D4000

40 GHz RF Downconverter

Programmer's Guide - Preliminary

Version 0.3

May 2021
Document no. 75-0043-210519

Important notice

Trademarks

ThinkRF Corp

390 March Road
Kanata, ON K2K 0G7
(613) 369-5104

HARDWARE WARRANTY AND LIMITATION OF LIABILITY

Read this warranty carefully before you use the product.

D4000 40 GHz RF Downconverters are warranted for workmanship and materials for a period of one (1) year from the date of shipment as identified by the Customer's packing slip or carrier waybill. ThinkRF reserves the right to void the warranty on any equipment that has been altered or damaged due to Customer negligence, unauthorized repair, misuse of equipment, evidence of physical or environmental damage, transportation abuse or removal of any ThinkRF identification labels or serial numbers.

It will remain the responsibility of the Customer, having obtained a Return Material Authorization (RMA) and shipping instructions from ThinkRF, to return, at the Customer's expense, the defective unit to ThinkRF's repair facilities. ThinkRF will incur shipping charges for the return of warranty repaired equipment. The RMA number can be secured by calling ThinkRF Customer Service and Support (1-613-369-5104). If the product does not fall within ThinkRF's warranty period or the product is found to be functioning as designed, then under the terms of ThinkRF's warranty policy, all costs of repairs and shipping will be charged directly to the Customer. ThinkRF will warrant repaired units for a period of 90 days from date of shipment from ThinkRF to the Customer. If the remaining period on the original hardware warranty is greater than 30 days, then ThinkRF will honor this remaining warranty period.

THINKRF EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, WARRANTIES, CONDITIONS OR REPRESENTATIONS OF WORKMANSHIP, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, DURABILITY, OR THAT THE OPERATION OF THE HARDWARE OR LICENSED SOFTWARE WILL BE ERROR FREE. IN NO EVENT WILL THINKRF BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

USE OF PRODUCTS IN HIGH RISK ACTIVITIES

THINKRF PRODUCTS ARE INTENDED FOR STANDARD INDOOR COMMERCIAL USE. WITHOUT THE APPROPRIATE NETWORK DESIGN ENGINEERING, THEY MUST NOT BE USED FOR ANY "HIGH RISK ACTIVITY", as described in this paragraph. Customer acknowledges and agrees that the products supplied hereunder are not fault-tolerant and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail safe performance including but not limited to the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of products could lead directly to death, personal injury, or severe physical or environmental damage, all of which are examples of "High Risk Activity". THINKRF AND ITS SUPPLIERS EXPRESSLY DISCLAIM ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR HIGH RISK ACTIVITIES.

Table of Contents

Abbreviations

ADC	Analog-to-Digital Converter
API	Application Programming Interface
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate Frequency
LAN	Local Area Network
MSB	Most Significant Byte
PLL	Phase-Locked Loop
RF	Radio Frequency
RFE	Receiver Front-End
Downconverter	40 GHz RF Downconverter
SCPI	Standard Commands for Programmable Instruments
TCP/IP	Transmission Control Protocol/Internet Protocol

List of Figures

List of Tables

Preface

This preface describes the audience for, the organization of, and conventions used in this document. It also identifies related documentation and explains how to access electronic documentation.

Audience

This document is written for software developers wishing to develop and/or maintain a software interface to the D4000 and who have a basic understanding, familiarity and experience with network test and measurement equipment.

Conventions

This section describes the conventions used in this document.

Grayed-out Font
Indicates a command or a feature is not yet available in the current release.

`Courier Font`
Illustrates this is an example for a command or a concept.

Light Blue Font
Contains hyperlink to the referenced source that can be clicked on.

Normal Bold Font
When used within a sentence or a paragraph, it emphasizes an idea to be paid attention to particularly.

Red Font
Conveys special information of that section.

Note: This symbol means **take note**. Notes contain helpful suggestions or references to additional information and material.

Caution: This symbol means **be careful**. In this situation, you might do something that could result in equipment damage or loss of data.

Warning: This symbol means **danger**. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with the standard practices for preventing accidents.

Obtaining Documentation and Releases

You can access the most current ThinkRF documentation and the latest release bundles at http://www.thinkrf.com/resources.

Document Feedback

Please send your comments about this document or our other documentation to support@thinkrf.com.

Thank you, we appreciate your comments.

Obtaining Technical Assistance

The ThinkRF Support website provides online documents for resolving technical issues with ThinkRF products at http://www.thinkrf.com/resources.

For all customers who hold a valid end-user license, ThinkRF provides technical assistance 9 AM to 5 PM Eastern Time, Monday to Friday. Contact us at support@thinkrf.com or by calling **+1.613.369.5104**.

Before contacting Support, please have the following information available:

- D4000's serial number which is located on the identification label on the D4000's underside.
- The product version.
- The firmware version running on the D4000 (using *IDN? command).
- Versions of any ThinkRF software you are using.
- The operating system and version you are using.

D4000 Functional Overview

This section overviews the D4000's functionality and protocols used, and summarizes the SCPI command sets for controlling the individual functions.

Note: This is a living and evolving document. We welcome your feedback.

The features and functionality described in this section **may** exist in the current product firmware release or are scheduled for a future product firmware release (grayed out commands and/or text). Please refer to Appendix E: SCPI Commands Quick Reference for the complete list of commands and the availability information. No hardware upgrade is required at each feature release (unless specified though unlikely).

System Overview

D4000 40 GHz Downconverter is used to convert RF signals in the range of 24-40 GHz down to an intermediate frequency (IF) of 1.536 GHz. This is designed to extend the functionality of existing spectrum analyzers that operate to a maximum frequency of 4 GHz to measure and analyze 5G signals in the range of 24-40 GHz band. Figure 1 shows a simplify interconnect diagram with a spectrum analyzer or similar equipment. Refer to the D4000 User Guide for the setup instructions.

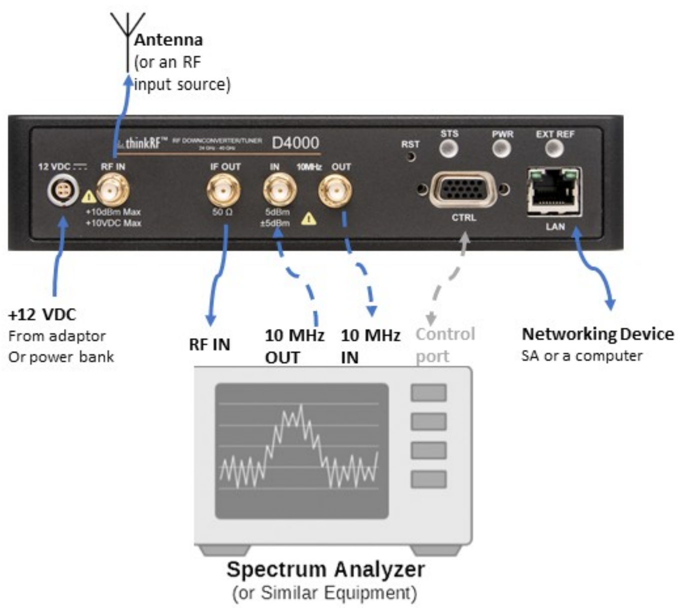


Figure 1: D4000 Interconnect Diagram with A Spectrum Analyzer

ThinkRF's products conform with standardized protocols for interoperability. The D4000 supports the Standard Commands for Programmable Instruments (SCPI) protocol for controlling and obtaining status.

Refer to Appendix A for how to connect to a Downconverter.

The D4000 provides system level control and status commands as defined in Table 1.

Table 1: System Level Control/Status Commands

SCPI Command	Description
:SYSTem	*Page 16*
:COMMunicate	
:LAN<commands>	Subset of commands for configuring/querying Downconverter's LAN settings
:ERRor?	Returns the error code and messages from the SCPI error/event queue
:OPTions?	Returns comma separated 3-digit values to represent the hardware option(s) or features available with a particular Downconverter model
:VERSion?	Returns the SCPI version number that the instrument complies with
:STATus	*Page 20*
:OPERation	
[:EVENt]?	Returns the standard Operation Status Register (OSR) and clears the register
:CONDition?	Returns the standard Operation Condition Register (OCR)
:ENABle[?]	Sets or queries the Operation Status Enable Register (OSE)
:PRESET	Presets the D4000 (similar to *RST)
:QUEStionable	
[:EVENt]?	Returns the Questionable Status Register (QSR) and clears the register
:CONDition?	Returns the Questionable Condition Register (QCR)
:ENABle[?]	Sets or queries the Questionable Status Enable Register (QSE)
:TEMPerature?	Returns the D4000's internal ambient temperature

See SCPI Command Set section (page 13 onward) for further details on the commands.

Caution pertaining to multi-user: The current firmware version of the D4000 allows multiple applications to connect to the unit simultaneously but it does not support independent sessions. Therefore, the actions of one user may over-write those of another. This could potentially damage the unit for instance if the front-end's gain were incorrectly set. If multiple applications are connecting to the unit, it is advised that only one of those is controlling the unit at any time.

RF Receiver Front-End

The receive front-end (RFE) has been largely defined through the hardware specifications. The primary commands have to do with setting the center frequency of the Downconverter and switching in the front-end gain for improved noise figure, ThinkRF provides the user access to other blocks within the radio receiver. The command set is defined in Table 2.

Table 2: RF Front-End Control/Status Commands

SCPI Command	Description
:INPut	*Page 27*
:COUPling?	Queries the input coupling method
:FILTer:PRESelect[?]	Select or query the input preselect filter
:GAIN[?]	Set or query an input gain stage to be on or off.
[:SENSe]	*Page 28*
:ATTenuator[?]	Queries or sets the input attenuator
:AUTO[?]	Enables or disables automatic selection of the input attenuator
:DCONVerter	
:BAND?	Queries the band edge information
:BAND:COUNt?	Queries the number of bands
:FREQuency:CENTer[?]	Sets the center frequency of the D4000 RF input
:LO:COUNt?	Queries the number of LOs
:LO<1|2>	
:FREQuency[?]	Queries or sets the LO frequencies (LO1, LO2)
:LOCK?	Queries the lock status for an LO (LO1, LO2)
:RF:LOCK?	Queries the lock status of the downconverter
:SOURce	
:REFerence[?]	Queries or selects the 10 MHz reference clock source
:AUTO[?]	Enables or disables the automatic selection of the
:FREQuency?	Queries the reference frequency
:OUTPut:ENABle[?]	Enables or disables the reference clock output
:OUTPut	*Page 32*
:ATTenuator[?]	Queries or sets the IF output attenuation in dB
:IF	
:ATTenuator[?]	Queries or sets the IF attenuation in dB
:FREQuency?	Queries the output IF frequency

See SCPI Command Set section (page 13 onward) for further details on each set of commands.

SCPI Command Set

The D4000 supports the Standard Commands for Programmable Instruments (SCPI) standard version 1999.0 as described in the following sections. SCPI lends itself to a command line interface and scripting, is supported by the major instrument vendors and provides a high level of familiarity for instrument users.

Note: The D4000 receives SCPI commands and sends query responses using the network interface of raw socket on port 5025. See Appendix A: Booting up and Connecting to the D4000 for more details.

SCPI Language Overview

In the early 1990s, a group of instrument manufacturers developed Standard Commands for Programmable Instrumentation (SCPI) for controlling programmable instruments via a communication link, such as RS232, USB, LAN, etc. SCPI specifies the command structure and syntax using ASCII characters to provide some basic standardization and consistency to the control commands. SCPI commands, hence, lend themselves to communications with equipments via command line interface, scripting and/or programming languages such as C/C++, MATLAB®, Python, etc.

The SCPI language is based on a hierarchical or tree structure as illustrated in Figure 2 an example command set. The top level of the tree is the root node, which is followed by one or more lower-level nodes.

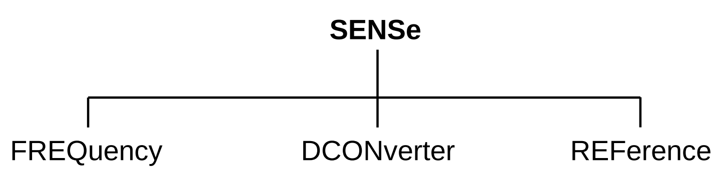


Figure 2: SCPI Language Hierarchical or Tree Structure Example

The traditional model of a typical SCPI instrument involves either a measurement function where an external input is digitized and processed, or a source function where a signal is generated and sent to an external output. The D4000 does not fit this traditional model in that it performs no intermediary digital processing. However, it performs tasks such as frequency conversion purely in the analog domain. Figure 3 shows the D4000 simplified instrument model.

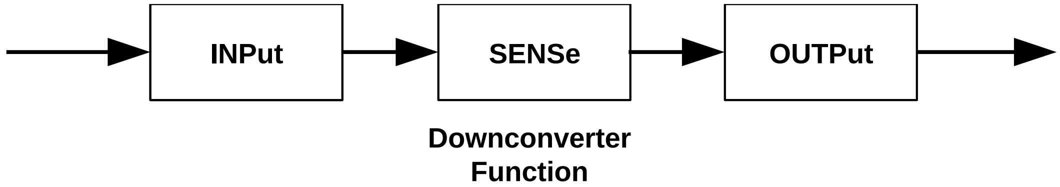


Figure 3: SCPI Downconverter Instrument Model

Refer to the Appendix B: SCPI Command Syntax section for the general SCPI command syntax format and usage details.

IEEE Mandated SCPI Commands

These commands control and query the communication event/error and status registers as defined in the Appendix C: SCPI Status and Event Registers section. They are mandated by the IEEE.

Caution: The mandated IEEE SCPI commands are not affected by *RST command.

*CLS

The Clear Status (CLS) command clears all the event status registers in the device status-reporting mechanism and the error/event queue. This also results in the corresponding summary bits in the Status Byte (STB) to be cleared.

Syntax	*CLS
Parameter/Response	None

*ESE/*ESE?

*ESE command sets bits in the ESE register. The decimal integer value entered is the binary equivalent of the desired 8-bit mask. Bits set in the ESE enables the corresponding bit in the ESR to assert the Standard ESR summary bit in the STB (bit 5).

*ESE? query returns the decimal sum of the bits in the ESE register.

See Figure 4 for the ESE/ESR register bit mapping.

Syntax	*ESE <integer> *ESE?
Parameter/Response	0 - 255
I/O Data Type	<integer>

*ESR?

Query the standard Event Status Register (ESR), which returns the decimal sum of the bits in the ESR. The ESR will only appear set if and only if its event has occurred and the corresponding bit in the ESE is also enabled.

See Figure 4 for the ESR register bits mapping.

Caution: This is a destructive read. Once queried, the register is cleared.

Syntax	*ESR?
Query Response	<integer>
Description	Refer to the Appendix C: SCPI Status and Event Registers section for the ESR register bit definition

*IDN?

Returns the D4000's identification information string.

Note: The model string returned will not include the options. To find out which options a model has, use :SYSTem:OPTions? command.

Syntax	*IDN?
Query Response	"<Manufacturer>,<Model>,<Serial number>,<Firmware version>"
Output Data Type	<string>

*OPC/*OPC?

The *OPC/*OPC? commands allow synchronization between the controller and the D4000.

*OPC (Operation Complete) sets bit 0 in the ESR to 1 when all commands received before *OPC or *OPC? have been completed.

*OPC? returns the ASCII character 1 in the Standard Event register indicating completion of all pending operations. The query also stops any new commands from being processed until the current processing is complete.

Syntax	*OPC *OPC?
Parameter	None
Query Response	1

*RST

Resets the D4000 to its default settings. *RST does not affect the registers or queues associated with the IEEE mandated commands. Each non-IEEE mandated command description in this reference shows the *RST value when affected.

Syntax	*RST
Parameter/Response	None

*SRE/*SRE?

The *SRE (Service Request Enable) command enables bits in the SRE register. The decimal integer value entered is the binary equivalent of the desired 8-bit mask.

*SRE? query returns the decimal sum of the enabled bits in the SRE register. The decimal sum is the binary equivalent of the 8-bit mask.

See Figure 4 for the SRE/STB register bit mapping.

Syntax	*SRE <integer> *SRE?
Parameter/Response	<integer>

*STB?

*STB? (Status Byte) query returns the decimal sum of the bits set in the STB register without erasing its content. Each bit corresponds to the underlying Status Data Structure.

See Figure 4 for the ESE/ESR register bits mapping and the Status Byte Register (SBR) section of the Appendix C for the bit definitions.

Syntax	*STB?
Query Response	<integer>

*TST?

*TST? (self-test) query initiates the device's internal self-test and returns one of the following results:

- 0 - all tests passed.
- 1 - one or more tests failed.

Syntax	*TST?
Query Response	0 | 1
Output Data Type	<integer>

*WAI

*WAI (Wait-to-Continue) command suspends the execution of any further commands or queries until all operations for pending commands are completed.

Syntax	*WAI
Parameter/Response	None

SYSTem Commands

These commands control and query the communication event and status registers as defined in the Appendix C: SCPI Status and Event Registers. They are the minimal :SYSTem sets required in all SCPI instruments.

:SYSTem:COMMunicate:LAN:APPLy

This command will save the changes to the LAN settings to the unit's internal memory. ***The new settings will take effect only after the D4000 has been rebooted or power cycle.*** Once the LAN settings are saved, they are not affected by :STATus:PRESET or *RST.

Caution: When changing from DHCP to STATIC mode, this command should to be sent only when all the required LAN settings are set using the appropriate subsequent :SYSTem:COMMunicate:LAN commands.

Syntax	:SYSTem:COMMunicate:LAN:APPLy
Parameter/Response	None
*RST State	N/A
Examples	`:SYST:COMM:LAN:APPLY`

:SYSTem:COMMunicate:LAN:CONFigure

The set command stores the new LAN configuration type in the Downconverter temporary. The new configuration does not take effect until :SYSTem:COMMunicate:LAN:APPLy is sent (please refer to the Caution note of the :APPLy command). Once the option is applied, it is not affected :STATus:PRESET or *RST.

The query command will return the option set or that of the actual current configuration if one is not set. The CURRENT query will return what is currently and actually used by the Downconverter's LAN interface.

Notes:
- The default factory configuration is STATIC mode with IP 192.168.1.2
- *RST command cannot be used to set the box to its manufacturing default state of STATIC mode. To set the box back to STATIC mode from a working DHCP/auto mode, use this command or perform a factory reset.

Syntax	SYSTem:COMMunicate:LAN:CONFigure {DHCP | STATIC} SYSTem:COMMunicate:LAN:CONFigure? [CURRENT]
Parameter	Set: {DHCP | STATIC} Query: [CURRENT]
Query Response	{DHCP | STATIC}
I/O Data Type	<character>
*RST State	N/A
Examples	`:SYST:COMM:LAN:CONF DHCP` `:SYST:COMM:LAN:CONF? CURRENT`

:SYSTem:COMMunicate:LAN:GATEway

The set command stores the new LAN gateway in the Downconverter temporary. The new gateway does not take effect until :SYSTem:COMMunicate:LAN:APPLy is sent (please refer to the Caution note of the :APPLy command). Once the setting is applied, it is not affected by :STATus:PRESET or *RST.

The query will return the gateway address set or that of the actual current configuration if one is not issued. The CURRENT query will return what is currently and actually used by the Downconverter's LAN interface.

Syntax	SYSTem:COMMunicate:LAN:GATEway <IPv4 address> SYSTem:COMMunicate:LAN:GATEway? [CURRENT]
Parameter	Set: D.D.D.D where D = 0 – 255 Query: [CURRENT]
Query Response	D.D.D.D
I/O Data Type	<string>
*RST State	N/A

Examples	`SYST:COMM:LAN:GATEWAY 102.101.0.13`
	`SYSTEM:COMMUNICATE:LAN:GATEWAY?`
	`SYST:COMM:LAN:GATE? CURRENT`

:SYSTem:COMMunicate:LAN:IP

The set command stores the new LAN IP in the Downconverter temporary. The new IP does not take effect until :SYSTem:COMMunicate:LAN:APPLy is sent (please refer to the Caution note of the :APPLy command). Once the setting is applied, it is not affected by :STATus:PRESET or *RST.

The query command will return the IP address set or that of the actual current configuration if one is not issued. The CURRENT query will return what is currently and actually used by the Downconverter's LAN interface.

Note: The default factory reset STATIC IP is 192.168.1.2.

Syntax	SYSTem:COMMunicate:LAN:IP <IPv4 address> SYSTem:COMMunicate:LAN:IP? [CURRENT]
Parameter	Set: D.D.D.D where D = 0 – 255 Query: [CURRENT]
Query Response	D.D.D.D
I/O Data Type	<string>
*RST State	N/A

Examples	`SYST:COMM:LAN:IP 101.125.1.16`
	`SYSTEM:COMM:LAN:IP?`
	`SYST:COMM:LAN:IP? CURRENT`

:SYSTem:COMMunicate:LAN:NETMask

The set command stores the new LAN netmask address in the Downconverter temporary. The new gateway does not take effect until :SYSTem:COMMunicate:LAN:APPLy is sent (please refer to the Caution note of the :APPLy command). Once the setting is applied, it is not affected by :STATus:PRESET or *RST.

The query command will return the netmask address set or that of the actual current configuration if one is not issued. The CURRENT query will return what is currently and actually used by the Downconverter's LAN interface.

Syntax	SYSTem:COMMunicate:LAN:NETMask <address> SYSTem:COMMunicate:LAN:NETMask? [CURRENT]
Parameter	Set: D.D.D.D where D = 0 – 255 Query: [CURRENT]
Query Response	D.D.D.D
I/O Data Type	<string>
*RST State	N/A

Examples	`SYST:COMM:LAN:NETMASK 255.255.255.0`
	`SYSTEM:COMMUNICATE:LAN:NETM?`
	`SYST:COMM:LAN:NETM? CURRENT`

:SYSTem:ERRor[:NEXT]?

This query command returns the oldest uncleared error code and message from the SCPI error/event queue. When there are no error messages, the query returns 0,"No error". *RST does not affect the error queue.

Note: It is recommended to do this query command after each non-query command is sent to ensure that the non-query command is executed without error. Since each error message is queued into a buffer, if multiple commands have been sent follow by only one :SYSTem:ERRor[:NEXT]? command, it would be unclear which command has resulted in which error.

Syntax	:SYSTem:ERRor[:NEXT]?
Query Response	<error code>,<description>
Output Data Type	<integer>,<string>
Description	Refer to the Appendix C: SCPI Status and Event Registers section
Example	`:SYST:ERR?`

:SYSTem:ERRor:ALL?

This query command returns all the uncleared error codes and messages from the SCPI error/event queue. If there are no error messages, the query returns 0,"No error".

Syntax	:SYSTem:ERRor:ALL?
Query Response	<error code>,<description>{,<error code>,<description>}
Output Data Type	<integer>,<string>{,<integer>,<string>}
Description	Refer to the Appendix D: SCPI Error Codes Used section
Example	`:SYST:ERR:ALL?`

:SYSTem:OPTions?

This command queries the hardware option(s) or features that a particular Downconverter model supported. The response string contains comma separated 3-digit values to represent the options. See Table 3 for the translated list.

Syntax	:SYSTem:OPTions?
Query Response	<xxx>{,<xxx>}
Output Data Type	Comma separated 3-digit value (ex: 000, 001, 002)
*RST State	None
Example	`:SYST:OPT?`

Table 3: Downconverter Option Codes and the Corresponding Description

Option Code	Description	Related SCPI Command
000	No Special Option	

:SYSTem:VERSion?

This query returns the SCPI version number that the instrument software complies with.

Syntax	:SYSTem:VERSion?
Query Response	<NR2>
Output Data Type	<string> (decimal number YYYY.V)
Example	:SYST:VERS?

STATus Commands

The STATus commands control the SCPI-defined status-reporting structures as illustrated in Figure 4. These structures aggregate a set of device conditions that can be used to assert a Service Request (SRQ) to a controller. Each condition can be selectively enabled as required by the controller application.

Status Reporting Structures

SCPI defines the QUEStionable, OPERation, Instrument SUMmary and INSTrument registers in addition to those in IEEE 488.2. These registers conform to the IEEE 488.2 specification and each may be comprised of a condition register, an event (status) register, an enable register, and negative and positive transition filters.

SCPI also defines an IEEE 488.2 queue for status. The queue provides a human readable record of instrument events. The application programmer may individually enable events into the queue. :STATus:PRESET enables errors and disables all other events. If the summary of the queue is reported, it shall be reported in bit 2 of the status byte register. A subset of error/event numbers is defined by SCPI.

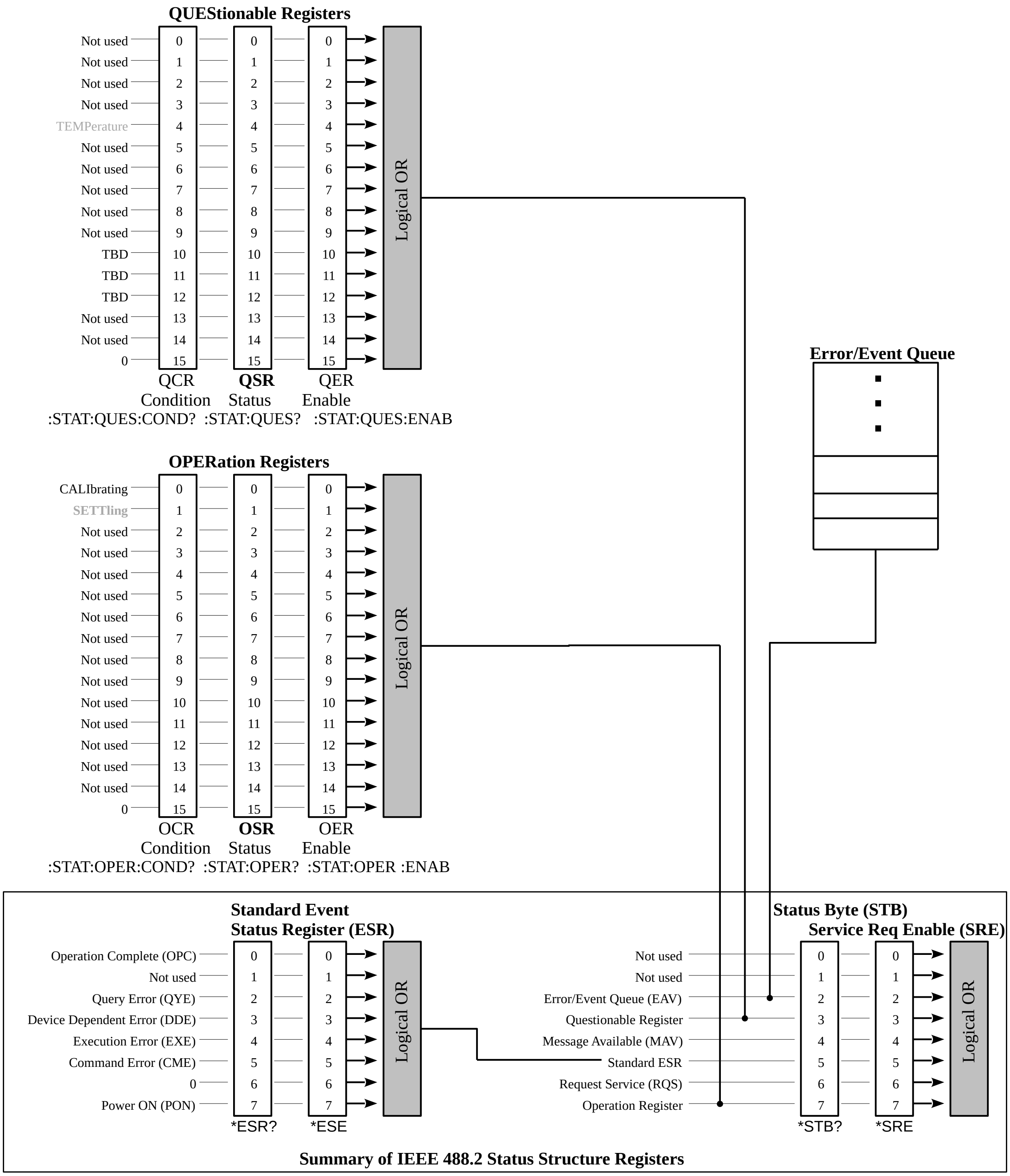


Figure 4: Status Reporting Structure with Status & Enable Registers

Bits 0-5 and 7 in the Status Byte (STB) serve as summary bits for underlying Status Data Structures (SDS). Bit 6 is the Request Service flag, which is always 0 when the STB is read.

An SDS is defined as either a Register Model or a Queue Model. The Queue Model applies to the Error/Event Queue. The summary bit is set to 1 whenever the queue is not empty, indicating that the device has messages to retrieve from the queue.

The SDS Register Model (see Figure 5) applies to both OPERation and QUEStionable registers. The summary bit is set to 1 when an enabled condition is asserted. The controller can then query the corresponding event status register to determine which events occurred. Each Register Model consists of a set of 16-bit registers that capture device conditions and configure behavior. Each bit position corresponds to a condition. For IEEE-488 legacy reasons, bit 15 is unused in all registers and is always zero.

Register Name	Description (per bit)
Condition Register	Reflects the current state of the underlying condition.
Enable Register	Determines if the condition affects the summary bit.
Event Status Register	Latches a condition event based on the transition register configuration. Cleared when read.
Negative Transition Register	A high-to-low condition transition sets the corresponding Status Register bit.
Positive Transition Register	A low-to-high condition transition sets the corresponding Status Register bit.

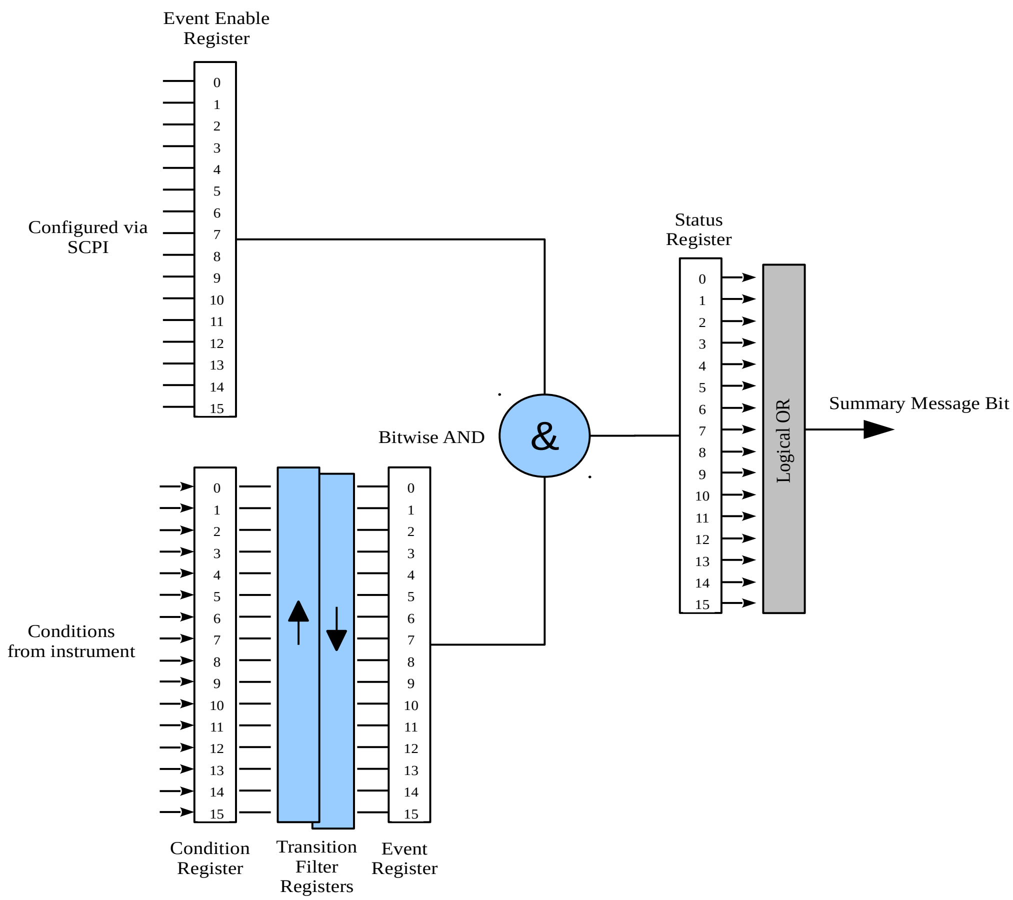


Figure 5: SDS Register Model

The Operation Status Register (OSR) contains conditions which are part of the device's normal operation. These conditions can be used for synchronizing between a controller and the D4000.

The Questionable Status Register (QSR) contains conditions which give an indication of errors or quality issues (e.g. out-of-calibration, out-of-lock, over-temperature, etc.). These conditions can be used to signal the controller of exceptional events that may require corrective action.

The controller determines the source of the service request by querying the Status Byte (*STB?) and then querying the underlying Status Data Structures (SDS) for each summary bit that is set using the appropriate command.

Refer to the following commands and Appendix C: SCPI Status and Event Registers for the bits supported by D4000.

:STATus:OPERation[:EVENt]?

This command queries the standard Operation Status Register (OSR) for any event. The query returns the decimal sum of the bits set in the OSR. Refer to Appendix C: SCPI Status and Event Registers).

The OSR records changes in conditions assigned in the OCR based on the configuration of the corresponding positive and negative transition registers.

Caution: This query clears all bits in the register to 0 as well as bit 7 (Operation Register summary) in the STB.

See Figure 4 for the Operation Status register bit mapping.

Syntax	:STATus:OPERation[:EVENt]?
Query Response	0 – 32767 (2^{15}-1)
Output Values	<integer>
*RST State	None
Example	`:STAT:OPER?`

:STATus:OPERation:CONDition?

This command queries the standard Operation Condition Register (OCR) for any questionable event. The query returns the decimal sum of the bits set in the OCR. The OCR reflects the current state of each condition and remains unchanged when read.

See Figure 4 for the Operation Condition register bit mapping.
58%

Syntax	:STATus:OPERation:CONDition?
Query Response	0 – 32767 (2^{15}-1)
Output Values	<integer>
*RST State	None
Example	`:STAT:OPER:COND?`

:STATus:OPERation:ENABle

This command enables or queries bits in the Operation Enable register (OER). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the OER register and the corresponding OSR register bit is also set, the Standard Operation Status Summary bit (bit 7) in the STB register is set. See Figure 4.

Syntax	:STATus:OPERation:ENABle <integer> :STATus:OPERation:ENABle?
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:OPER:ENAB 256` `:STAT:OPER:ENAB?`

:STATus:OPERation:NTRansition

This command enables bits in the Operation Negative Transition Register (ONTR). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the ONTR, a high-to-low transition in the OCR bit will set the corresponding OSR bit. See Figure 4.

Syntax	:STATus:OPERation:NTRansition <integer>
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:OPER:NTR 256`

:STATus:OPERation:PTRansition

This command enables bits in the Operation Positive Transition Register (OPTR). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the OPTR, a low-to-high transition in the OCR bit will set the corresponding OSR bit. See Figure 4.

Syntax	:STATus:OPERation:PTRansition <integer>
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:OPER:PTR 256`

:STATus:PRESET

This command presets the D4000 (similar to *RST), and OSE and QSE to zero.

Syntax	:STATus:PRESET
Parameter/Response	None

:STATus:QUEStionable[:EVENt]?

This command queries the standard Questionable Status Register (QSR) for any event. The query returns the decimal sum of the bits set in the QSR. The decimal sum is the binary equivalent of the 16-bit mask. Bit 15 is unused. Refer to Appendix C: SCPI Status and Event Registers.

The QSR records changes in conditions assigned in the QCR based on the configuration of the corresponding positive and negative transition registers.

Caution: This query clears all bits in the register to 0 as well as bit 3 (Questionable Register summary) in the STB.

See Figure 4 for the Questionable Status register bits mapping.

Syntax	:STATus:QUEStionable[:EVENt]?
Query Response	0 – 32767 (2^{15}-1)
Output Data Type	<integer>
*RST State	None
Example	`:STAT:QUES?`

:STATus:QUEStionable:CONDition?

This command queries the standard Questionable Condition Register (QCR) for any questionable event. The query returns the decimal sum of the bits set in the QCR. The decimal sum is the binary equivalent of the 16-bit mask. Bit 15 is unused. Refer to Appendix C: SCPI Status and Event Registers. The content of the QCR remains unchanged after it is read.

The data in this register is continuously updated to reflect the most current conditions.

See Figure 4 for the Questionable Condition register bits mapping.

Syntax	:STATus:QUEStionable:CONDition?
Query Response	0 – 32767 (2^{15}-1)
Output Data Type	<integer>
*RST State	None
Example	`:STAT:QUES:COND?`

:STATus:QUEStionable:ENABle

This command enables bits in the Questionable Enable register (QER). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the QER register and the corresponding QSR register bit is also set, the Standard Operation Status Summary bit (bit 7) in the STB register is set.

Bits enabled in QER and set in QSR/QCR register will result in the Standard Questionable Status Summary bit (bit 3) in the STB register being set. See Figure 4.

Syntax	:STATus:QUEStionable:ENABle <integer> :STATus:QUEStionable:ENABle?
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:QUES:ENAB 256` `:STAT:QUES:ENAB?`

:STATus:QUEStionable:NTRansition

This command enables bits in the Questionable Negative Transition Register (QNTR). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the QNTR, a high-to-low transition in the QCR bit will set the corresponding QSR bit. See Figure 4.

Syntax	:STATus:QUEStionable:NTRansition <integer>
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:QUES:NTR 256`

:STATus:QUEStionable:PTRansition

This command enables bits in the Questionable Positive Transition Register (QPTR). The decimal integer value entered is the binary equivalent of the desired 16-bit mask to be enabled. When a bit is set in the QPTR, a low-to-high transition in the QCR bit will set the corresponding QSR bit. See Figure 4.

Syntax	:STATus:QUEStionable:PTRansition <integer>
Parameter/Response	0 – 32767 (2^{15}-1)
I/O Data Type	<integer>
*RST State	0
Examples	`:STAT:QUES:PTR 256`

:STATus:TEMPerature?

This command queries the internal ambient temperature of the D4000.

Syntax	:STATus:TEMPerature?
Query Response	<NRf>
Unit	Degrees Celsius
*RST State	None

INPut Commands

:INPut:COUPling?

This command queries the current input coupling method.

Syntax	:INPut:COUPling?
Query Response	AC
Output Data Type	<character>
*RST State	AC
Examples	`:INP:COUP?`

:INPut:FILTer:PRESelect

This command sets or queries the current input pre-select filter. Any out of range index will result in an Execution Error response.

Syntax	:INPut:FILTer:PRESelect <Index> :INPut:FILTer:PRESelect? [MAX]
Parameter	Index
Input Data Type	<integer>
Allowable Values	1 - <number of pre-select filters, model dependent>
Query Response	<integer>
*RST State	Dependent on default input frequency
Examples	`:INP:FILT:PRES 2` `:INP:FILT:PRES?`

:INPut:GAIN

This command sets or queries the gain setting of the D4000's RF input.

Note: This gain refers to the switch-able pre-amp of the RF Input.

Syntax	:INPut:GAIN {ON | OFF | 1 | 0} :INPut:GAIN?
Parameter	ON | OFF | 1 | 0
Input Data Type	<boolean>
Query Response	1 | 0
Output Data Type	<integer>
*RST State	0
Examples	`:INP:GAIN ON` `:INPUT:GAIN?`

SENSe Commands

[:SENSe]:ATTenuator

This command sets or queries the value of the front-end attenuator.

Syntax	[:SENSe]:ATTenuator <attenuation> [:SENSe]:ATTenuator?
Parameter/Response	0 to 30 dB attenuation, in 1 dB step
I/O Data Type	<integer>
Unit	dB
*RST State	10
Examples	`:SENSE:ATTENUATOR 30` `:SENSE:ATT?`

[:SENSe]:DCONverter:BAND?

This command queries information about the band edge. Use with [:SENSe]:DCONverter:BAND:COUNt? command. Any out of range index will result in an Execution Error response.

Syntax	[:SENSe]:DCONverter:BAND? <index>
Parameter	Index
Input Data Type	<integer>
Query Response	<bandstart>,<bandstop>
Output Data Type	<integer>,<integer>
Unit	Hz,Hz
*RST State	N/A
Examples	`:SENSE:DCONverter:BAND? 2`

[:SENSe]:DCONverter:BAND:COUNt?

This command queries the number of bands in the downconverter

Syntax	[:SENSe]:DCONverter:BAND:COUNt?
Query Response	<integer>
*RST State	N/A
Examples	`:DCONverter:BAND:COUNt?`

[:SENSe]:DCONverter:BYPass

This command sets or queries the state of the downconverter bypass.

Syntax	[:SENSe]:DCONverter:BYPass {ON | OFF | 1 | 0}
Parameter	ON | OFF | 1 | 0
Input Data Type	<boolean>
Query Response	1 | 0
Output Data Type	<integer>
*RST State	0
Examples	`:DCONverter:BYPASS 1`

[:SENSe]:FREQuency:CENTer

This command sets or queries the expected frequency at the RF input. The LO frequencies are set according to the frequency plan to downconvert the signal to the IF output.

The tuning resolution is 100 kHz, any frequency values that are not a multiple of the 100 kHz will be round down to the nearest 100 kHz value.

Syntax	[:SENSe]:FREQuency:CENTer <NRf [unit]> [:SENSe]:FREQuency:CENTer? [{MAX | MIN}]
Set Parameter	<center frequency [unit]>
Input Data Type	NRf [character]
Allowable Values	Range: 24.0 GHz – 40.0 GHz Tuning Resolution: 100 kHz
Query Parameters	Optional {MAX | MIN}
Input Data Type	[<character>]
Query Response	<integer>
Default I/O Unit	Hz
*RST State	40000000000

Examples	`:FREQ:CENTER 27.5 GHZ`
	`SENSE:FREQ:CENT 28000000000`
	`FREQ:CENT?`

[:SENSe]:LO:COUNt?

This command queries the number of LO in the system.

Syntax	[:SENSe]:LO:COUNt?
Query Response	<integer>
*RST State	N/A
Examples	`SENS:LO:COUNT?`

[:SENSe]:LO<1|2>:FREQuency

This command sets or queries the current LO frequencies. The tuning resultion is 100 kHz, any frequency values that are not a multiple of the 100 kHz will be round down to the nearest 100 kHz value.

Syntax	[:SENSe]:LO<1|2>:FREQuency <NRf [unit]> [:SENSe]:LO<1|2>:FREQuency? [{MIN | MAX}]
Set Parameter	<LO frequency [unit]>
Input Data Type	NRf [character]
Query Parameters	Optional {MAX | MIN}
Input Data Type	[<character>]
Query Response	<integer>
Default I/O Unit	Hz
*RST State	Depending on the tuning center frequency
Examples	`SENS:LO1:FREQ 24.15 GHz` `SENS:LO2:FREQ?`

[:SENSe]:LO<1|2>:LOCK?

This command queries the state of the lock status for an LO.

Syntax	[:SENSe]:LO[1|2]:LOCK?
Query Response	1 | 0
Output Data Type	<integer>
*RST State	N/A
Examples	`SENS:LO1:LOCK?` `LO2:LOCK?`

[:SENSe]:RF:LOCK?

This command queries the state of the lock status for the downconverter.

Syntax	[:SENSe]:RF:LOCK?
Query Response	1 | 0
Output Data Type	<integer>
*RST State	N/A
Examples	`SENS:RF:LOCK?`

SOURce Commands

[:SOURce]:REFerence

This command selects and queries the 10 MHz reference clock source, whether it be via the internal source or through the external SMA connector.

Caution: When the external 10 MHz reference input is used, its reference level **must** be between 0 and 10 dBm, with 5 dBm recommended. Exceeding the level of 10 dBm will result in permanent damage to the internal clock circuit. Additionally, the 10 MHz reference must be powered down prior to powering down the D4000.

Syntax	:SOURce:REFerence {INT | EXT} :SOURce:REFerence?
Parameter/Response	INT | EXT
I/O Data Type	<character>
*RST State	INT
Examples	`:SOURCE:REF INT` `:SOUR:REF?`

[:SOURce]:REFerence:AUTO

This command enables or disables the automatic selection of the reference clock.

Syntax	[:SOURce]:REFerence:AUTO {ON | OFF | 1 | 0} [:SOURce]:REFerence:AUTO?
Parameters	ON | OFF | 1 | 0
Input Data Type	<boolean>
Query Response	1 | 0
Output Data Type	<integer>
*RST State	1
Examples	`SOUR:REF:AUTO ON` `SOURCE:REFERENCE:AUTO?`

[:SOURce]:REFerence:FREQuency?

This command queries the reference clock's frequency.

Syntax	[:SOURce]:REFerence:FREQuency?
Query Response	<integer>
Default I/O Unit	Hz
*RST State	N/A
Examples	`SOUR:REF:FREQ?`

[:SOURce]:REFerence:OUTPut:ENABle

This command enables or disables the reference clock output.

Syntax	[:SOURce]:REFerence:OUTPut:ENABle {ON | OFF | 1 | 0} [:SOURce]:REFerence:OUTPut:ENABle?
Parameters	ON | OFF | 1 | 0
Input Data Type	<boolean>
Query Response	1 | 0
Output Data Type	<integer>
*RST State	0
Examples	`SOUR:REF:OUTP:ENABl 1` `SOURCE:REFERENCE:OUTPUT:ENABlE?`

OUTPut Commands

:OUTPut:ATTenuator

This command sets or queries the attenuation setting of the D4000's IF output.

Syntax	:OUTPut:ATTenuator <NRf> :OUTPut:ATTenuator?
Parameter	<dB attenuation>
Input Data Type	<NRf>
Allowable Values	0 to 31.25 in 0.25 dB step
Query Response	<dB attenuation>
Output Data Type	<NRf>
Default I/O Unit	dB
*RST State	Depending on the tuning center frequency
Examples	`:OUTPUT:ATTENUATION 31` `:OUTP:ATT?`

:OUTPut:IF:ATTenuator

This command queries or sets the middle IF attenuator.

Syntax	:OUTPut:IF:ATTenuator <attenuation> :OUTPut:IF:ATTenuator?
Parameter/Response	0 to 30 dB attenuation in 1 dB step
I/O Data Type	<integer>
Unit	dB
*RST State	Depends on the device calibration
Examples	`:OUTP:IF:ATT?` `:OUTPUT:IF:ATTENUATOR 30`

:OUTPut:IF:FREQuency?

This command queries the output IF frequency.

Syntax	:OUTPut:IF:FREQuency?
Query Response	Frequency
Output Data Type	<integer>
Default I/O Unit	Hz
Examples	`:OUTPUT:IF:FREQ?`

Appendix A: Booting up and Connecting to the D4000

This section provide a brief information regarding booting up and connecting to the D4000. Refer to the D4000 User Guide for more usage instructions.

Bootup Sequence

The Downconverter starts up in the following manner:

1. Apply power to the Downconverter to power up the unit. The unit takes a few seconds to boot and get ready. When powered and ready, the LED on the panel will turn green.
2. Establish a TCP/IP connection using one of the connection methods described in the following sections.
3. Once the TCP/IP connection is successful, the unit is ready for interfacing using SCPI commands described in this document.

See Code Example of TCP/IP Connection and SCPI Control section for a C code example of establishing raw TCP/IP connection and sending some SCPI commands in a Windows system.

Connecting to D4000

ThinkRF's Downconverters are network ready devices conveying control commands and data using the TCP/IP protocol. Network application access is via SCPI Raw.

SCPI commands and responses are sent as character strings terminated by a Program Message Terminator (PMT) as defined in IEEE-488.2. The PMT is typically a newline character (ASCII-encoded byte 10) in purely text-based access method like SCPI Raw.

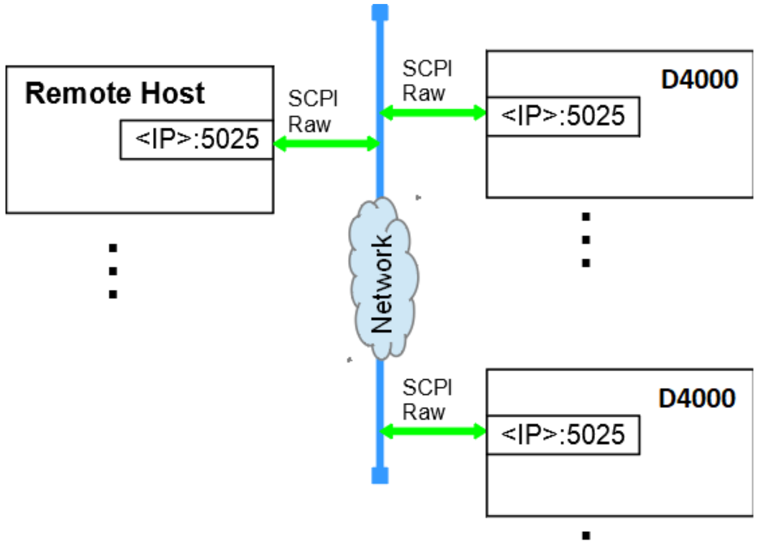
Note: The default configuration at the first power up is set to STATIC type with IP 192.168.1.2.

To change the network configuration, such as changing to DHCP or STATIC type, see the `:SYSTem:COMMunicate:LAN` command set of the SYSTem Commands section for the appropriate SCPI commands to use.

SCPI Raw

SCPI Raw uses a TCP/IP connection to establish a bidirectional link with minimal overhead. Although it can be used as a command line interface, it is better suited for use with an application.

SCPI Raw is accessible via TCP port 5025, as shown in the following picture.



Code Example of TCP/IP Connection and SCPI Control

The following code is a simple example, written in C, to illustrate how to establish TCPI/IP connection, send SCPI commands and receive responses with a D4000.

```
/*****************************************************************************
 * A demo of TCP/IP client socket connection to a Downconverter (DCN) at port
 * 5025, with sending and receiving some example SCPI commands.
 *
 * NOTES:
 * - This example is for Windows socket only.
 *
 * Usage: tcpip_scpi_ex.exe <IP of the DNC>
 * ex: tcpip_scpi_ex.exe 192.168.1.2
 *
 * References:
 * For TCP/IP socket related background:
 * https://msdn.microsoft.com/en-us/library/windows/desktop/ms738545.aspx
 * http://beej.us/guide/bgnet/
 *****************************************************************************/

#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <math.h>

#pragma comment(lib, "Ws2_32.lib")

#define DNC_PORT "5025"  // the DNC raw TCP/IP socket port to be connected to
#define MAX_SCPI_LEN 512 // max number of characters for a SCPI string

void *get_in_addr(struct sockaddr *);
const char *_inet_ntop(int, const void *, char *, socklen_t);
int socket_setup(const char *, int *, const char *, int);
int socket_send(int, char const *, int);
int socket_recv(int, unsigned char *, int, unsigned int, int *);


/**
```

```
 * Main function: establish connection to a DNC to do some examples of
 * SCPI control/communication with a DNC.
 */
int main(int argc, char *argv[])
{
    struct WSAData ws_data;   // create an instance of Winsock data type
    int sock_fd;
    char scpi_cmd[4][MAX_SCPI_LEN];
    char scpi_rsp[MAX_SCPI_LEN];
    int result, i;
    int bytes_rcvd;
    int timeout = 2000;              // in milliseconds

    // Check for the correct number of arguments
    if (argc != 2) {
       fprintf(stderr, "Usage: %s <Server IP>\n", argv[0]);
       exit(1);
    }

    // Initialize Winsock
    result = WSAStartup(MAKEWORD(2,2), &ws_data);
    if (result != 0) {
        fprintf(stderr, "WSAStartup failed: %d\n", result);
    }

    result = socket_setup(argv[1], &sock_fd, DNC_PORT, timeout);
    if (result != 0) {
        fprintf(stderr, "Program terminates with error in socket_setup().\n");
    }

    // **********
    // Sends some SCPI commands to show that the DNC is interacting
    // Suggestion: rewrite this section to use a loop to do any SCPI control
    // **********


    // ----------
    // Example 1: Queries
    strcpy(scpi_cmd[0], ":SYSTEM:ERROR?");
    strcpy(scpi_cmd[1], "*IDN?");
    strcpy(scpi_cmd[2], ":INP:DCON:MAN:FILT:PRES?");
    strcpy(scpi_cmd[3], "OUTPUT:IF:FREQ?");

    for (i = 0; i < 4; i++) {
        result = socket_send(sock_fd, scpi_cmd[i], sizeof(scpi_cmd[i]));
        if (result < 0) exit(1);

        result = socket_recv(sock_fd, scpi_rsp, MAX_SCPI_LEN, timeout,
                             &bytes_rcvd);
        if (result < 0) exit(1);
        if (scpi_rsp[bytes_rcvd - 1] == '\n')
            scpi_rsp[bytes_rcvd - 1] = '\0';
        printf("%s query returns: %s\n\n", scpi_cmd[i], scpi_rsp);
    }

    // ----------
    // Example 2: set and queries
    strcpy(scpi_cmd[0], ":FREQ:CENT 29 GHZ");
    strcpy(scpi_cmd[1], "FREQ:CENT?");
    // an example compound commands
    strcpy(scpi_cmd[2], ":INP:DCON:MAN:FILTER:PRES 1;:INP:GAIN ON");
    strcpy(scpi_cmd[3], ":INPUT:GAIN?");

    for (i = 0; i < 4; i += 2) {
        // Send set command
        result = socket_send(sock_fd, scpi_cmd[i], sizeof(scpi_cmd[i]));
        if (result < 0) exit(1);
```

```
        // Send the query of the value just set
        result = socket_send(sock_fd, scpi_cmd[i+1], sizeof(scpi_cmd[i+1]));
        if (result < 0) exit(1);

        // check the response
        result = socket_recv(sock_fd, scpi_rsp, MAX_SCPI_LEN, timeout,
&bytes_rcvd);
        if (result < 0) exit(1);

        // strip out anything after end of line, potentially due to previous
response
        if (scpi_rsp[bytes_rcvd - 1] == '\n')
            scpi_rsp[bytes_rcvd - 1] = '\0';
        printf("%s query returns %s\n\n", scpi_cmd[i+1], scpi_rsp);
    }

    closesocket(sock_fd);

    return 0;
}

/**
 * Get sockaddr of IPv4 or IPv6
 *
 *  sock_addr - a socket address structure
 *
 * Return: The socket address
 */
void *get_in_addr(struct sockaddr *sock_addr)
{
    if (sock_addr->sa_family == AF_INET)
        return &(((struct sockaddr_in*) sock_addr)->sin_addr);

    return &(((struct sockaddr_in6*) sock_addr)->sin6_addr);
}


/**
 * similar to inet_ntop so that old windows version could use it
 */
const char *_inet_ntop(int af, const void *src, char *dst, socklen_t cnt)
{
    if (af == AF_INET) {
        struct sockaddr_in in;

        memset(&in, 0, sizeof(in));
        in.sin_family = AF_INET;
        memcpy(&in.sin_addr, src, sizeof(struct in_addr));
        getnameinfo((struct sockaddr *) &in, sizeof(struct
            sockaddr_in), dst, cnt, NULL, 0, NI_NUMERICHOST);

        return dst;
    }
    else if (af == AF_INET6) {
        struct sockaddr_in6 in;

        memset(&in, 0, sizeof(in));
        in.sin6_family = AF_INET6;
        memcpy(&in.sin6_addr, src, sizeof(struct in6_addr));
        getnameinfo((struct sockaddr *) &in, sizeof(struct
            sockaddr_in6), dst, cnt, NULL, 0, NI_NUMERICHOST);

        return dst;
    }
    return NULL;
```

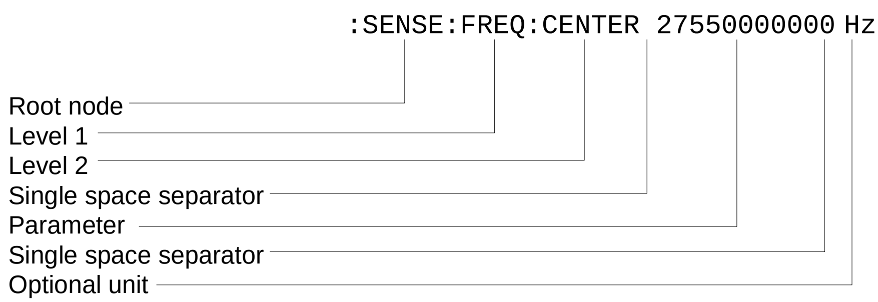
```
}


/**
 * Look up, verify and establish the socket once deemed valid
 *
 *  sock_addr - the IP address
 *  sock_fd - a socket file descriptor with specific socket value to be set up
 *  sock_port - the socket port
 *
 * Return: 0 for success or negative value when fail.
 */
int socket_setup(const char *sock_addr, int *sock_fd, const char *sock_port,
                int timeout)
{
    struct addrinfo hint_ai, *ai_list, *ai_ptr;
    int temp_fd = 0;
    int get_ai;
    char str[INET6_ADDRSTRLEN];
    struct timeval tv;

    // Construct the local address structure
    memset(&hint_ai, 0, sizeof(hint_ai)); // Zero out structure
    hint_ai.ai_family = AF_UNSPEC;        // Unspec to use with any address
                                          // family (IPv4, IPv6, etc.)
    hint_ai.ai_socktype = SOCK_STREAM;    // For TCP/IP type
    hint_ai.ai_flags = 0;
    hint_ai.ai_protocol = 0;              // to auto chose the protocol

    // Check the address at the given port
    get_ai = getaddrinfo(sock_addr, sock_port, &hint_ai, &ai_list);
    if (get_ai != 0) {
        fprintf(stderr, "getaddrinfo(): %s\n", gai_strerror(get_ai));
        return -1;
    }

    // loop through all the results and connect to the first we can
    for(ai_ptr = ai_list; ai_ptr != NULL; ai_ptr = ai_ptr->ai_next) {
        temp_fd = socket(ai_ptr->ai_family, ai_ptr->ai_socktype,
                        ai_ptr->ai_protocol);
        if (temp_fd == -1) {
            perror("client: socket()");
            continue;
        }

        tv.tv_sec  = timeout / 1000;
        tv.tv_usec = timeout * 1000;

        // Note: this setup does not work with win32, replace timeout where tv
        // is directly
        setsockopt(temp_fd, SOL_SOCKET, SO_RCVTIMEO, (char*) &tv, sizeof(tv));

        if (connect(temp_fd, ai_ptr->ai_addr, ai_ptr->ai_addrlen) == -1) {
            perror("client: connect()");
            closesocket(temp_fd);
            continue;
        }

        break;
    }

    if (ai_ptr == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return -1;
    }

    // convert IP's binary representation to network printable presentation
```

```
    _inet_ntop(ai_ptr->ai_family, get_in_addr((struct sockaddr *)ai_ptr->ai_addr),
            str, sizeof(str));
    printf("client: connected to %s\n", str);

    // all done with this structure so free the list
    freeaddrinfo(ai_list);

    *sock_fd = temp_fd;

    return 0;
}


/**
 * Sends a string to the server.
 *
 *  sock_fd - the socket at which the data will be received.
 *  out_str - the string to be sent out
 *  len - length of the string to be sent
 *
 * Return: Number of bytes sent on success, or negative otherwise.
 */
int socket_send(int sock_fd, char const *out_str, int len)
{
    int total_txed = 0;
    int bytes_txed;
    int bytes_left = len;
    char cmd_str[MAX_SCPI_LEN];

    // Add end of line character, required for SCPI command
    cmd_str[0] = '\0';
    strcat(cmd_str, out_str);
    strcat(cmd_str, "\n");

    // Loop to send all the bytes
    while (total_txed < len) {
        bytes_txed = send(sock_fd, cmd_str + total_txed, bytes_left, 0);

        // Check the returned value
        if (bytes_txed > 0) {
            fprintf(stderr, "Sent '%s' to server.\n", out_str);

            // Update all the count
            total_txed += bytes_txed;
            bytes_left -= bytes_txed;
        } else if (bytes_txed == -1) {
            return -1;
        } else {
            // Client closed connection before we could reply to
            // all the data it sent, so exit early.
            return -1;
        }
    }

    return total_txed;
}


/**
 * Reads data from the server socket of buf_size bytes at a time.
 * It does not loop to keep checking buf_size of bytes received.
 * This socket receive function makes used of select() function to check for
 * data availability before receiving, & FD_SET to make receiving non blocking.
 *
 *  sock_fd - the socket at which the data will be received
 *  rx_buf_ptr - a buffer to store the incoming bytes
```

```
 *  buf_size - the size of the buffer in bytes
 *  time_out - time out in milliseconds.
 *  bytes_received - number of bytes read (on success)
 *
 * Return: 0 on success or a negative value on error
 */
int socket_recv(int sock_fd, unsigned char *rx_buf_ptr, int buf_size,
                      unsigned int time_out, int *bytes_received)
{
    fd_set read_fd; // temp file descriptor for select()
    int ret_val;    // return value of a function

    struct timeval timer;
    long seconds = (long) floor(time_out / 1000.0);

    // Set the time out timer
    timer.tv_sec = seconds;
    timer.tv_usec = (long) (time_out - (seconds * 1000)) * 1000;

    FD_ZERO(&read_fd);
    FD_SET(sock_fd, &read_fd);
    ret_val = select(sock_fd + 1, &read_fd, NULL, NULL, &timer);

    // Make reading of socket non-blocking w/ time-out of s.ms sec
    if (ret_val == -1) {
        fprintf(stderr, "init select() function returned with error %d (\"%s\")",
                errno, strerror(errno));

        return -1;
    } else if (ret_val) {
        //fprintf(stderr, "Data is available now.\n");
    } else {
        fprintf(stderr, "No data received within %d milliseconds.\n", time_out);
        if(ret_val) {
            fprintf(stderr, "In socket_recv: select returned %d\n", ret_val);
        }

        return -1;
    }

    // If the socket is read-able, rx packet of incoming data
    if (FD_ISSET(sock_fd, &read_fd)) {
        ret_val = recv(sock_fd, (char *) rx_buf_ptr, buf_size, 0);
        if (ret_val == 0) {
            fprintf(stderr, "Connection is already closed.\n");

            return -1;
        } else if (ret_val < 0) {
            fprintf(stderr, "recv() function returned with error %d (\"%s\")",
                    errno, strerror(errno));

            return -1;
        }

        //fprintf(stderr, "Received (%d bytes)\n", ret_val);
    }

    *bytes_received = ret_val;
    return 0;
}
```

Appendix B: SCPI Command Syntax

Each SCPI command consists of a root node, one or more lower level nodes, follow by applicable parameters and separators:



Entering Commands

SCPI commands have both a long and short version, such as :FREQUENCY and :FREQ. The SCPI interface responds to either version, but will not respond to variations of either version. The interface does not differentiate between upper-case and lower-case letters but only the long or short form of a command.

An example correct and incorrect SCPI entry format for :FREQuency command:

	Command Entry		
Correct Entry	:FREQUENCY :FREQ	:FREQuency :freq	:frequency
Incorrect Entry	:fre :FREQu	:FREQU	

Note:
- At the end of each SCPI command string, whether a single command or multiple commands separated by semicolons ";", *a new line-feed or carriage return is required.* Example in C: ":FREQ:CENTER 27 GHZ\n" or "FREQ:CENT 27 GHZ;INP:GAIN 1\n".
- Maximum number of characters allow for a SCPI command string (single or multiple commands) is 512.

Notation

<table>
<tr><th>Notation</th><th>Description</th></tr>
<tr><td>:</td><td>Links command keywords together</td></tr>
<tr><td>;</td><td>Separates multiple commands entered together on a single program message</td></tr>
<tr><td>single space</td><td>Uses to separate a parameter from a command or unit from a parameter</td></tr>
<tr><td>,</td><td>Uses to separate multiples parameters of a command</td></tr>
<tr><td>[]</td><td>Uses to optionally enclose zero or more parameters</td></tr>
<tr><td>{.} or {.}*</td><td>The enclosed item maybe included zero or more times</td></tr>
<tr><td>{.}+</td><td>The enclosed items occurs one or more times</td></tr>
<tr><td>{.|.|.}</td><td>One and only one of the two or more enclosed items separated by | maybe included</td></tr>
<tr><td><></td><td>Uses to enclose required parameter descriptions</td></tr>
<tr><td>?</td><td>Indicates query command, use where applicable</td></tr>
<tr><td>|</td><td>Indicates “or” and is used to separate alternative parameter options</td></tr>
<tr><td>::=</td><td>Means “is defined as”</td></tr>
</table>

Parameter types

This section defines different SCPI parameter data type.

<table>
<tr><th>Parameter Type</th><th>Description</th></tr>
<tr><td><boolean></td><td>ON | OFF | 1 | 0
Boolean parameters are always returned as 1 or 0 in NR1 format by query commands</td></tr>
<tr><td><integer>
<integer></td><td>Unsigned integer of NR1 format
Ex: 1 or 3432</td></tr>
<tr><td><NR1></td><td>Signed integer without a decimal point (implied radix point)
Ex: -25 or 0</td></tr>
<tr><td><NR2></td><td>Signed number with an explicit radix point
Ex: -1.234 or 1.0 or 0.0</td></tr>
<tr><td><NR3></td><td>Scaled explicit decimal point numeric value with and an exponent
Ex: 2.73e+2 or 2.351e2</td></tr>
<tr><td><NRf></td><td><NR1>|<NR2>|<NR3></td></tr>
<tr><td><NRr></td><td>Non-decimal numeric value such as hexadecimal, octal or binary</td></tr>
<tr><td><char>
<character></td><td>Character program data
Ex: MAXimum or MINium</td></tr>
<tr><td><string></td><td>ASCII string surrounded by single or double quotes
Ex: “This is an example”</td></tr>
</table>

Default Units

Parameter	Default Unit
frequency	Hz
time	s or ns where applicable
voltage	V
absolute amplitude	dBm
relative amplitude	dB

Units other than the default may be specified. If units are not specified then the default units apply. Note the following examples, which are all equivalent.

Example	`:FREQ:CENTer 27.55 GHz`
is equivalent to	`:FREQ:CENTer 27550000000`
is equivalent to	`:FREQ:CENTer 27550000000 Hz`
is equivalent to	`:FREQ:CENTer 27550 MHZ`
is equivalent to	`:FREQ:CENTer 27.55e9`

Appendix C: SCPI Status and Event Registers

The Downconverter's SCPI interface has a status and event reporting system that enables the user to handle device events. The interface conforms to IEEE Std 488.2-1987 and SCPI 1999.0. This section discusses these status registers, status register enable masks, event queues and event handling.

Status Byte Register (SBR)

The SBR is used to determine the specific nature of the event or condition. It can be read by issuing a *STB? command. The individual summary bits of the SBR are cleared by reading the underlying event registers in the Status Data Structures (SDS) issuing a *CLS command or by clearing the status event registers

Bits in the SBR will be set only when the corresponding bits in the Service Request Enable Register are set.

Bit	Name	Description
0	not used	This bit is available for future vendor use.
1	not used	This bit is available for future vendor use.
2	Error / Event Available (EAV)	This bit is set if there are any unread error or event in the System Error queue. Error and event messages are read using the :SYSTem:ERRor[:NEXT]? query command.
3	Questionable Register Summary	Summary of the Questionable Status register
4	Message Available (MAV)	This bit is set if there is any unread data in the message queue.
5	Standard Event Status Bit (ESB)	This bit is set if there is any unread or non-cleared data in the Standard Event Status register.
6	Request Service	Summary of the Request Service register.
7	Operation Register Summary	Summary of the Operation Status register

Standard Event Status Register (ESR)

The ESR is used to determine the nature of the status and error conditions. It is read by issuing a *ESR? command. The contents of the ESR are clear by issuing either a *ESR? or *CLS command.

Bits in the ESR will be set only when the corresponding bits in the Standard Events Status Enable Register are set.

Bit	Name	Description
0	Operation Complete (OPC)	Set to indicate that all pending operations are complete and the D4000 is ready to accept another command, or that query results are available.
1	Request Control (RQC)	This bit is not used and is always 0.

Bit	Name	Description
2	Query Error (QYE)	Set to indicate that a query has been made for which no response is available. Query errors have SCPI error codes from –499 to –400.
3	Device Dependent Error (DDE)	Set to indicate that a device-dependent error has occurred. Device-dependent errors have SCPI error codes from –399 to –300 and 1 to 32767.
4	Execution Error (E)	Set to indicate that a parameter exceeds its allowed range. Execution errors have SCPI error codes from –299 to –200.
5	Command Error (CME)	Set to indicate that a command error has occurred. Command errors have SCPI error codes from –199 to –100.
6	not used	This bit is always 0.
7	Power ON (PON)	Set once upon power-up. This bit has no effect on the Error / Event Available (EAV) bit in the Status Byte Register.

Operational Status (OSR) Register

The OSR is a 16-bit register that is used to determine the state of operation. The current states are read by issuing a :STATus:OPERation:CONDition? command. The latched event states are read and cleared by issuing a :STATus:OPERation[:EVENt]? command.

Bit	Name	Description
0	not used	This bit is reserved for the CALIbrating event.
1	SETTling	Set to indicate that the D4000 is in the process of tuning and that the IF output signal is not yet valid.
2	not used	This bit is reserved for the RANGing event.
3	not used	This bit is reserved for the SWEeping event.
4	not used	This bit is reserved for the MEASuring event.
5	not used	This bit is reserved for the TRIGger event.
6	not used	This bit is reserved for the ARM event.
7	not used	This bit is reserved for the CORRecting event.
8	not used	Set to indicate that the signal at the IF output is spectrally inverted
9-12	not used	This bit is available for future vendor use.
13	not used	This bit is reserved for the INSTrument summary bit.
14	not used	This bit is reserved for the PROGram running bit.
15	reserved	This bit is always 0.

Questionable Status (QSR) Register

The QSR is a 16-bit register that is used to indicate abnormal events. The current states are read by issuing a :STATus:QUEStionable:CONDition? command. The latched event states are read and cleared by issuing a :STATus:QUEStionable[:EVENt]? command.

Bit	Name	Description
0	not used	This bit is reserved for the VOLTage event.
1	not used	This bit is reserved for the CURRent event.
2	not used	This bit is reserved for the TIME event.
3	not used	This bit is reserved for the POWer event.

Bit	Name	Description
4	TEMPerature	This bit is set if the D4000 temperature is outside of normal operational limits.
5	not used	This bit is reserved for the FREQuency event.
6	not used	This bit is reserved for the PHASe event.
7	not used	This bit is reserved for the MODulation event.
8	not used	This bit is reserved for the CALibration event.
9-12	not used	These bits are available for future vendor use.
13	not used	This bit is reserved for the INSTrument summary bit for devices that have multiple logical instruments.
14	not used	This bit is reserved for the PROGram running bit.
15	reserved	This bit is always 0.

Output Queue

The D4000 has an Output FIFO Queue that is structured as a FIFO and holds the response messages to queries. The SBR's MAV bit is set when there are messages in the queue. The unread results of a previous command are cleared from the queue when a new command or query is received.

Error and Event Queue

The D4000 has an Error and Event FIFO Queue that holds up to 16 errors and events. It is queried using the :SYSTem:ERRor[:NEXT]? command. The *CLS command clears all entries from the queue.

Appendix D: SCPI Error Codes Used

Code	Message	Description
0	No error	
Command error, range [-199, -100]		
-144	Character data too long	The character data contained more than 12 characters.
-171	Invalid expression	The command syntax was incorrect.
Execution error, range [-299, -200]		
-200	Execution error	A generic execution error for which more specific information is not available.
-210	Trigger error	
-220	No matched module	The specific operation is not installed.
-221	Settings conflict	Indicates that a legal program data element was parsed but could not be executed due to the current device state
-222	Data out of range	A parameter was of the proper type but outside of the defined range for the specific command.
-223	Too much data	A parameter was received that contained more data than the device could handle.
-224	Illegal parameter value	A parameter was received that is NOT allowed for the particular command.
-230	Data corrupt or stale	Possibly invalid data; new reading started but not completed since last access.
-240	Hardware error	Indicates that a legal program command or query could not be executed because of a hardware problem in the device.
-241	Hardware missing	Indicates that a legal program command or query could not be executed because of missing device hardware. For example, an option is not installed
Device specific error, range [-399, -300]		
-310	System error	
-321	Out of memory	An internal operation needed more memory than that was available.
-330	Self test failed	
-340	Calibration failed	
-350	Query overflow	The SCPI remote interface error queue overflowed.
Query error, range [-499, -400]		
-410	Query INTERRUPTED	A condition causing an INTERRUPTED query error occurred
ThinkRF's Downconverter Specific, range [-999, -900]		
-901	No data	Read trace command issued while there is no data available.
-911	Need firmware upgrade	The current firmware needs upgrading.
-912	Invalid option license	The option could not be installed because of invalid license.

Appendix E: SCPI Commands Quick Reference

This section summarizes the SCPI commands available for interfacing with D4000. The commands are listed alphabetically based on the main node, then sub-nodes, so on. The sub-nodes are grouped and listed alphabetically based on functionality.

See Appendix B's Notation section for details on notations used in the Parameter column.

The Release column indicates from which **firmware** release version that the commands are available. **Grayed-out** commands are not yet implemented.

Keyword	Parameter	Description	Release
IEEE Mandated		*Page 14*	
*CLS		Clear all status registers	
*ESE	<integer>	Event Status Enable register	
*ESE?		Query ESE register	
*ESR?		Query Event Status Register	
*IDN?		Query device identification	v1.0
*OPC		Operation Complete	
*OPC?		Query OC	v1.0
*RST		Reset to factory default	v1.0
*SRE	<integer>	Service Request Enable bits	
*SRE?		Query SRE register	
*STB?		Query Status Byte register	
*TST?		Query self-test status	v1.0
*WAI		Wait-to-Continue	
:INPut		*Page 27*	
:COUPling?		Query the input coupling mode	v1.0
:FILTer			
:PRESelect	{1 | 2}	Select the input preselect filter	v1.0
:PRESelect?	[MAX]	Query the input preselect filter	v1.0
:AUTO	{ON | OFF | 1 | 0}	Set the automatic selection of the preselect filter	v1.0
:AUTO?		Query the automatic selection of the preselect filter	v1.0
:GAIN	{ON | OFF | 1 | 0}	Set an input gain stage to be on or off.	v1.0
:GAIN?			
:OUTput		*Page 32*	
:ATTenuator	<NRf>	Set the IF output attenuation from 0 to 31.25 dB in 0.25 dB steps	v1.0
:ATTenuator?		Query the IF output attenuation setting	v1.0
:IF			
:ATTenuator		Set the middle IF attenuator, 0 to 30 dB in 1 dB step	v1.0
:ATTenuator?		Query the middle IF attenuator value	v1.0
:FREQuency?		Query the IF frequency at the IF output	v1.0
[:SENSe]		*Page 28*	
:ATTenuator		Set the front-end IF attenuator, 0 to 30 dB in 1 dB step	v1.0
:ATTenuator?		Query the front-end IF attenuator value	v1.0

Keyword	Parameter	Description	Release
:DCONverter			
:BAND?		Queries band edge information	
:BAND			
:COUNt?		Queries the number of bands	
:FREQuency			
:CENTer	24.0 GHz – 40.0 GHz with 100 kHz resolution	Set the center frequency of the D4000	v1.0
:CENTer?	[{MAX | MIN}]	Query the center frequency of the D4000	v1.0
:LO			
:COUNt?		Queries the number of LOs	
:LO<1|2>			
:FREQuency	<NRf [unit]>	Set the LO1 or LO2 frequency	v1.0
:FREQuency?		Query the LO1 or LO2 frequency	v1.0
:LOCK?		Queries the lock status for an LO (LO1, LO2)	v1.0
:RF			
:LOCK?		Queries the lock status of the downconverter	v1.0
[:SOURce]			
:REFerence	{INT | EXT}	Select the 10 MHz reference clock source	v1.0
:REFerence?		Queries the 10 MHz reference clock source	v1.0
:AUTO	{ON | OFF | 1 | 0}	Enable auto selection of reference clock source	v1.0
:AUTO?		Query auto selection of reference clock source	v1.0
:FREQuency?		Query reference clock frequency	
:OUTPut			
:ENABle	{ON | OFF | 1 | 0}	Enable/disable the reference clock output	v1.0
:ENABle?		Query the reference clock output state	v1.0
:STATus		*Page 20*	
:OPERation		Return the standard Operation Status Register (OSR) for any event	
[:EVENt]?			v1.0
:CONDition?			v1.0
:ENABle	<integer>		v1.0
:ENABle?			
:PRESET		Presets the D4000 (similar to *RST)	v1.0
:QUEStionable		Return the standard Questionable Status Register (QSR) for any event	
[:EVENt]?			v1.0
:CONDition?			v1.0
:ENABle	<integer>		v1.0
:ENABle?			
:TEMPerature?		Return the D4000's internal ambient temperature	v1.0
:SYSTem		*Page 16*	
:COMMunicate			
:LAN			
:APPLy		Apply the new Downconverter's LAN settings from the commands above, which will then take effect. This command should be applied only once all the required LAN settings have been set.	v1.0
:CONFigure	{DHCP | STATIC}	Set the Downconverter's LAN to use DHCP or STATIC configuration type	v1.0

Keyword	Parameter	Description	Release
:CONFigure?	[CURRENT]		v1.0
:GATEway	<IPv4 address>	Set the Downconverter's LAN Gateway address	v1.0
:GATEway?	[CURRENT]		
:IP	<IPv4 address>	Set the new IPv4 address for the Downconverter's LAN	v1.0
:IP?	[CURRENT]		
:NETMask	<IPv4 address>	Set the Downconverter's LAN netmask address	v1.0
:NETMask?	[CURRENT]		
:ERRor [:NEXT]?		Return the next SCPI error/event in the queue	v1.0
:OPTions?		Returns comma separated 3-digit values to represent the hardware option(s) or features available with a particular Downconverter model	v1.0
:VERSion?		Return the SCPI compliance version	v1.0

References

1. “Standard Commands for Programmable Instruments (SCPI)”, SCPI Consortium, May 1999, version 1999.0, http://www.spiconsortium.org
2. "IVI-6.1: IVI High-Speed LAN Instrument Protocol", IVI Foundation, 24 February 2011, Version 1.1, http://www.ivifoundation.org
3. "IEEE Standard Codes, Formats, Protocols, and Common Commands", ANSI/IEEE Standard 488.2-1992, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&isnumber=5581&arnumber=213762&punumber=2839

Document Revision History

This section summarizes document revision history.

Document Version[1]	Release Date	Revisions and Notes
v1.0	TBD	First release

[1] Document Version is not the same as the firmware Release Version as mentioned in Appendix E: SCPI Commands Quick Reference.