Returns the minimum allowed dwell time in milliseconds for all points in the power sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	6	Code for Get Minimum Sweep Dwell Time
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Sweep_MinDwell	The minimum dwell time in milliseconds split over 2 bytes: Sweep_MinDwell = (256 * BYTE1) + BYTE2
2	Sweep_MinDwell	The minimum dwell time in milliseconds split over 2 bytes

Examples

The below returned array indicates that the signal generator's minimum dwell time is 20ms:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	0	The minimum dwell time in milliseconds split over 2 bytes: Sweep_MinDwell = (256 * 0) + 20 = 20 ms
2	20	The minimum dwell time in milliseconds split over 2 bytes

5.6.5. POWER SWEEP - GET SWEEP FREQUENCY

Description

Returns the constant frequency used for the power sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	3	Code for Get Sweep Frequency
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: Freq = (256^4) * BYTE1 + (256^3) * BYTE2 + (256^2) * BYTE3 + (256) * BYTE4 + BYTE5
2	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes
3	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes
4	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes
5	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes

Examples

The below example returned array indicates that the constant frequency to be used for the current power sweep is 1,000 MHz:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	0	Constant frequency (Hz) of the sweep, split over 5 bytes: Freq = (256^4) * 0 + (256^3) * 59 + (256^2) * 154 + (256) * 202 + 0 = 1,000,000,000 Hz = 1,000 MHz
2	59	Constant frequency (Hz) of the sweep, split over 5 bytes
3	154	Constant frequency (Hz) of the sweep, split over 5 bytes
4	202	Constant frequency (Hz) of the sweep, split over 5 bytes
5	0	Constant frequency (Hz) of the sweep, split over 5 bytes

5.6.6. POWER SWEEP - GET SWEEP START POWER

Description

Returns the start power for the current sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	0	Code for Get Sweep Start Power
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Power (+/-)	Power polarity of the sweep: 0 = Positive (Power = 1 * Power_Mag) 1 = Negative (Power = -1 * Power_Mag)
2	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 * BYTE2 + BYTE3) / 100
3	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes

Examples

The below returned array indicates that the start power for the sweep is -12.25 dBm:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	1	Power value is negative (Power = (-1) * Power_Mag)
2	4	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 * 4 + 201) / 100 = 12.25 dBm Power = (-1) * Power_Mag = -12.25 dBm
3	201	Power magnitude (dBm) of the sweep, split over 2 bytes

5.6.7. POWER SWEEP - GET SWEEP STOP POWER

Description

Returns the stop power for the current sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	1	Code for Get Sweep Stop Power
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Power (+/-)	Power polarity of the sweep: 0 = Positive (Power = 1 * Power_Mag) 1 = Negative (Power = -1 * Power_Mag)
2	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 * BYTE2 + BYTE3) / 100
3	Power_Mag	Power magnitude (dBm) of the sweep, split over 2 bytes

Examples

The below returned array indicates that the stop power for the sweep is +10.00 dBm:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	0	Power value is negative (Power = 1 * Power_Mag)
2	3	Power magnitude (dBm) of the sweep, split over 2 bytes: Power_Mag = (256 *3 + 232) / 100 = 10.00 dBm Power = (+1) * Power_Mag = 10.00 dBm
3	232	Power magnitude (dBm) of the sweep, split over 2 bytes

5.6.8. POWER SWEEP - GET SWEEP POWER STEP SIZE

Description

Returns the power step size for the current sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	2	Code for Get Sweep Power Step Size
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Power_Step	Power step size (dBm), split over 2 bytes: Power_Step = (256 * BYTE1 + BYTE2) / 100
2	Power_Step	Power step size (dBm), split over 2 bytes

Examples

The below returned array indicates that the power step size for the sweep is 0.25 dBm:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	0	Power step size (dBm), split over 2 bytes: Power_Step = (256 * 0 + 25) / 100 = 0.25 dBm
2	25	Power step size (dBm), split over 2 bytes

5.6.9. POWER SWEEP - GET SWEEP TRIGGER-IN MODE

Description

Returns the trigger-in mode which specifies how the generator will respond to an external trigger during the power sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	8	Code for Get Sweep Trigger-In Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each power Wait for external trigger (Trigger In = logic 1) only at the start of the sweep

Examples

The below returned array indicates that the generator is configured to wait for an external trigger input before setting each power in the sweep:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	1	Wait for Trigger-In before each power point

5.6.10. POWER SWEEP - GET SWEEP TRIGGER-OUT MODE

Description

Returns the trigger-out mode which specifies when the generator will provide an external trigger signal during the power sweep.

Transmit Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	9	Code for Get Sweep Trigger-Out Mode
2 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each power Set trigger output (logic 1) only at the start of the sweep

Examples

The below returned array indicates that the generator is configured to produce an external trigger output at the beginning of the power sweep:

Byte	Data	Description
0	203	Interrupt code for Get Power Sweep Parameter
1	2	Set Trigger-Out at start of sweep

5.6.11. POWER SWEEP - SET SWEEP DIRECTION

Description

Sets the direction in which the generator will execute the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	5	Code for Set Sweep Direction
2	Sweep_Direction	The direction to execute the sweep: 0 = From start to stop power 1 = From stop to start power 2 = From start to stop, then back to start
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The below transmit array will set the generator to sweep backwards, from stop power to start power:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	1	Execute the sweep backwards

5.6.12. POWER SWEEP - SET SWEEP DWELL TIME

Description

Sets the dwell time to be used by the generator between setting each power in the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	4	Code for Set Sweep Dwell Time
2	Sweep_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (Sweep_Dwell / 256)
3	Sweep_Dwell	The dwell time in milliseconds split into 2 bytes: BYTE3 = Sweep_Dwell - (BYTE2 * 256)
4 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The below transmit array sets the signal generator's dwell time to 300 ms:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	4	Code for Set Sweep Dwell Time
2	1	The dwell time in milliseconds split into 2 bytes: BYTE2 = INT (300 / 256) = 1
3	44	The dwell time in milliseconds split into 2 bytes: BYTE3 = 300 - (1 * 256) = 44

5.6.13. POWER SWEEP - START/STOP SWEEP SEQUENCE

Description

Starts or stops the power sweep using the previously defined parameters. The sweep will stop automatically if any other command is sent.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	8	Code for Start/Stop Sweep Sequence
2	Sweep_Mode	Set the hop mode: Sweep sequence is disabled Start sweep (the sequence will continue until Sweep_Mode is set to 0 or any other command is sent)
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The below transmit array enables the power sweep (all sweep parameters must be set first):

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	8	Code for Start/Stop Sweep Sequence
2	1	Start the sweep

5.6.14. POWER SWEEP - SET SWEEP FREQUENCY

Description

Sets the constant frequency to be used during the power sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	3	Code for Set Sweep Frequency
2	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (Freq / 256^4)
3	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = Freq - BYTE2 * 256^4 BYTE3 = INT (REMAINDER1 / 256^3)
4	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = REMAINDER1 - BYTE3 * 256^3 BYTE4 = INT (REMAINDER2 / 256^2)
5	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = REMAINDER2 - BYTE4 * 256^2 BYTE5 = INT (REMAINDER3 / 256)
6	Freq	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = REMAINDER3 - BYTE5 * 256 If REMAINDER4 is greater than 0 then: BYTE6 = INT (REMAINDER4) If REMAINDER4 is less than or equal to 0 then: BYTE6 = 0
7 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The following transmit array sets a constant frequency of 1000 MHz to be used for the power sweep:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	3	Code for Set Sweep Frequency
2	0	Constant frequency (Hz) of the sweep, split over 5 bytes: BYTE2 = INT (1000000000 / 256^4) = 0
3	59	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER1 = 1000000000 - 0 * 256^4 = 1000000000 BYTE3 = INT (1000000000 / 256^3) = 59
4	154	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER2 = 1000000000 - 59 * 256^3 = 10144256 BYTE4 = INT (10144256 / 256^2) = 154
5	202	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER3 = 10144256 - 154 * 256^2 = 51712 BYTE5 = INT (51712 / 256) = 202
6	0	Constant frequency (Hz) of the sweep, split over 5 bytes: REMAINDER4 = 51712 - 202 * 256 = 0 BYTE6 = INT (0) = 0
7 - 63	Not used	"Don't care" bytes, can be any value

5.6.15. POWER SWEEP - SET SWEEP START POWER

Description

Sets the start power for the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	0	Code for Set Sweep Start Power
2	Power (+/-)	Power polarity of the start power: 0 = Positive (Power = 1 * Power) 1 = Negative (Power = -1 * Power)
3	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE3 = INT (Power_Mag * 100 / 256)
4	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE4 = Power_Mag * 100 - (BYTE3 * 256)
5 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The following transmit array sets -12.25dBm as the start power for the sweep:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	0	Code for Set Sweep Start Power
2	1	Power value is negative (Power = (-1) * Power_Mag)
3	4	Power magnitude (dBm), split over 2 bytes: BYTE3 = INT (12.25 * 100 / 256) = 4
4	201	Power magnitude (dBm), split over 2 bytes: BYTE4 = 12.25 * 100 - (4 * 256) = 201
5 - 63	Not used	"Don't care" bytes, can be any value

5.6.16. POWER SWEEP - SET SWEEP STOP POWER

Description

Sets the stop power for the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	1	Code for Set Sweep Stop Power
2	Power (+/-)	Power polarity of the stop power: 0 = Positive (Power = 1 * Power) 1 = Negative (Power = -1 * Power)
3	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE3 = INT (Power_Mag * 100 / 256)
4	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE4 = Power_Mag * 100 - (BYTE3 * 256)
5 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The following transmit array sets +10.00dBm as the start power for the sweep:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	1	Code for Set Sweep Stop Power
2	0	Power value is positive (Power = 1 * Power_Mag)
3	3	Power magnitude (dBm), split over 2 bytes: BYTE3 = INT (10 * 100 / 256) = 3
4	232	Power magnitude (dBm), split over 2 bytes: BYTE4 = 10 * 100 - (3 * 256) = 232
5 - 63	Not used	"Don't care" bytes, can be any value

5.6.17. POWER SWEEP - SET SWEEP POWER STEP SIZE

Description

Sets the power step size for the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	2	Code for Set Sweep Power Step Size
2	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE2 = INT (Power_Mag * 100 / 256)
3	Power_Mag	Power magnitude (dBm), split over 2 bytes: BYTE3 = Power_Mag * 100 - (BYTE2 * 256)
4 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The following transmit array sets 0.25dBm as the sweep power step size:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	2	Code for Set Sweep Power Step Size
2	0	Power magnitude (dBm), split over 2 bytes: BYTE2 = INT (0.25 * 100 / 256) = 0
3	25	Power magnitude (dBm), split over 2 bytes: BYTE3 = 0.25 * 100 - (0 * 256) = 25
4 - 63	Not used	"Don't care" bytes, can be any value

5.6.18. POWER SWEEP - SET SWEEP TRIGGER-IN MODE

Description

Sets the trigger-in mode, specifying how the generator will respond to an external trigger during the power sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	6	Code for Set Sweep Trigger-In Mode
2	Trigger_Mode	The trigger-in mode: Ignore trigger input Wait for external trigger (Trigger In = logic 1) before setting each power Wait for external trigger (Trigger In = logic 1) only at the start of the sweep
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The below transmit array will set the generator to wait for an external trigger input before setting each power in the sweep:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	6	Code for Set Sweep Trigger-In Mode
2	1	Wait for Trigger-In before each power is set

5.6.19. POWER SWEEP - SET SWEEP TRIGGER-OUT MODE

Description

Sets the trigger-out mode, specifying when the generator will provide an external trigger signal during the sweep.

Transmit Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	7	Code for Set Sweep Trigger-Out Mode
2	Trigger_Mode	The trigger-out mode: Disable trigger output Set trigger output (logic 1) on setting each power Set trigger output (logic 1) only at the start of the sweep
3 - 63	Not used	"Don't care" bytes, can be any value

Returned Array

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter

Examples

The below transmit array will set the generator to produce an external trigger output at the beginning of the frequency sweep:

Byte	Data	Description
0	202	Interrupt code for Set Power Sweep Parameter
1	7	Code for Set Sweep Trigger-Out Mode
2	2	Set Trigger-Out at start of sweep

5.7. Ethernet Configuration Functions

These commands and queries apply to Mini-Circuits Ethernet enabled series of signal generators for configuring the Ethernet parameters.

5.7.1. SET STATIC IP ADDRESS

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.2. SET STATIC SUBNET MASK

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.3. SET STATIC NETWORK GATEWAY

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.4. SET HTTP PORT

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.5. SET TELNET PORT

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.6. USE PASSWORD

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.7. SET PASSWORD

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.8. USE DHCP

Description

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

Transmit Array

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	207	Interrupt code for Use DHCP
2	DHCP_Mode	0 = DCHP disabled (static IP settings in use) 1 = DHCP enabled (default - dynamic IP in use)
3 - 63	Not significant	Any value

Returned Array

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

Examples

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

Byte	Data	Description
0	250	Interrupt code for Set Ethernet Configuration
1	207	Interrupt code for Use DHCP
2	1	Enable DHCP

5.7.9. GET STATIC IP ADDRESS

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.10. GET STATIC SUBNET MASK

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.11. GET STATIC NETWORK GATEWAY

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.12. GET HTTP PORT

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.13. GET TELNET PORT

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.14. GET PASSWORD STATUS

Description

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

Transmit Array

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

Returned Array

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

Examples

The following returned array indicates that password protection is enabled:

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

5.7.15. GET PASSWORD

Description

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

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.16. GET DHCP STATUS

Description

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

Transmit Array

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

Returned Array

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

Examples

The following returned array indicates that DHCP is enabled:

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

5.7.17. GET DYNAMIC ETHERNET CONFIGURATION

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

Transmit Array

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

Returned Array

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

Examples

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

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

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

5.7.18. GET MAC ADDRESS

Description

Returns the MAC address of the signal generator.

Transmit Array

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

Returned Array

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

Examples

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

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

5.7.19. RESET ETHERNET CONFIGURATION

Description

Forces the signal generator to reset and adopt the latest Ethernet configuration. Must be sent after any changes are made to the configuration.

Transmit Array

Byte	Data	Description
0	111	Reset Ethernet configuration sequence
1	101	Reset Ethernet configuration sequence
2	102	Reset Ethernet configuration sequence
3	103	Reset Ethernet configuration sequence
4 - 63	Not significant	Any value

Returned Array

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

6. 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.

6.1. 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. Refer to Link-Local / Auto IP Address for details of supported models
2. 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)
3. This IP can be used to connect to the device and update the Ethernet configuration as needed

6.2. 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.

7. Contact

Mini-Circuits

13 Neptune Avenue

Brooklyn, NY 11235

Phone: +1-718-934-4500

Email: sales@minicircuits.com

Web: www.minicircuits.com