SpectralBER System (2.5 Gb/s and below)

Remote Control Manual (Part No J4221-90053)

Where to Find it - Online and Printed Information:

System installation (hardware/software)	Installation & System Reference Manual
	VXIbus Configuration Guide*
Module configuration/control	Individual Module Manuals
SCPI programming	This Manua
	Installation & System Reference Manual
SCPI example programs	This Manual
	Installation & System Reference Manual
SCPI command reference	This Manual
Register-Based Programming	This Manual
VXI programming	HP SpectralBER Online Help
VXI example programs	HP SpectralBER Online Help
VXI function reference	HP SpectralBER Online Help
Soft Front Panel information	Installation & System Reference Manual
	Individual Module Manuals
VISA language information	HP VISA User's Guide

**Supplied with HP Command Modules , Embedded Controllers, and VXLink.*

Contents

Chapter 1
Common Commands

SCPI Command Format

Instrument functions such as making measurements, retrieving data, and querying status, are performed by stringing together SCPI “nodes” into commands. The SCPI commands are based on a hierarchical structure called a “subsystem” that comprises a top level “root” node and one or more lower-level nodes and their parameters as follows:

:SOURce4:DATA:TELecom:ERRor:TYPE PATTern or CVS or CVL
:SOURce4 is the root node
:DATA is a second level node
:TELecom is a third level node
:ERRor is a forth level node
:TYPE is a fifth level node

PATTern, CVS and CVL are parameters of the fifth-level :TYPE node.

Command Syntax

Commands are shown as a mix of upper and lowercase characters.

Commands can be abbreviated for shorter program line lengths. The uppercase characters define the abbreviated form of the command.

Commands are formed by linking the root node with lower-level nodes. A colon (:) is used to link nodes.

If the command requires a parameter, a space must separate the lowest level node and the parameter. If there is more than one parameter, a comma (,) is used to separate the parameters.

An example of typical commands and their abbreviated form are shown below :

:SOURce:DATA:TELecom:PAYLoad:PATTern PRBS15	full form
:SOUR:DATA:TEL:PAYL:PATT PRBS23	abbreviated form
:OUTPut2:TELecom:FORMat SONET	full form
:OUTP2:TEL:FORM SONET	abbreviated form

SCPI Long Form Command Headers

The general rule for SCPI long form command headers that are greater than four characters in length is as follows:

- Abbreviated short form mnemonics - the first four characters from the long form command header are used unless the fourth character is a vowel. In such cases, the vowel is dropped and only the first three characters are used.
- If the command is four characters long then all four characters are used, irrespective of whether the fourth character is a vowel or not.

Linking Command Statements

Command statements can be linked using semicolons (;). For example:

```
:SENSe2:DATA:TELecom:PAYLoad:PATTern PRBS23;:SOURce2:DATA:TELecom: PAYLoad:PATTern PRBS23
```

Parameters

In this manual, parameters are shown in angled brackets < >. There are five parameter types used in commands as listed in Table 1-1.

Table 1-1. Parameter Types

Parameter Types	Description
<Numeric>	All commonly used decimal numbers including optional signs, decimal points, and scientific notation. Examples are 123, 123E2, -123, -1.23E2, .123, .123E2 and 1.2300E-01. Special cases include MINimum and MAXimum. A numeric parameter can also be specified in hex, octal, and/or binary. Examples are #H7B, #Q173 and #B11110111.
<Boolean>	A single binary condition that is either true or false. Examples are ON, OFF, 1 and 0.
<Discrete>	Values that are represented by a string of alphanumeric characters. Examples are INTernal and EXTernal.
<String>	Any set of ASCII characters enclosed within single quotes or double quotes. Examples are '1111111111111111' and “0000000000000000”.
<Block>	Used to transfer large quantities of related data. Blocks can be sent as **definite length blocks** (#<numeric><numeric>) or **indefinite length blocks** (#0).

Remote Control Commands

The remote control commands in this manual have been grouped into Sections that relate to instrument functions. The SpectralBER Virtual Instrument consists of a VXI Commander module and up to five VXI servant modules. There are currently two types of servant module - Transmitters & Receivers. The instrument can have any mix of these up to the limit of 5 modules.

The modular nature of the instrument is reflected in the structure of the SCPI command set. Two “supersystems” are defined one for Transmitter modules and one for Receivers. An instrument has one instance of the appropriate “supersystem” for each Transmitter and Receiver module equipped.

Table 1-2. SCPI Supersystems

Instrument Functions	**Subsystem**
To control Transmitter Modules	:TMODule<m>
To control Receiver Modules	:RMODule<m>

The parameter <m> indicates the VXI module to which a SCPI command is addressed. These designators are allocated to Modules depending on the VXI Logical Address of the Module using the following scheme.

The SpectralBER Transmitter module with the lowest Logical Address is allocated m=1. Subsequent numbers are allocated to Transmitter modules in order of ascending Logical address. The same scheme is used for Receiver modules.

The Transmitter & Receiver “supersystems” each have subsystems within them. Each of these is replicated 4 times reflecting the fact that there are 4 channels on each card.

For Transmitters, these subsystems are as follows:

Table 1-3. Transmitter SCPI Subsystems

Instrument Functions	**Subsystem**
To control SIGNAL OUT ports	:OUTPut
To control the transmitter	:SOURce

For Receiver modules the subsystems are:

Table 1-4. Receiver SCPI Subsystems

Instrument Functions	**Subsystem**
To control SIGNAL IN ports	:INPut
To control the receiver, and get results	:SENSe
To obtain results directly accumulated by the instrument.	:FETCh

As well as the supersystems & subsystems listed above, the instrument has other subsystems which are common to all modules in the system. These are:

Table 1-5. Common SCPI Subsystems

Instrument Functions	Subsystem
To control the instrument OTHER functions.	:SYSTem
To control Status Reporting.	:STATus
VXI Adman. functions	:VXI
Instrument Heartbeat Control	:TRIGger2
Synchronous Command Pulse Control	:TRIGger3
VXI Trigger Line Control	:OUTput5
VXI Trigger Line Parametrics	:SOURce5

Instrument Options

As mentioned in “Remote Control Commands” on page 9, the SpectralBER product currently consists of 3 broad types of module:

- SpectralBER Commander
- SpectralBER Transmitters
- SpectralBER Receivers.

SpectralBER is a “virtual instrument” in VXI terms, consisting of one Message Based Commander and up to 5 Register based servants. The servants can be any mix of Transmitter or Receiver modules.

To create the VXI instrument, the normal VXI rules as regards Logical Address settings must be observed. Each Transmitter and Receiver servant must have its Logical Address set such that it is unique within the VXI rack and lies within the Servant Area setting of the SpectralBER commander. The Logical Addresses of the Servant modules are set by switches on the modules. The Logical Address of the Commander and its Servant Area are also set by switches on that module.

The Logical Address setting of the Servant modules determines which SCPI supersystem (as defined in Table 1-2, “SCPI Supersystems,” on page 9) will control which module. The Transmitter module with the lowest Logical Address will be controlled by the :TMOD1 system. The :TMOD2 system will control the module with the next lowest logical address and so on. The Receiver systems are allocated in the same way with :RMOD1 controlling the Receiver module with the lowest Logical Address and each subsequent Receiver being allocated in order of ascending Logical Address.

A typical configuration would be:

VXI SLOT	Module Type	Logical Address	Servant Area
0	Slot 0 Controller	9	255
1	SpectralBER Commander	16	8
2/3	SpectralBER Transmitter	17	N/A
4/5	SpectralBER Transmitter	18	N/A
6/7	SpectralBER Receiver	19	N/A
8/9	SpectralBER Receiver	20	N/A
10/11	SpectralBER Receiver	21	N/A

In the above configuration the SpectralBER servant modules are allocated SCPI Supersystems thus:

Module	Logical Address	SCPI System
SpectralBER Transmitter	17	:TMOD1
SpectralBER Transmitter	18	:TMOD2
SpectralBER Receiver	19	:RMOD1
SpectralBER Receiver	20	:RMOD2
SpectralBER Receiver	21	:RMOD3

Note that it is not necessary to equip the modules in the VXI rack in order of ascending Logical Address as in the above example. However it is recommended that this be done in order that the SCPI commands used to control a module more closely reflect its physical position in the rack.

Commands sent to SCPI supersystems which are not equipped (e.g. :RMOD4 or :TMOD3) will return a “Hardware Missing” SCPI error.

Thus the system shown above will be addressed by SCPI commands in the following way:

To set up channel 3 of the Transmitter module in Slots 2/3:
:OUTPUT 70902; :TMOD1:SOUR3:DATA:TEL:PAYL:PATT PRBS23

The 5 digit GPIB address is determined by:

- 1st digit (7) - GPIB select code on Controlling Computer
- 2nd & 3rd digits (09) - GPIB address of the Slot 0 Controller
- 3rd & 4th digits (02) - Secondary GPIB address of SpectralBER Commander. This is the modules Logical Address divided by 8.

To setup channel 4 of the Receiver module in Slots 10/11:
:OUTPUT 70902;:RMOD3:SENS4:DATA:TEL:PAYL:PATT PRBS23

SYSTem Subsystem

:SYSTem:CNUMber?

Returns the number of servant modules that are allocated to the SpectralBER commander module. The servant modules in the case of SpectralBER will be SpectralBER Transmitter and Receiver modules.

Returns :	<numeric>

:STem:CTYPe? <module>

<module> =	<numeric>	0 to n

This command returns a string indicating the type of servant module allocated to the SpectralBER commander by the Slot 0 Controller.

The maximum value for <module> is that which is returned by the SYSTem:CNUMber? command described above.

Possible responses are:

Returns :	“HEWLETT_PACKARD, J4225A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4226A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4227A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4230A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4231A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4232A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4233A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4234A,<Opt1>,<Opt2>,<Opt3>,<Opt4>
Returns :	“HEWLETT_PACKARD, J4235A,<Opt1>,<Opt2>,<Opt3>,<Opt4>

The response when the <module> parameter is 0 is always:

Returns :	“UNKNOWN,UNKNOWN,0,0,0,0

Note that the value for <module> used in the above command reflects the order in which the Slot 0 commander has allocated devices to the SpectralBER controller and is, if fact, the position of the modules’ entry in the Commanders Servant List. This number has no relationship to the module’s logical address or position in the VXI rack. The numbers used to identify modules in the module specific commands have no relationship with this number.

The values returned in <Opt1-4> reflect the options equipped in the corresponding module. These can take the following values:

<Optn>	**Option Equipped**
0	No Option
001	High Power Laser (Tx)
200	1.2 Gb/s Operation

:SYSTem:CDEScription? <module>

<module>	<numeric>	0 to n

This command returns a textual description of the specified servant module allocated to the SpectralBER commander by the Slot 0 Controller.

The maximum value for <module> is that which is returned by the SYSTem:CNUMber? command described above.

Possible responses are:

Returns :	“Short Reach Receiver”
Returns :	“Long Reach Receiver”
Returns :	“Long Reach Transmitter 1550 nm
Returns :	“Long Range Transmitter 1330 nm
Returns :	“ITU Transmitter 1550 nm"
Returns :	"Multi rate Short Reach Receiver"
Returns :	"Multi rate Long Reach Transmitter 1310 nm"
Returns :	"Multi rate Long Reach Transmitter 1550 nm"
Returns :	"Multi rate ITU Transmitter 1550 nm"

The response when the <module> parameter is 0 is always:

Returns :	“No Module”

Note that the value for <module> used in the above command reflects the order in which the Slot 0 commander has allocated devices to the SpectralBER controller and is, if fact, the position of the modules’ entry in the Commanders Servant List. This number has no relationship to the modules logical address or position in the VXI rack. The numbers used to identify modules in the module specific commands have no relationship with this number.

:SYSTem:CNUMber:TMODules?

This command returns the number of transmitter modules controlled by the SpectralBER commander module. The returned value will be in the range 0 to 5.

<numeric>	0 to 5

The value returned here represents the maximum value for the TMODule<m> command header.

:SYSTem:CNUMber:RMODules?

This command returns the number of receiver modules controlled by the SpectralBER commander module. The returned value will be in the range 0 to 5.

<numeric>	0 to 5

The value returned here represents the maximum value for the RMODule<m> command header.

:SYSTem:CORDinal? <logical_address>

<logical_address>	<numeric>	0 to 255

This command returns a number indicating the SCPI Supersystem which controls the module with the specified logical address. For example, if the command returns 3 for a Transmitter module at the specified logical address, this module will be controlled by the TMODule3 SCPI supersystem. If the command returns 3 for a Receiver module then it will be controlled by the :RMOD3 SCPI supersystem. See the Instrument Options section of this chapter for information on how SCPI supersystems are allocated to modules.

A list of the logical addresses of modules allocated to the Spectralber Virtual Instrument can be obtained using the VXI:CONF:DLAD? command.

:SYSTem:VERSion?

Requests the revision state of the SCPI remote control.

The revision state is returned in the form YYYY.V. YYYY signifies the year and V signifies the revision number.

Returns :	<version> =	YYYY.V

:SYSTem:ERRor?

Requests the SpectralBER remote control Error status.

The error status is returned as a numeric value and a string containing a description of the error.

Returns :	<numeric>,<string>

IEEE Common Capabilities

***CLS**

Clear Status - Clears all status registers and the error queue.

***ESE <numeric>**

Event Status Enable - Sets the mask of the Event Status Register.

<numeric>,<string>	1	Operation Complete
	2	Request Control
	4	Query Error
	8	Device Dependent Error
	16	Execution Error
	32	Command Error
	64	User Request
	128	Power On

***ESE?**

Event Status Enable Query - Returns the current mask setting.

***ESR?**

Event Status Register Query - Returns the state of the Event Status Register in numeric form.

***IDN?**

Identification Query - Returns the Manufacture Name, Model Number & Name, Serial Number, Firmware Revision Number as a string: "HEWLETT-PACKARD, J4223A, GBnnnnnnnn, A.nn.nn" GB signifies the country of origin (Great Britain).

***LRN?**

Learn Query - Returns the instrument settings configuration in <#0 Block> form.

***OPC**

Operation Complete - Masks the OPC bit in the Event Status Register.

***OPC?**

Operation Complete Query - Returns a 1 when the OPC bit in the Event Status Register is set to 1 (true).

***OPT?**

Option Identification Query - Returns the Option and Plug-in state of the instrument. For SpectralBER, a number will be returned for each module fitted in the virtual instrument. The numbers will indicate the type of modules fitted. Up to 5 numbers may be returned. If no modules are fitted, 0 is returned.

Option/Plug-In Fitted	**Returned Result**
No Modules Fitted	0
J4225A - Short Reach Receiver	001
J4226A - Long Reach Receiver	002
J4230A - 1310nm Short Reach Transmitter	003
J4231A - 1550nm High Performance Transmitter	004
J4231A #001 - 1550nm High Power Transmitter	005
J4232A 1550nm ITU Transmitter	006
J4227A Multi rate Short Reach Receiver	007
J4233A Multi rate 1310nm Short Reach Transmitter	008
J4234A Multi rate 1550nm Transmitter	009
J4235A Multi rate 1550nm ITU Transmitter	010

The list is returned with each list item separated by a comma:

***PSC <numeric>**

Sets the value of the Power On Status Clear flag. Controls the automatic clearing of SRQ Enable register, Standard ESR & Parallel Poll Enable register after power on. See *IEE 488.2 Section 10.25.*

***PSC?**

Returns the status of the PSC flag.

***RST**

Set the instrument to the default settings listed in “Default Settings” on page 29.

***SRE <numeric>**

Service Request Enable - Sets the status byte mask.

<numeric> =	8	QUES Status Summary
	16	Message Available
	32	Event Status Summary
	64	Request Service
	128	OPER Status Summary

***SRE?**

Service Request Enable Query - Returns the current mask setting in numeric form.

***STB?**

Status Byte Query - Returns the value of the status byte in numeric form.

VXI Subsystem

This subsystem contains commands which control the administration functions associated with operation a VXI-based system. It permits register level access to VXI register-based servants and can be used for either debug or for control of non-supported modules.

:VXI:CONF:DNUM?

This command returns the total number of devices that are allocated to the module, including the SpectralBER DWDM controller itself. This constitutes the SpectralBER logical instrument.

<numeric>	0 to 256

:VXI:CONF:DLAD?

This command returns a numeric list of device logical addresses for the SpectralBER controller and all its Transmitter & Receiver servants as allocated by the Resource Manager. The number of entries in the returned list is given by the response to the VXI:CONF:DNUM? command.

<numeric>	0 to 255

:VXI:READ? <logical_addr>,<reg_addr>

This command allows access to the entire 64-byte A16 register address space for the device specified by <logical_addr>, to read the data from that address. Since the VXIbus system is byte-addressed, while the registers are 16-bits wide, registers are specified by even addresses only. This method of identifying registers follows the VXIbus format.

<logical_addr>	<numeric>	0 to 255
<reg_addr>	<numeric>	0 to 62

Returned data is of the form:

<numeric>	-32768 to 32768

:VXI:WRITe <logical_addr>,<reg_addr>,<value>

This command allows access to the entire 64-byte A16 register address space for the device specified by <logical_addr> to write data into that address. Since the VXI system is byte-addressed, while registers are 16 bits wide, registers are specified by even address only. This method of identifying registers follows the VXIbus standard format.

<logical_addr>	<numeric>	0 to 255
<reg_addr>	<numeric>	0 to 62
<value>	<numeric>	-32768 to 32768

:VXI:RES <logical_address>

This command performs a soft reset of the device at <logical_address>

<logical_addr>	<numeric>	0 to 255

This command first sets the “sysfail” inhibit bit in the device’s control register, then sets the reset bit, waits 100uS then clears reset followed by “sysfail” inhibit.

Instrument Measurement Timing

This section describes the SpectralBER measurement timing system.

Measurements in the SpectralBER virtual instrument are normally controlled by the DWDM Controller. To do this the controller transmits two signals on the VXI backplane to the servant modules - A 100ms Measurement "Heartbeat" signal and a Synchronous Command pulse. These signals can be transmitted on any one of 4 pairs of signal lines on the VXI backplane.

The Heartbeat signal is train of pulses occurring every 100ms and is used to control the update of measurement results. The Synchronous Command pulse is a single pulse which causes some pre-defined action to occur in the modules which receive it. It is used to start and stop measurement periods.

When the SpectralBER virtual instrument is the only instrument in a VXI rack, you will rarely need to exercise control over these signals. The instrument power-on default settings ensure that the signals are enabled correctly and measurements can be started and stopped using the command sequences described at the end of this section.

However, when a SpectralBER virtual instrument co-exists with another instrument in the same VXI rack, you will need to make sure that the system is set up according to you measurement requirements. For example, if two or more instruments are to make measurements independently you need to ensure that they are not using the same lines on the VXI backplane. If you need to make co-coordinated measurements, you must ensure that the instruments are using the same VXI signals but that these are sourced from only one of them.

The Heartbeat and Synchronous Command Pulse signals are controlled by two subsystems -

TRIGger2 - Controls the 100ms Heartbeat signal

TRIGger3 - Controls the Synchronous Command pulse signal.

These signal are described in the following sections.

TRIGger2 Subsystem

The TRIGger2 subsystem controls the timing of the SpectralBER measurement updates. This is done from a 10Hz “heartbeat” signal supplied on one of the VXI TTL Trigger lines. In SpectralBER this signal is normally sourced from the DWDM Controller. However, the commands in this section operate in the same way if the pulse is supplied from another instrument in the VXI rack.

Three commands are available.

- A command to continuously initiate the system
- A command to select the TTL line which sources the heartbeat
- A command to return the system to an idle state

These are described below:

:INITiate2:CONTinuous <state>

<state>	<discrete>	0 or OFF
	<discrete>	1 or ON

Sending this command with the ON parameter causes the SpectralBER measurement system to be continuously initiated. Hence all measurements will be continuously updated on receipt of a 10Hz heartbeat signal.

The default state for this subsystem is OFF. which means that the system is idle. When the INIT2:CONT ON command is sent, the trigger system is initiated and enters the “wait for trigger” state. The trigger in this case is the 10Hz heartbeat. On completion of a trigger cycle, the system immediately enters the “wait for trigger” state again without first returning to the idle state.

:INITiate2:CONTinuous?

The query version of the command returns the presently configured state of the subsystem

Returns :	<discrete> =	0 or 1

:ABORt2

This command resets the TRIGger2 subsystem to the idle state.

In this state the module will not respond to any heartbeat signals and measurement system updates will not take place. Note, that the effect of the command depends on the state of the subsystem as configured by the previous INITiate2:CONTinuous command. If the subsystem has been placed in the Continuous Triggering state by INITiate2:CONTinuous ON, then on receiving the ABORT2 command the system will enter the idle state and immediately exit it and go back to the wait for trigger state. If the subsystem has been exited from the Continuous Triggering state by a INITiate2:CONTinuous OFF command, the ABORt2 command causes the subsystem to return to the idle state and stay there.

This means that to ensure the TRIGger2 sub-system in the idle state, you must send two commands. An INITiate2:CONTinous OFF followed by an ABORt2.

:TRIGger2:SOUR <source>

This command selects the VXI signal line pair on which the DWDM Controller expects to find the Heartbeat and Synchronous Command pulses which control TRIGger2 and TRIGer3 subsystems respectively. These signals together provide control of the SpectralBER measurement system. The Heartbeat signal controls measurement updates while the Synchronous Command Pulse controls the periods over which measurements are made. The signal for the TRIGger2 subsystem is a 10Hz clock signal while the signal for the TRIGger3 subsystem is a pulse.

In SpectralBER these signals are normally sourced by the DWDM Controller module and distributed to the other SpectralBER modules (and possibly to other VXI instruments) on the selected VXI TTL trigger lines. The 10Hz Heartbeat signal is sent on the even-numbered signal of the pair while the sync pulse is distributed on the odd-numbered signal.

In a system where the DWDM Controller does not source these signals, you would use this command to select the pair on which the Controller expects to receive them.

Note that this command also selects the VXI pair on which the SpectralBER servant modules expect to receive the Heartbeat and Synchronous Command pulse signals. These modules do not have the capability to source these signals.

This command selects which pair of lines are used.

<source> =	TTLTrg0	TTLT0 - Heartbeat TTLT1 - Sync Pulse
	TTLTrg2	TTLT2 - Heartbeat TTLT3 - Sync Pulse
	TTLTrg4	TTLT4 - Heartbeat TTLT5 - Sync Pulse
	TTLTrg6	TTLT6 - Heartbeat TTLT7 - Sync Pulse

Both trigger systems must be in the idle state for this command to work. This can be done by sending the following commands :

- INIT2:CONT OFF
- INIT3:CONT OFF
- ABOR2
- ABOR3

:TRIGger2:SOURce?

The query version of the command returns the currently selected pair in the short-form shown above.

TRIGger3 Subsystem

This subsystem controls the Synchronous Command Pulse function. The purpose of this is to provide a means of synchronizing the execution of commands across several VXI modules allowing the execution of those commands to be precisely controlled by eliminating the effects of command transmission and parsing delays inherent in the normal system.

For example, to synchronize a measurement across several instruments you would first set them all up to operate from the same VXI trigger pair. Next you select one of the instruments to be the, source for the Heartbeat and Synchronous Command pulse signals. Now, you issue a START command to each of the other instruments. At this point, none of the instruments has begun a measurement but they are all primed to do so when they receive a Synchronous Command Pulse. Finally, you issue a Synchronous Command pulse from the source instrument and all the instruments which you have primed begin to measure.

Five commands are available to control this subsystem. These are

- A command to continuously initiate the system
- A command to query the TTL line configured for the Synchronous Command Pulse.
- A command to return the system to an idle state
- A command to select the Synchronous Command to be executed
- A command to cause the subsystem to trigger independently of the Synchronous Command Pulse

These are described fully below.

:INITiate3:CONTinuous <state>

<state>	<discrete>	0 or OFF
	<discrete>	1 or ON

Sending this command with the ON parameter causes the Synchronous Command system to be continuously initiated. This causes the SpectralBER measurement system to react to all external synchronous command pulses or immediate trigger commands.

The default value for the subsystem is OFF. and in this state the system is idle. When the INIT3:CONT ON command is sent, the trigger system is initiated and enters the “wait for trigger” state. On completion of a trigger cycle, the system immediately enters the “wait for trigger” state again without first returning to the idle state.

:INITiate3:CONTinuous?

The query version of the command returns the presently configured state of the subsystem

Returns :	<discrete> =	0 or 1

:TRIGger3:COMMand <command>

This command selects the Synchronous Command to be executed when the sub-system receives its trigger. Commands are:

<command> =	<discrete>	STARt
		STOP
		ONCE
		CONTinue

Commands are executed on the heartbeat pulse (TRIGger2) immediately following the Synchronous Command Pulse (TRIGger3) or a TRIG3:IMM command.

A selected command is active and will be executed on each command pulse or trigger immediate command until replaced by another command.

The default command is STOP.

The STARt command resets all measurement counters to zero and starts continuous measurement updates controlled by the Heartbeat signal. The EIPER status bit in the OPERation status register is also updated.

The measurement period is determined by the time between Synchronous Command Pulses on the selected VXI trigger line or TRIG3:IMM commands. These events cause a re-triggering of the STARt command and hence the beginning of a new measurement period. There is no “dead time” between measurement periods.

The STOP command stops the measurement system and freezes all measurement counters.

The ONCE command causes the system to enter a transient state during which one measurement period occurs. On the first Synchronous Command Pulse or TRIG3:IMM after this command has been set by the TRIG3:COMM command a measurement period is started. At this point, the ONCE command is changed to a STOP command by the system. Hence on the next Synchronous Command Pulse or TRIG3:IMM the measurement period is stopped. Note that the EIPER status bit in the OPERation status register is not updated until another measurement period has begun. Also note that after the ONCE command is issued, it cannot be replaced by another command. Only a Synchronous Command Pulse or a TRIG3:IMM command can change it.

The CONT command resumes a measurement from where it was stopped retaining the values in the current measurement store.

Note Use :RMODule<m>:SENSe<n>:GATE:COMMand <action> on page 35 when controlling a single channel.

:TRIG3:COMMand?

The query version of the command returns the currently selected command in the short-form shown above.

Returns :	<Command>

:TRIGger3:IMMediate

This command causes the TRIGger3 sub-system to trigger immediately. It behaves as if a Synchronous Command Pulse had been received on the appropriate VXI TTL trigger line. This overrides normal operation where the subsystem waits for a Synchronous Command Pulse before triggering. Note that the TRIGger3 Subsystem must be in a “wait for trigger” state for this command to have an effect.

Note Use :RMODule<m>:SENSe<n>:GATE:IMMediate on page 36 when controlling a single channel.

:TRIGger3:SOURce?

This command returns the identity of the VXI TTL line which has been selected a the trigger source for the TRIgger3 subsystem. This is selected by the TRIGger2:SOURce command along with the TRIGger2 subsystem source. The TRIGger3 subsystem source is always an odd-numbered line one higher than that selected for the TRIGger2 subsystem.

Returns :	<discrete>	TTLT1
		TTLT3
		TTLT5
		TTLT7

OUTput5 Subsystem

This subsystem is used to control the VXI trigger lines. The SpectralBER measurement system uses a “heartbeat” signal (10Hz) and Synchronous Command pulses to synchronize measurement periods across Receiver modules. These signals are implemented as a pair of VXI TTL trigger lines and can be generated on the selected pair by the SpectralBER commander. Alternatively, these signals can be sourced from other VXI modules which have the capability of generating them. A further possibility is to source them from a Slot0 controller which has the capability to route signals from its TRIGGER IN port onto the VXI trigger bus and to route signals from that bus onto its TRIGGER OUT port. Using this method these signals can be transmitted between VXI racks and hence allow co-ordination of measurements between racks. Note however that this scheme is limited by the fact that the Slot0 controllers have only one input and one output port. Hence, two racks can be coordinated by “swapping” heartbeat and Synchronous Command Pulse signals but a scheme involving more than two racks can share only one signal.

The Heartbeat is a 10Hz square wave signal which is injected onto an even-numbered trigger line (TTLT0,2,4,6) under the control of the OUTP5:TTLT<n> command. The Synchronous Command Pulse is injected onto an odd-numbered trigger line (TTLT1,3,5,7) under the control of either the SOURce5 subsystem or the OUTP5:TTLT<n>:IMM command. The TRIG2:SOUR command select which pair of trigger lines will carry the Heartbeat and Synchronous Command Pulse.

Note An OUTP5:TTLT1:IMM or TRIG3:IMM command is required to start the generation of Synchronous Command Pulses after they are enabled and after the execution of a STOP Synchronous Command Action. Synchronous Command Pulses should be disabled when controlling a single channel.

:OUTP5:TTLTrigger<n>:STATe <state>

This command either enables or disables the generation of Heartbeat and Synchronous Command Pulse from the SpectralBER controller on the designated VXI Trigger lines.

<n> =	<numeric>	0 to 7
<state>	<numeric>	0 - Disable
	<numeric>	1 - Enable

The command will work only for that pair of lines previously selected by the TRIG2:SOURce command. Each line in the pair must be enabled or disabled by a separate instance of this command.

If a TRIG2:SOUR command is issued to change the selected pair while they are enabled then the lines are forced to a disabled state.

Note that for SpectralBER, the power-on default state of the Heartbeat signal is enabled. This means that the instrument monitors alarm conditions in it’s power-on default state without you having to perform any set-up. The default selection for the Synchronous Command pulse is disabled.

:OUTP5:TTLTrigger<n>:STATe?

The query version of the command returns the enabled/disabled state of the selected trigger line.

Returns :	<numeric>	0 - Disabled
	<numeric>	1 - Enabled

:OUTP5:TTLTrigger<n>:IMM

<n> =	<numeric>	1
	<numeric>	3
	<numeric>	5
	<numeric>	7

This command causes a Synchronous Command Pulse to be issued on the selected VXI Trigger line. This feature allows the SpectralBER commander to synchronize command execution on other modules.

The selected VXI Trigger line must be odd-numbered and must have been previously enabled by the OUTP5:TTLTrigger<n> command.

Note Use :RMODule<m>:SENSe<n>:GATE:IMMediate on page 36 when controlling a single channel.

SOURce5 Subsystem

The SOURce5 subsystem is used to configure the VXI trigger lines selected by the TRIG2:SOUR command to produce a specific measurement period.

:SOURce5:PULSe1:PERiod?

This command returns the time interval between the heartbeats in seconds. The response is always 0.1 seconds

Returns :	<numeric>	1.000E-001

:SOURce5:PULSe2:PERiod <period>

This command selects the period in seconds over which the measurement system will accumulate measurements, that is, the measurement period.

<period>	<numeric>	1 to 8640000

Once set up by this command, the measurement system will produce a Synchronous Command Pulse at the end of each measurement period on the previously selected (and enabled) TTL Trigger line. To be effective, the configured Synchronous Command must be STARt.

After enabling the OUTPut5 subsystem, any change in period made using this command will not take effect until after the next Synchronous Command Pulse.

The default period is 1 second.

The query version of this command returns the currently selected period.

Note Use :RMODule<m>:SENSe<n>:GATE:IMMediate on page 36 when controlling a single channel.

:SOURce5:PULSe2:PERiod?

Returns :	<numeric>	1 to 8640000

Measurement Start and Stop Sequences

This section describes the sequence of commands which you can use to start and stop group gating measurements. For single channel gating measurements, refer to Chapter 5, Example Programs using SCPI.

Start Gating

The following sequence can be used to begin a 100 second timed measurement. This sequence will work from the power-on default settings in SpectralBER since VXI lines TTLT0 & TTLT1 will be selected as the sources for the TRIGger2 & TRIGger3 subsystems.

This assumes that the SpectralBER instrument is operating independently from any other VXI instruments which may be in the rack.

The instrument will gate for 100 seconds and then stop without any further commands being issued. This uses the ONCE command as described above in the TRIGger3 sub-system.

1. INIT2:CONT ON
2. INIT3:CONT ON
3. SOUR5:PULS:PER 100
4. OUTP5:TTLT0:STAT 1
5. OUTP5:TTLT1:STAT 1
6. TRIG3:COMM ONCE
7. OUTP5:TTLT1:IMM

Commands 1 & 2 put the TRIGger2 and TRIGer3 subsystems into continuous trigger mode. Command 3 sets the period for the Synchronous Command pulse.

Command 4 enables the DWDM Controller to supply Synchronous Command pulses on TTLT0 and Command 5 enables it to transmit the 100ms Heartbeat signal on TTLT1.

Command 6 primes the TRIgger3 sub-system to execute a ONCE command when it gets the next Synchronous Command pulse. Command 7 issues the Synchronous Command pulse and the instrument begins to gate.

Stop Gating

This sequence of commands will stop a measurement as started in the sequence given above.

1. TRIG3:COMM STOP
2. OUTP5:TTLT1:IMM
3. INIT2:CONT OFF
4. INIT3:CONT OFF
5. ABOR2
6. ABOR3

Command 1 primes the TRIGger3 sub-system to execute a STOP command when it gets the next Synchronous Command pulse. Command 2 issues the Synchronous Command pulse and the instrument stops gating.

Command 3 though 6 return the TRIGger2 and TRIGger3 subsystems to their idle states.

Default Settings

The default settings and associated SCPI commands are listed in the following table

Commands	Default
TMOD<m>:OUTP<n>:STAT	ON
TMOD<m>:OUTP<n>:TEL:FORM	SONet
TMOD<m>:SOUR<n>:DATA:TEL:PAYL:PATT	PRBS23
TMOD<m>:SOUR<n>:DATA:TEL:TOH:J0:DATA	" MOD#m CHAN#n "
TMOD<m>:SOUR<n>:DATA:TEL:ERR:TYPE	PATTern
TMOD<m>:SOUR<n>:DATA:TEL:ERR:RATE	OFF
RMOD<m>:SENS<n>:DATA:TEL:PAYL:PATT	PRBS23

Note The instrument can be configured to its default settings by using the SCPI SYSTem:PRESet command or the *RST IEE 488.2 command.

Chapter 2
Receiver Module Commands

RMODule<m> System - Receiver Commands

This chapter describes the commands used to set up the SpectralBER Receiver modules. Up to 5 SpectralBER Receiver modules can be controlled by the SpectralBER Commander module. Each module has its own RMODule system identified by the <m> value in the RMODule SCPI command header.

Within the RMODule system, there are three further sub-systems:-

SENS<n>

FETch<n>

INPut<n>

Each of these sub-systems concerns one receiver channel. Within a SpectralBER receiver module there are 4 such channels and each channel has its own sub-system identified by the <n> value in the headers shown above.

Note that it is also possible to set up several receivers with one command. For all the set type commands in this chapter, the RMODule<m> header can be replaced by RMODuleALL. In this case the subsequent part of the command is applied to all equipped Receiver Modules. For example, a command whose header begins with:

:RMODuleALL:SENSe1:

will set up Channel 1 in all equipped Receiver modules.

This also applies within Receiver modules where, for example, the SENSe<n> node can be replaced with SENSeALL. For example a command whose header begins with:

:RMODule3:SENSeALL:

would set up all Channels on Receiver module 3.

These can also be combined so that a command beginning with:

:RMODuleALL:SENSeALL:

would set up all channels on all equipped Receivers.

Note This does not apply to the query versions of commands.

Module Identification

This command lets you determine which type of receiver modules are equipped.

:RMODule<m>:TYPE?

A string is returned identifying the type of receiver module controlled by the RMODule<m> super-system.

Possible responses are:

```
<Returns> =    "HEWLETT-PACKARD,J4225A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4226A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4227A,<op1>,<op2>,<op3>,<op4>"
```

If the module type is unrecognized then "UNKNOWN" will be returned.

For Receivers, the possible option for op1 through op4 are shown below.

1.25 G Operation	200

SENSe<n> Subsystem - Receiver Settings Commands

These commands set-up the SpectralBER Receiver channels.

:RMODule<m>:SENSe<n>:DATA:TELecom:PAYLoad:PATTern <discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	PRBS23	2^{23}-1
	PRBS15	2^{15}-1
	PRBS11	2^{11}-1
	PRBS9	2^{9}-1

Selects the Receiver Payload data pattern.

This is the pattern with which the Received signal payload will be compared for the determination of Bit Errors.

The corresponding query returns the receiver SDH payload data pattern in discrete form as listed above.

:RMODule<m>:SENSe<n>:DATA:TELecom:PAYLoad:PATTern?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

SENSe<m> Subsystem - Gating Commands

:RMODule<m>:SENSe<n>:GATE:COMMand <action>

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL
<action> =	STOP STARt ONCE CONTinue FREErun

This command selects an action to take place on the heartbeat pulse (TRIGger2) immediately following one of three trigger events:

A Synchronous Command Pulse (TRIGger3).
An end of Measurement Period.
The execution of command :RMODule<m>:SENSe<n>GATE:IMMediate.

A selected action is active and will be executed after each trigger event until replaced by another action.

The STOP action stops measurement updates on a port and freezes all measurement counters.

The STARt action resets all measurement counters to zero and starts continuous measurement updates controlled by the Heartbeat signal. The EIPER status bit in the OPERation status register is also updated. Trigger events cause a re-triggering of the STARt action and hence the beginning of a new measurement period. There is no "dead time" between measurement periods.

The ONCE action causes one measurement period to occur. On the first trigger event, ONCE acts as a STARt action. At this point, the ONCE action is changed to a STOP action by the system. Hence on the next trigger event, all measurement counters are frozen. The EIPER status bit in the OPERation status register is not updated until another measurement period has begun.

The CONTinue action resumes updating measurements from where it was stopped.

The FREErun action starts a measurement period similar to the STARt action, but will not re-trigger. The manual execution of the STOP action is required to terminate a measurement. The Gating Period is ignored.

The default action is STOP.

Note RMODuleALL:SENSeALL:GATE:COMMand<ACTION> (<m> and <n> = ALL) is equivalent to TRIGger3:COMMand<command>, but allows single channel control.

:RMODule<m>:SENSe<n>:GATE:COMMand?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<action>

The query version returns the currently selected action.

Note "ALL" arguments do not apply to the query version of this command.

:RMODule<m>:SENSe<n>:GATE:IMMediate

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL

This command is one of three trigger events needed to trigger the TRIGger3 sub-system and execute the command specified by: RMODule<m>:SENSe<n>:GATE:COMMand <command>. It behaves as if a Command Pulse had been received on the appropriate VXI TTL trigger line, but may be directed to individual channels. This overrides normal operation where the subsystem waits for a Command Pulse or an end of Measurement Period before triggering.

Note The TRIGger3 Subsystem must be in a "wait for trigger" state for this command to have an effect.

Note RMODuleALL:SENSeALL:GATE:IMMediate (<m> and <n> = ALL) is equivalent to: TRIGger3:IMMediate, but allows single channel control.

:RMODule<m>:SENSe<n>:GATE:PERiod <period>

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL
<period> =	1 to 8640000 (seconds)

This command selects the period in seconds over which each port will accumulate measurements, that is, the measurement period. The maximum period is 8640000 seconds = 100 days.

The default period is 1 second. A new period takes effect after the execution of RMODule<m>:SENSe<n>:GATE:COMMand<action>.

Note RMODuleALL:SENSeALL:GATE:PERiod<period> (<m> and <n> = ALL) is equivalent to: SOURce5:PULSe2:PERiod <period>, but allows single channel control.

:RMODule<m>:SENSe<n>:GATE:PERiod?

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL
Returns :	<period>

The query version returns the currently selected period in seconds.

Note "ALL" arguments do not apply to the query version of this command.

SENSe<m> Subsystem - Result Returning Commands

These commands return the SpectralBER measurement results. The results are those accumulated over a pre-defined measurement period.

:RMODule<m>:SENSe<n>:DATA? <"result">

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL
"ECOunt:PATTern"	Payload Pattern Bit Error Count
"ERATio:PATTern"	Payload Pattern Bit Error Ratio
"ECOunt:CVS"	B1 Section BIP/Section CV Error Count
"ERATio:CVS"	B1 Section BIP/Line CV Error Ratio
"ECOunt:CVL"	B2 Line BIP/Line CV Error Count
"ERATio:CVL"	Line B2 BIP/Line CV Error Ratio
"ETIMe:CUMulative"	Elapsed measurement time in seconds

Note that results can be read from several receivers with one command using the ALL options for both <m> and <n>. In this case the results are returned as a comma separated list in the order shown below. The list returned always contains 20 values, one for each possible receiver channel fitted. For non-equipped receivers, NAN (not-a-number) is returned. The result fields are a constant width of 15 characters and are in the order shown below:

```
<RMOD1:CHAN1>,<RMOD1:CHAN2>,<RMOD1:CHAN3>,<RMOD1:CHAN4>
<RMOD2:CHAN1>,<RMOD2:CHAN2>,<RMOD2:CHAN3>,<RMOD2:CHAN4>
<RMOD3:CHAN1>,<RMOD3:CHAN2>,<RMOD3:CHAN3>,<RMOD3:CHAN4>
<RMOD4:CHAN1>,<RMOD4:CHAN2>,<RMOD4:CHAN3>,<RMOD4:CHAN4>
<RMOD5:CHAN1>,<RMOD5:CHAN2>,<RMOD5:CHAN3>,<RMOD5:CHAN4>
```

Note RMODuleALL:SENSeALL:DATA? "ETIMe:CUMulative" (<m> and <n> = ALL) is equivalent to: SENSe:DATA? "ETIMe:CUMulative", except it reads the elapsed time for each of many channels.

:SENSe:DATA? "ETIMe:CUMulative"

This command returns the elapsed time in seconds for the current measurement period.

<numeric> =	Elapsed time in seconds.

Note Use RMODule<m>:SENSe<n>:DATA? "ETIMe:CUMulative" when requiring single channel control.

FETCh<n> Subsystem

The FETCh subsystem is used to retrieve data directly accumulated by the instrument.

:RMODule<m>:FETCh<n>:STRing:DATA:TELecom:J0?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<string>

The Section Overhead Trace contained in the J0 byte is returned as a 16 character ASCII character string if CRC7 is not detected, or a 15 ASCII character string if CRC7 is detected. If the string contains any non printing characters then ~ is substituted. This is a snapshot of the trace string and is captured once per second.

INPut<n> Subsystem

:RMODule<m>:INPut<n>:REACh?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete> SHORT
	<discrete> LONG

This command returns a discrete value indicating the type of receiver equipped in the indicated Receiver channel. There are two possible responses - SHORT - indicating that Short Reach optics are equipped and LONG indicating that Long Reach optics are equipped.

:RMODule<m>:INPut<n>:TELecom:FORMat<discrete>

<m> =	1 to 5 or all	
<n> =	1 to 4 or all	
<discrete> =	FRAMed	Receiver expects SONET/SDH structured signal
	UNFRamed	Receiver expects unstructured signal.

This command allows the signal structure for the selected port to be configured. The choices are FRAMed or UNFRamed. In the case of the FRAMed selection, the receiver expects a SONET/SDH structured signal. In the UNFRamed case, the receiver expects no signal structure.

Note that when the signal rate selected by the RMOD<m>:INP<n>:TEL:RATE command is M1244, the only available choice is UNFRamed.

:RMODule<m>:INPut<n>:TELecom:FORMat?

The corresponding query command will return the currently configured Signal Structure for the indicated port in the form shown in the table above.

<m> =	1 to 5 or all
<n> =	1 to 4 or all
Returns :	<discrete>

:RMODule<m>:INPut<n>:TELecom:RATE<discrete>

<m> =	1 to 5or all	
<n> =	1 to 4 or all	
<discrete> =	M2488	2.488 Gb/s Operation
	M1244	1.244 Gb/s Operation
	M622	622 Mb/s Operation
	M155	155 Mb/s Operation

This command allows the signal rate for the given port to be selected. The corresponding query command returns the currently configured signal rate for the indicated port in the form shown in the table above.

For Single Rate modules, the only valid choice is M2488.

For Multi Rate modules 1.244Gb/s operation is available only when Option200 is fitted.

Note that when 1.244 Gb/s is selected, the only signal format allowed by the RMOD<n>:INP<n>:TEL:FORMAT command is UNFRamed.

:RMODule<m>:INPut<n>:TELecom:RATE?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

Chapter 3
Transmitter Commands

TMODule<n> Subsystem - Transmitter Commands

This chapter describes the commands used to set up the SpectralBER Transmitter modules. Up to 5 SpectralBER Transmitter modules can be controlled by the SpectralBER commander module. Each module has its own TMODule super-system identified by the <m> value in the TMODule SCPI command header.

Within the TMODule super-system there are two lower level sub-systems

OUTput<n>

SOURce<n>

Each of these sub-systems sets up one Transmitter channel. Within a SpectralBER Transmitter module there are 4 such channels and each channel has its own sub-system identified by the <n> value in the headers shown above.

Note It is also possible to set up several transmitters with one command. For all the set type commands in this chapter, the TMODule<m> header can be replaced by TMODuleALL. In this case the subsequent part of the command is applied to all equipped Transmitter Modules.

For example, a command whose header begins with:

:TMODuleALL:SOURce1:

will set up Channel 1 in all equipped Transmitters modules.

This also applies within Transmitter modules where, for example, the SOURce<n> node can be replaced with SOURceALL. For example a command whose header begins with:

:TMODule3:SOURceALL:

would set up all Channels on Transmitter module 3.

These can also be combined so that a command beginning with:

:TMODuleALL:SOURceALL:

would set up all channels on all equipped Transmitters.

Note This does not apply to the query versions of commands.

Module Identification

This command lets you determine which type of transmitter modules are equipped.

:TMODule<m>:TYPE?

A string us returned identifying the type of transmitter module controlled by the TMODule<m> super-system.

Possible responses are:

```
<Returns> =    "HEWLETT-PACKARD,J4230A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4231A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4232A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4233A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4234A,<op1>,<op2>,<op3>,<op4>"
               "HEWLETT-PACKARD,J4235A,<op1>,<op2>,<op3>,<op4>"
```

If the module type is unrecognized then "UNKNOWN" will be returned.

OP1 to OP 4 can take 1 of two values, these values are listed below.

High Power Operation	001
1.25 G operation	200

SOURce<n> Subsystem - Transmitter Settings

This subsystem provides the command which set up the SpectralBER Transmitter channels. There are 4 Transmitter channels in a Transmitter module and these are identified by the <n> value in the SOURce<n> command header.

:TMOD<m>:SOURce<n>:DATA:TELecom:PAYLoad:PATTern <discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	PRBS23	2^{23}-1
	PRBS15	2^{15}-1
	PRBS11	2^{11}-1
	PRBS9	2^{9}-1

Selects the transmitted payload data pattern.

The corresponding query returns the transmitter payload data pattern in discrete form, as listed above.

:TMOD<m>:SOURce<n>:DATA:TELecom:PAYLoad:PATTern?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

:TMOD<m>:SOURce<n>:DATA:TELecom:TOH:J0:DATA <string>

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL
<string> =	

Sets the string to be transmitted in the J0 Section Trace byte of the transport overhead. The pattern should be 15 characters long. The instrument automatically appends a E.164 CRC character to make up a 16 character sequence. If less than 15 characters are input, the instrument will pad with the required number of NULL characters. The pattern repeats every 16 characters and is transmitted character by character in subsequent frames.

The corresponding query returns the value of the user defined pattern as a string, as defined above. If the string contains any non printing characters, ~ is substituted.

:TMODule<m>:SOURce<n>:DATA:TELecom:TOH:J0:DATA?

<m> =	1 to 5
<n> =	1 to 4
Returns:	<string>

:TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:TYPE <discrete>

This command selects the type of errors to be injected into the transmit signal.

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	PATTern	Payload Pattern Bit Errors
	CVS	B1 Section BIP/Section CV Errors
	CVL	B2 LINE BIP/Line CV Errors

The corresponding query returns the selected error type in discrete form as listed above.

:TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:TYPE?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

:TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:RATE <discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	OFF	Errors Off
	E_7	1x10^{-7} rate
	E_8	1x10^{-8} rate
	E_9	1x10^{-9} rate

Selects the transmitter channel Error rate of the error type selected by TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:TYPE.

The corresponding query returns the selected transmitter channel error rate in discrete form, as listed above.

:TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:RATE?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

:TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:ADD:SINGle

This command causes a single error of the type selected by TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:TYPE to be transmitted in the indicated transmitter channel.

The command is invalid and will not be accepted unless the Error Rate as selected by TMODule<m>:SOURce<n>:DATA:TELecom:ERRor:RATE is set to OFF.

<m> =	1 to 5 or ALL
<n> =	1 to 4 or ALL

OUTPut<n> Subsystem

This subsystem controls the characteristics of the output ports on a SpectralBER Transmitter module. There are 4 such ports identified by the <n> value.

:TMODule<m>:OUTPut<n>:STATus <discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	OFF	Laser Off
	ON	Laser ON

This command allows control of the laser output for transmitter channels.

The corresponding query command returns the status of the laser output for the specified channel in the form show in the table above.

:TMODule<m>:OUTPut<n>:STATus?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

:TMODule<m>:OUTPut<n>:WAVElength?

<m> =	1 to 5	
<n> =	1 to 4	
Returns :	<string>	nnnn.nn

This command returns the wavelength of laser light from the indicated output port. The string will be of the form nnnn.nn and will indicate the wavelength in manometers.

:TMODule<m>:OUTPut<n>:TELecom:FORMat <discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	SONET	SONET Structure
	SDH	SDH Structure
	UNFRamed	Unstructured

This command allows selection of the Signal Structure for the selected port. The signal can be configured as either a structured SONET or SDH signal or a completely unstructured signal. The only differences in the transmitted signal between the

SONET and SDH choices is the numbering of the Z0 bytes in the Transport Overhead and the value of the SS bits in the AU pointer.

Note that when the signal rate selected by the TMOD<m>:OUTP<n>:TEL:RATE command is M1244, the only available choice is UNFRamed.

The corresponding query command will return the currently configured Signal Structure for the indicated port in the form shown in the table above.

:TMODule<m>:OUTPut<n>:TELecom:FORMat?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

:TMODule<m>:OUTput<n>:TELecom:RATE<discrete>

<m> =	1 to 5 or ALL	
<n> =	1 to 4 or ALL	
<discrete> =	M2488	2.488 Gb/s Operation
	M1244	1.244 Gb/s Operation
	M622	622 Mb/s Operation
	M155	155 Mb/s Operation

This command allows the transmit signal rate for a given module and channel to be selected.

For Single Rate modules, the only valid choice is M2488.

For Multi Rate modules 1.244 Gb/s operation is available only when Option200 is fitted.

Note that when 1.244 Gb/s is selected, the only signal format allowed by the TMOD<n>:OUTP<n>:TEL:FORMAT command is UNFRamed.

The corresponding query command will return the currently configured signal rate for the indicated port in the form shown above.

:TMODule<m>:OUTput<n>TELecom:RATE?

<m> =	1 to 5
<n> =	1 to 4
Returns :	<discrete>

Chapter 4
Status Reporting

The status reporting capability of SpectralBER provided by the Status Registers and the Status Byte. The STATus subsystem and some IEEE common capability commands control the status registers and the status byte.

The status registers provided in SpectralBER are listed in Table 4-1 and their relationship to each other is illustrated in “Figure 4-1” on page 52.:

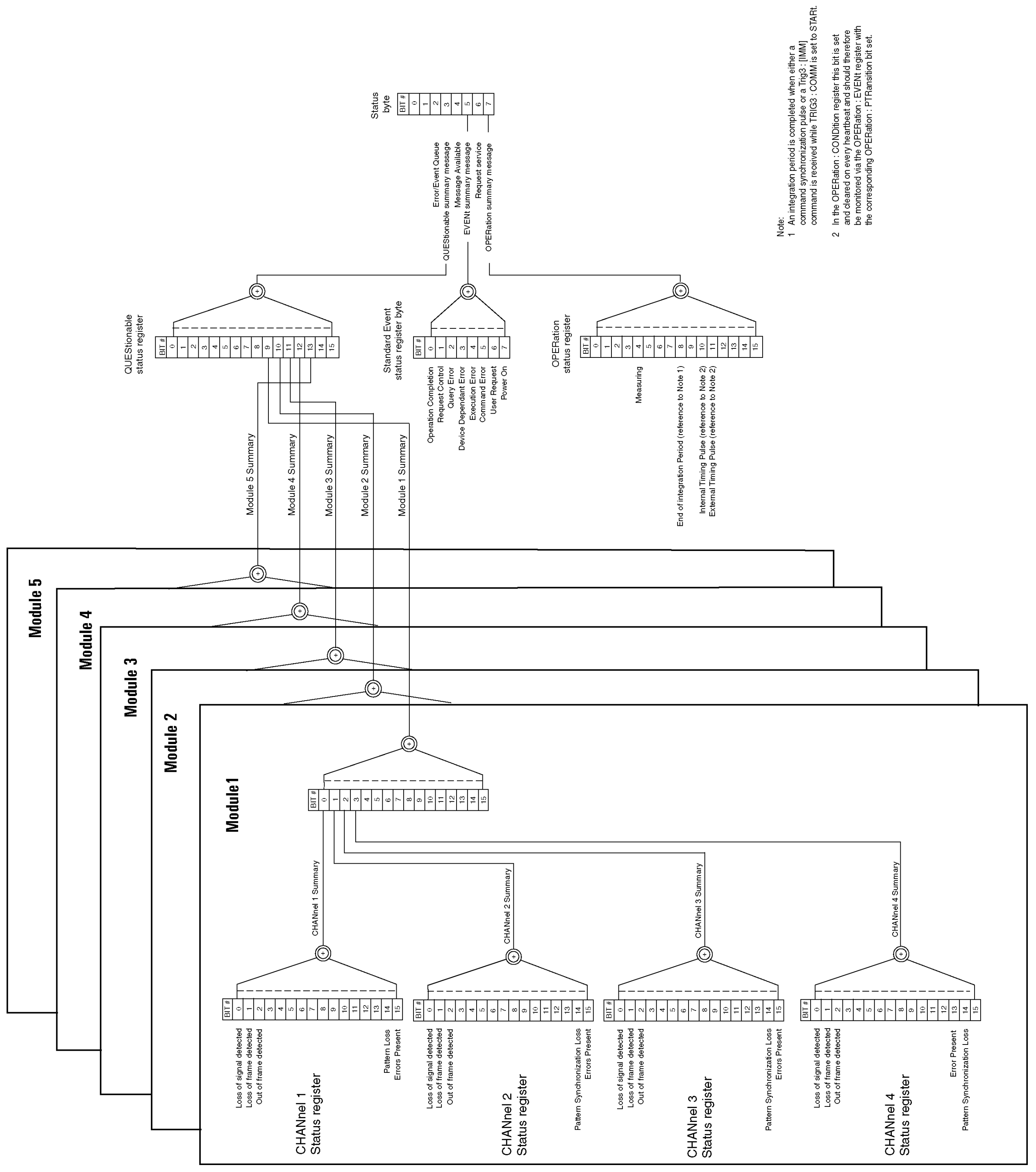
Table 4-1. Status Registers

Status Register	Description
Standard Event	This register is accessed by issuing the *ESR? common capability command.
QUEStionable	Defined by SCPI.
OPERation	Defined by SCPI.
MODule<n>	Registers which summarizes the status of the 4 channels in a Receiver module. A SpectralBER system has a register for each equipped Receiver
CHANnel<n>	Monitors the status of a the 4 individual channels of a Receiver module. A SpectralBER system has 4 such registers for each equipped Receiver Module.

General Status Register

The status registers conform to IEEE 488.2 and each comprises 4 registers as shown in Figure 4-2.

Condition Register	Monitors the defined status conditions. There is no latching of conditions in this register, it is updated in real time.
Transition Filter	Determines whether positive or negative transitions (true or false) in the Condition register sets the Event register.
Event Register	Latches the transient states that occur in the Condition register as specified by the Transition Filter.
Event Enable Register	Acts like a mask on the Event register. It determines which bits in the Event register set the summary bit in the Status Byte.



Note:
1 An integration period is completed when either a command synchronization pulse or a Trig3 : [IMM] command is received while TRIG3 : COMM is set to STARt.

2 In the OPERation : CONDition register this bit is set and cleared on every heartbeat and should therefore be monitored via the OPERation : EVENt register with the corresponding OPERation : PTRansition bit set.

Figure 4-1. Status Register Plan

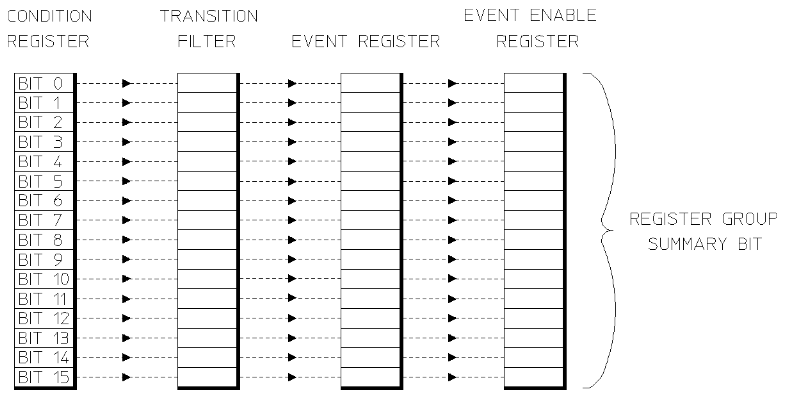


Figure 4-2. General Status Register

STATus Subsystem

This subsystem controls the status reporting registers. Standard SCPI status registers QUEStionable, and OPERation are provided and in addition instrument defined status registers MOD<m> and CHAN<n> are also provided.

Both standard SCPI and instrument defined registers, have a set of commands associated with them. These are listed below. Refer to the General Status Register model in Figure 4-1 to get a better understanding of what these commands do.

:STATus:CHIStory

Clears the contents of all History registers.

:STATus:PRESet

Presets the enable and transition filter registers to their default state.

Registers are set to:

Register	ENABLe	PTRanstion	NTRansition
OPERation	all 0’s	all 1’s	all 0’s
QUEStionable	all 0’s	all 1’s	all 0’s
All others	all 1’s	all 1’s	all 1’s

For each of the <**Status Registers**>’s listed in Table 4-1 (excluding the Standard Event Register) the following commands exist.

:STATus:<Status Register>:ENABle <numeric>

<numeric>

Sets the Event Enable register mask which allows true conditions in the Event register to be reported in the <**Status Register**>’s summary bit. If a bit is 1 in the Event Enable register and its associated event bit makes the transition to true, a positive transition will occur in the <**Status Register**>’s summary bit.

The corresponding query returns the current mask setting.

:STATus:<Status Register>:ENABle?

Returns :	<numeric>

:STATus:<Status Register>:PTRansition <numeric>

Sets the positive Transition Filter. Setting a bit in the positive Transition filter shall cause a 0 to 1 transition in the corresponding bit of the <**Status Register**>’s Condition register to cause a 1 to be written in the corresponding bit of the <**Status Register**>’s Event register.

The corresponding query returns the current setting.

:STATus:<Status Register>:PTRansition?

Returns : <numeric>

:STATus:<Status Register>:NTRansition <numeric>

Sets the negative Transition filter. Setting a bit in the negative Transition Filter shall cause a 1 to 0 transition in the corresponding bit of the <**Status Register**>'s Condition register to cause a 1 to be written in the corresponding bit of the <**Status Register**>'s Event register.

The corresponding query returns the current setting.

:STATus:<Status Register>:NTRansition?

Returns : <numeric>

:STATus:<Status Register>:EVENt?

Returns : <numeric>

Returns the contents of the Event register associated with the <**Status Register**>. Reading this register clears its contents.

:STATus:<Status Register>:CONDition?

Returns : <numeric>

Returns the contents of the Condition register associated with the <**Status Register**>. Reading this register does not clear its contents.

:STATus:<Status Register>:HISTory?

Returns : <numeric>

Returns the contents of the History register associated with the <**Status Register**>. This is in effect a latched version of the Condition register. A bit set to 1 in the Condition register will set the corresponding bit in the History register. This register is not cleared when it is read. The only time the History register is cleared is at a start of measurement period, or when the commands *RST or are sent.

Status Byte

***STB?** or a serial poll - Returns the value of the Status Byte in numeric form.

***SRE <numeric>** - Sets the Status Byte mask.

***SRE?** - Returns the current mask setting in numeric form.

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
OPER	RQS	ESR	MAV	QUES	-	-	-

DB0 - DB2	Not used, always read as 0.
DB3	QUES - QUEStionable status register summary. Indicates that a bit has been set in the QUEStionable status register.
DB4	MAV - Message Available. Remains set until all output messages are read from SpectraBER19A.
DB5	ESR - Event status register summary. Indicates that a bit has been set in the Event status register.
DB6	RQS - Request Service. Set when an SRQ is generated for whatever reason. Cleared by SPOLL or *STB?.
DB7	OPER - OPERation status register summary. Indicates that a bit has been set in the OPERation status register.

Standard Event Status Register

***ESR?** - Returns the Standard Event Status Register value in numeric form.

***ESE <numeric>** - Sets the event enable register mask.

***ESE?** - Returns the current mask setting.

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
PWR	URQ	CME	EXE	DDE	QUE	RQC	OPC

DB0	OPC - Operation Complete.
DB1	RQC - Request Control.
DB2	QUE - Query Error.
DB3	DDE - Device Dependent Error.
DB4	EXE - Execution Error.
DB5	CME - Command Error.
DB6	URQ - User Request.
DB7	PWR - Power On.

QUEStionable Status Register

Provides a summary of the 5 MODule<m> status registers. One bit is allocated for each summary.

:STATus:QUEStionable:EVENt - Returns the QUEStionable Status Register value in numeric form.

:STATus:QUEStionable:ENABle <numeric> - Sets the event enable register mask.

:STATus:QUEStionable:ENABle? - Returns the current mask setting in numeric form.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
-	-	MOD5	MOD4	MOD3	MOD2	MOD1	-

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
-	-	-	-	-	-	-	-

DB0 - DB8	Not used, always read as 0.
DB9	MOD1 - Summary of MODule1 status register.
DB10	MOD2 - Summary of MODule2 status register.
DB11	MOD3 - Summary of MODule3 status register.
DB12	MOD4 - Summary of MODule4 status register.
DB13	MOD5 - Summary of MODule4 status register.
DB14	Not used, always reads as 0.
DB15	Not used, always reads as 0.

OPERation Status Register

Provides a summary of the INSTrument status register, and reports when a measurement is being made.

:STATus:OPERation:EVENt? - Returns the OPERation Status Register value in numeric form.

:STATus:OPERation:ENABle <numeric> - Sets the event enable register mask.

:STATus:OPERation:ENABle? - Returns the current mask setting in numeric form.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
-	-	-	-	XBEAT	IBEAT	-	EIPER

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
-	-	-	MEAS	-	-	-	-

DB0 - DB3	Not used, always reads as 0.
DB4	MEAS - Measuring. Currently making a measurement.
DB5 - DB7	Not used, always read as 0.
DB8	EIPER - End of Measurement Period.
DB9	Not used, always read as 0.
DB10	IBEAT- Internal Heartbeat Occurred.
DB11	XBEAT - External Heartbeat Occurred.
DB12 - DB15	Not used, always read as 0.

RMODule<m> Status Register

Reports the summary status of all Receiver channels in the indicated Receiver module. These registers exist under the QUEStionable register hierarchy.

:STATus:QUEStionable:RMODule<n>:EVENt? - Returns the contents of the event register associated with the indicated status register in numeric form.

:STATus:QUEStionable:RMODule<n>:ENABle <numeric> - Sets the event enable register mask for the indicated status register.

:STATus:QUEStionable:RMODule<n>:ENABle? - Returns the current mask setting in numeric form for the indicated status register.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
-	-	-	-	-	-	-	-

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
-	-	-	-	CH4	CH3	CH2	CH1

DB0	Summary of CHANnel1 register in the indicated Receiver module.
DB1	Summary of CHANnel2 register in the indicated Receiver module.
DB2	Summary of CHANnel3 register in the indicated Receiver module.
DB3	Summary of CHANnel4 register in the indicated Receiver module.
DB4 - DB15	Not used, always read as 0.

CHANnel<n> Status Register

Condition, Event, and Event Enable Registers provide alarm condition monitoring for each receiver channel. These registers exist under the QUEStionable register hierarchy.

:STATus:QUEStionable:RMODule<m>:CHANel<n>:CONDition? - Returns the current state of the alarms. This action does not clear the register.

:STATus:QUEStionable:RMODule<m>:CHANnel<n>:EVENt? - Returns the contents of the event register in numeric form. Each set bit of the event register indicates a change of the associated condition bit. This action clears the event register by reading it.

:STATus:QUEStionable:RMODule<m>:CHANnel<n>:ENABle <numeric> - Sets the event enable register mask. Each set bit of the mask will allow the associated event register bits to indicate a change of the associated condition bit.

:STATus:QUEStionable:RMODule<m>:CHANnel<n>:ENABle? - returns the current setting for the event enable register mask.

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8
-	PSL	ERRS	-	-	SCG	-	-

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
-	-	-	-	AIS-L	OOF	LOF	LOS

DB0	Loss of Signal Detected
DB1	Loss of Frame Detected
DB2	Out of Frame Detected
DB3	Line AIS Detected
DB4-DB9	Not used, always read as 0
DB10	Single Channel Gating Active
DB11-DB12	Not used, always read as 0
DB13	Errors Detected
DB14	Pattern Sync Loss Detected
DB15	Not used.

Programmed Status Reporting

This section will describe how to use the SpectralBER alarms registers to monitor alarms events. It may be useful to refer to “Figure 4-1” on page 52.

The first level of alarm monitoring is provided by the Condition registers. These are updated every decisecond whether the instrument is gating or not and hence provide a snapshot of the current status of the monitored alarms. Reading these registers does not alter their contents.

Since alarm events may be transitory, the Condition registers on their own are of limited usefulness. The Event registers allow alarm events to be latched and thus monitored at some time after the alarm events have occurred. You can capture either Positive (alarms going on) or Negative (alarms going off) transitions are detected by setting up the Transition Filter register. The Event register once read is cleared.

Each valid bit in an event register can be configured to contribute to a summary bit in a higher level event register. For example in Spectralber each bit in the CHANnel<m> registers can be configured to contribute to the appropriate summary bit in the RMODule event registers. Bits are configured to contribute by setting the mask bit in the Event Enable Register.

In the same way, the bits in the RMODule<n> event registers can be configured to contribute to summary bits in the QUEStionable event register. Finally, the MOD<n> bits in the QUEStionable register can be configured to contribute to the QUES summary bit in the Status Byte.

Now by monitoring one bit the Status Byte, it is possible for an application program to be informed of alarms events on all equipped Spectralber receivers. You can do this either by using a GPIB Serial Poll or by configuring you program to be interrupted by SRQ events on the GPIB. On detection of an event, the application program can quickly determine it’s source by examining the bits set in the register hierarchy. Firstly, by observing the QUEStionable:EVENt register it can determine which RMODule<n> register raised the event. Now by examining that RMODule<n> register it can determine which CHANnel<m> register raised the event. Finally by examining the CHANnel<m> register indicated it can determine which alarm condition raised the event.

Programming Example

This example configures the SpectralBER registers to report only LOS and LOF alarms from Receiver Module 2 Channel 3. Firstly the Event Enable Register mask of the Status Register that first detects the defined conditions is set up. Next, the Event Enable Register masks of all subsequent Status Registers between the reporting Status Register and the Status Byte are set up as required.

`:STAT:QUES:RMOD2:CHAN3:ENAB 3`	/* *Set the CHANnel3 event enable register to summarize for LOF(Bit1) + LOS (Bit0)* */
`:STAT:QUES:RMOD2:CHAN3:PTR 3;NTR 0`	/* *CHANnel3 Transition Filter register set up to pass only positive transitions on bits 0 and 1* */
`:STAT:QUES:RMOD2:ENAB 4`	/* *Set the RMODule2 event enable register to summarize for CHAN3(Bit2)* */
`:STAT:QUES:RMOD2:PTR 4;NTR 0`	/* *RMODule2 Transition filter register set to pass only positive transitions on bit 2* */
`:STAT:QUES:ENAB 5096`	/* *Set the QUEStionable event enable register to summarize for MOD2(Bit10)* */
`:STAT:QUES:PTR 5096;NTR 0`	/* *QUEStionable Transition filter passes only positive transitions on bit 10* */
`*SRE 40`	/* *Status Byte Mask configured such that SRQ/Serial Poll operation are only affected by the Questionable Register Summary(Bit3) or the Standard Event Register summary (Bit 5)* */

Chapter 5
Example Programs using SCPI

Introduction

The SpectralBER system can be controlled from a PC or workstation using either SCPI commands, Universal Instrument Drivers or manually using a Graphical User Interface (or soft front panel). This chapter provides examples of how SCPI commands can be used to control the system.

For more information on the Graphical User Interface and the Universal Instrument Drivers, see the *Installation & System reference Manual*.

The examples given here are written in “C”, but the general principles and sequence of SCPI commands apply to and can be adapted easily to other programming languages.

Start Group Gating

This program illustrates the sequence of SCPI commands required to start the system gating.

```
/*”start_group_gating.c”
  This example program starts the SpectralBER system gating.
                     Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”    /* Change the file path to suit.
                             Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

  /* Initialize device */
  viPrintf (vi, “*RST\n”);

  /* Enable Synchronous Command Pulse system */
  viPrintf (vi, “:INIT3:CONT ON\n”);

  /* Enable 100ms Heartbeat control system */
  viPrintf (vi, “:INIT2:CONT ON\n”);

  /* Enable 100ms Heartbeat generation */
  viPrintf (vi, “:OUTP5:TTLT0:STAT 1\n”);

  /* Disable Command Pulse generation */
  viPrintf (vi, “:OUTP5:TTLT1:STAT 0\n”);

  /* Set Measurement Period to 60 seconds */
  viPrintf (vi, “:RMODALL:SENSALL:GATE:PER 60\n”);

  /* Set Synchronous Command to ONCE */
  viPrintf (vi, “:RMODALL:SENSALL:GATE:COMM ONCE\n”);

  /* Issue a Command Trigger to START */
  viPrintf (vi, “:RMODALL:SENSALL:GATE:IMM\n”);

  /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Stop Group Gating

This program illustrates the sequence of SCPI commands required to stop the system gating.

```
/*"stop_group_gating.c"
  This example program stops the SpectralBER system gating.
                     Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include "c:\vxipnp\win95\include\visa.h"      /* Change the file path to suit
                           Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

 /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, "GPIB0::09::01::INSTR", VI_NULL,VI_NULL, &vi);

  /* Initialize device */
  viPrintf (vi, "*RST\n");

  /* Set Synchronous Command to STOP */
  viPrintf (vi, ":RMODALL:SENSALL:GATE:COMM STOP\n");

  /* Issue a Command Trigger to STOP */
  viPrintf (vi, ":RMODALL:SENSALL:GATE:IMM\n");

  /* Allow gating to stop before executing the following termination commands*/
  /* Disable 100ms Heartbeat generation */
  viPrintf (vi, ":OUTP5:TTLT0:STAT 0\n");

  /* Disable 100ms Heartbeat control system */
  viPrintf (vi, ":INIT2:CONT OFF\n");

  /* Disable Synchronous Command system */
  viPrintf (vi, ":INIT3:CONT OFF\n");

  /* Ensure Heartbeat system is IDLE */
  viPrintf (vi, ":ABORT2\n");

  /* Ensure Synchronous Command System is IDLE*/
  viPrintf (vi, ":ABORT3\n");

  /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Start Single Channel Gating

This program illustrates the sequence of SCPI commands required to start a single channel gating.

```
/*”start_single_gating.c”
  This example program starts single channels of the SpectralBER system gating.
                   Note: You must change the address to suit your system.) */
#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”    /* Change the file path to suit.
                             Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

  /* Initialize device */
  viPrintf (vi, “*RST\n”);

  /* Enable Synchronous Command Pulse system */
  viPrintf (vi, “:INIT3:CONT ON\n”);

  /* Enable 100ms Heartbeat control system */
  viPrintf (vi, “:INIT2:CONT ON\n”);

  /* Enable 100ms Heartbeat generation */
  viPrintf (vi, “:OUTP5:TTLT0:STAT 1\n”);

  /* Disable Command Pulse generation */
  viPrintf (vi, “:OUTP5:TTLT1:STAT 0\n”);

/* Module 1 Port 1 */
  /* Set Measurement Period to 60 seconds */
  viPrintf (vi, “:RMOD1:SENS1:GATE:PER 60\n”);

  /* Set Synchronous Command to ONCE */
  viPrintf (vi, “:RMOD1:SENS1:GATE:COMM ONCE\n”);

  /* Issue a Command Trigger to START */
  viPrintf (vi, “:RMOD1:SENS1:GATE:IMM\n”);

/* Module 2 Port 4 */
  /* Set Measurement Period to 60 seconds */
  viPrintf (vi, “:RMOD2:SENS4:GATE:PER 60\n”);

  /* Set Synchronous Command to ONCE */
  viPrintf (vi, “:RMOD2:SENS4:GATE:COMM ONCE\n”);

  /* Issue a Command Trigger to START */
  viPrintf (vi, “:RMOD2:SENS4:GATE:IMM\n”);
```

```
   /* Close session */
   viClose (vi);
   viClose (defaultRM);
}
```

Stop Single Channel Gating

This program illustrates the sequence of SCPI commands required to stop single channel gating.

```
/*"stop_single_gating.c"
  This example program stops single channels of the SpectralBER system gating.
                   Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include "c:\vxipnp\win95\include\visa.h"      /* Change the file path to suit
                           Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

 /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, "GPIB0::09::01::INSTR", VI_NULL,VI_NULL, &vi);

  /* Initialize device */
  viPrintf (vi, "*RST\n");

/* Module 1 Port 1 */
  /* Set Synchronous Command to STOP */
  viPrintf (vi, ":RMOD1:SENS1:GATE:COMM STOP\n");

  /* Issue a Command Trigger to STOP */
  viPrintf (vi, ":RMOD1:SENS1:GATE:IMM\n");

/* Module 2 Port 4 */
  /* Set Synchronous Command to STOP */
  viPrintf (vi, ":RMOD2:SENS4:GATE:COMM STOP\n");

  /* Issue a Command Trigger to STOP */
  viPrintf (vi, ":RMOD2:SENS4:GATE:IMM\n");

/* Allow gating to stop before executing the following termination commands */
  /* Disable 100ms Heartbeat generation */
  viPrintf (vi, ":OUTP5:TTLT0:STAT 0\n");

  /* Disable 100ms Heartbeat control system */
  viPrintf (vi, ":INIT2:CONT OFF\n");

  /* Disable Synchronous Command system */
  viPrintf (vi, ":INIT3:CONT OFF\n");

  /* Ensure Heartbeat system is IDLE */
  viPrintf (vi, ":ABORT2\n");

  /* Ensure Synchronous Command System is IDLE*/
  viPrintf (vi, ":ABORT3\n");
```

```
    /* Close session */
    viClose (vi);
    viClose (defaultRM);
}
```

Set up a Transmitter

This program illustrates a sequence of SCPI commands to set up a transmitter.

```
/*"tx_set_up.c"
  This example program sets up a SpectralBER Transmitter Module.
                   Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include "c:\vxipnp\win95\include\visa.h"      /* Change the file path to suit
                         Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

 /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, "GPIB0::09::01::INSTR", VI_NULL,VI_NULL, &vi);

  /* Set all channels of the first transmitter to have a PRBS23 payload */
  viPrintf (vi, "TMOD1:SOURceALL:DATA:TELecom:PAYLoad:PATTern PRBS23\n");

  /* Set the laser on for all outputs of the first transmitter */
  viPrintf (vi, "TMODule1:OUTPutALL:STATus ON\n");

  /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Set up a Receiver

This program illustrates a sequence of SCPI commands to set up a receiver.

```
/*"rx_set_up.c"
  This example program sets up a SpectralBER Receiver Module.
                   Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include "c:\vxipnp\win95\include\visa.h"     /* Change the file path to suit
                         Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

 /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, "GPIB0::09::01::INSTR", VI_NULL,VI_NULL, &vi);

 /* Set all channels of the first receiver to have a PRBS23 payload */
  viPrintf (vi, "RMODule1:SENSeALL:DATA:TELecom:PAYLoad:PATTern PRBS23\n");

    /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Extract Receiver Results

This program illustrates a sequence of SCPI commands to return results from a Receiver.

```
/*”results.c”
  This example program returns results from a SpectralBER Receiver Module.
                     Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”      /* Change the file path to suit
                           Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

/* Returns all the payload pattern bit error counts from the first receiver */
  viPrintf (vi, “RMODule1:SENSeAll:DATA? "ECOount:PATTern"\n”);

    /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Read Status Registers

This program illustrates a sequence of SCPI commands to read the event status register of a Receiver.

```
/*”register_read.c”
  This example program reads the event status register of a SpectralBER Receiver
Module.
                     Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”     /* Change the file path to suit
                           Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

/* Returns numerically the contents of the event register of the first receiver*/
  viPrintf (vi, “:STATus:QUEStionable:RMODule1:EVENt?"\n”);

    /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Read History Registers

This program illustrates a sequence of SCPI commands to read the history of the event status register of a Receiver.

```
/*”history_register_read.c”
  This example program reads a history register of a SpectralBER Receiver Module.
                      Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”      /* Change the file path to suit
                            Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

/* Returns numerically the contents of the history register of the event register
of the first receiver*/
  viPrintf (vi, “:STATus:QUEStionable:RMODule1:HISTory?"\n”);

    /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Set up a Status Register Mask

This program illustrates a sequence of SCPI commands to set up a status register mask.

```
/*”mask_register.c”
  This example sets up a register mask.
                     Note: You must change the address to suit your system.) */

#include <conio.h>
#include <stdio.h>
#include “c:\vxipnp\win95\include\visa.h”      /* Change the file path to suit
                           Note: This header file is supplied with Visa. */

void main () {

  ViSession defaultRM, vi;

  /* Open session to GPIB device (Change the address to suit)*/
  viOpenDefaultRM (&defaultRM);
  viOpen (defaultRM, “GPIB0::09::01::INSTR”, VI_NULL,VI_NULL, &vi);

/* Sets the event enable register to summarize for channel 3 of the first
receiver */
  viPrintf (vi, “:STATus:QUEStionable:RMODule1:ENAB 4"\n”);

    /* Close session */
  viClose (vi);
  viClose (defaultRM);
}
```

Chapter 6 SCPI Error Messages

The system-defined error/event numbers are chosen on an enumerated ("1 of N") basis. The SCPI defined error/event numbers and the error description portions of the ERRor query response are listed here. The first error/event described in each class (for example, –100, –200, –300, –400) is a “generic” error. In selecting the proper error/event number to report, more specific error/event codes are preferred, and the generic error/event is used only if the others are inappropriate.

No Error

This message indicates that the device has no errors.

No Error

The queue is completely empty. Every error/event in the queue has been read or the queue was purposely cleared by power-on, *CLS, etc.

Command Errors [–199, –100]

An < error/event number > in the range [-199, -100] indicates that an *IEEE 488.2* syntax error has been detected by the instrument's parser. The occurrence of any error in this class should cause the command error bit (bit 5) in the event status register (*IEEE 488.2*, section 11.5.1) to be set. One of the following events has occurred:

- An *IEEE 488.2* system error has been detected by the parser. That is, a controller-to-device message was received which is in violation of the *IEEE 488.2* standard. Possible violations include a data element which violates the device listening formats or whose type is unacceptable to the device.
- An unrecognized header was received. Unrecognized headers include incorrect device-specific headers and incorrect or not implemented *IEEE 488.2* common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside an *IEEE 488.2* <PROGRAM MESSAGE>.

Events that generate command errors shall not generate execution errors, device-specific errors, or query errors.

–100 Command error

This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error as defined in *IEEE 488.2,* 11.5.1.1.4 has occurred.

–101 Invalid character

A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of errors `-114, -121, -141`, and perhaps some others.

–102 Syntax error

An unrecognized command or data type was encountered; for example, a string was received when the device does not accept strings.

–103 Invalid separator

The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, `*ESE 1 :OUTP1:TEL:RATE 140 Mb/s`

–104 Data type error

The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.

–105 GET not allowed

A Group Execute Trigger was received within a program message (see *IEEE 488.2,* 7.7).

–108 Parameter not allowed

More parameters were received than expected for the header; for example, the `*RCL` common command only accepts one parameter, so receiving `*RCL 0,1` is not allowed.

–109 Missing parameter

Fewer parameters were received than required for the header; for example, the `*ESE` common command requires one parameter, so receiving `*ESE` is not allowed.

–110 Command header error

An error was detected in the header. This error message should be used when the device cannot detect the more specific errors described for errors `-111` through `-119`.

–111 Header separator error

A character which is not a legal header separator was encountered while parsing the header; for example, no white space followed the header, thus `*ESE1` is an error.

–112 Program mnemonic too long

The header contains more that twelve characters (see *IEEE 488.2*, 7.6.1.4.1).

–113 Undefined header

The header is syntactically correct, but it is undefined by this specific device; for example, `*XYZ` is not defined for any device.

–114 Header suffix out of range

Indicates that a non-header character has been encountered in what the parser expects is a header element.

–120 Numeric data error

This error, as well as errors `-121` through `-129`, are generated when parsing a data element which appears to be numeric, including the non-decimal numeric types. This particular error message should be used if the device cannot detect a more specific error.

–121 Invalid character in number

An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a "9" in octal data.

–123 Exponent too large

The magnitude of the exponent was larger than `32000` (see *IEEE 488.2*, 7.7.2.4.1).

–124 Too many digits

The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see *IEEE 488.2*, 7.7.2.4.1).

–128 Numeric data not allowed

A legal numeric data element was received, but the device does not accept one in this position for the header.

–130 Suffix error

This error, as well as errors `-131` through `-139`, are generated when parsing a suffix. This particular error message should be used if the device cannot detect a more specific error.

–131 Invalid suffix

The suffix does not follow the syntax described in *IEEE 488.2*, 7.7.3.2, or the suffix is inappropriate for this device.

–134 Suffix too long

The suffix contained more than 12 characters (see *IEEE 488.2*, 7.7.3.4).

–138 Suffix not allowed

A suffix was encountered after a numeric element which does not allow suffixes.

–140 Character data error

This error, as well as errors `-141` through `-149`, are generated when parsing a character data element. This particular error message should be used if the device cannot detect a more specific error.

–141 Invalid character data

Either the character data element contains an invalid character or the particular element received is not valid for the header.

–144 Character data too long

The character data element contains more than twelve characters (see *IEEE 488.2*, 7.7.1.4).

–148 Character data not allowed

A legal character data element was encountered where prohibited by the device.

–150 String data error

This error, as well as errors `-151` through `-159`, are generated when parsing a string data element. This particular error message should be used if the device cannot detect a more specific error.

–151 Invalid string data

A string data element was expected, but was invalid for some reason (see *IEEE 488.2*, 7.7.5.2); for example, an `END` message was received before the terminal quote character.

–158 String data not allowed

A string data element was encountered but was not allowed by the device at this point in parsing.

–160 Block data error

This error, as well as errors `-161` through `-169`, are generated when parsing a block data element. This particular error message should be used if the device cannot detect a more specific error.

–161 Invalid block data

A block data element was expected, but was invalid for some reason (see *IEEE 488.2*, 7.7.6.2); for example, an `END` message was received before the length was satisfied.

–168 Block data not allowed

A legal block data element was encountered but was not allowed by the device at this point in parsing.

–170 Expression error

This error, as well as errors `-171` through `-179`, are generated when parsing an expression data element. This particular error message should be used if the device cannot detect a more specific error.

–171 Invalid expression

The expression data element was invalid (see *IEEE 488.2*, 7.7.7.2); for example, unmatched parentheses or an illegal character.

–178 Expression data not allowed

A legal expression data was encountered but was not allowed by the device at this point in parsing.

–180 Macro error

This error, as well as errors `-181` through `-189`, are generated when defining a macro or executing a macro. This particular error message should be used if the device cannot detect a more specific error.

–181 Invalid outside macro definition

Indicates that a macro parameter placeholder (`$<number>`) was encountered outside a macro definition.

–183 Invalid inside macro definition

Indicates that the program message unit sequence, sent with a `*DDT` or `*DMC` command, is syntactically invalid (see *IEEE 488.2*, 10.7.6.3).

–184 Macro parameter error

Indicates that a command inside the macro definition had the wrong number or type of parameters.

Execution Errors [–299, –200]

An `<error/event number>` in the range `[-299, -200]` indicates that an error has been detected by the instrument's execution control block. The occurrence of any error in this class should cause the execution error bit (bit 4) in the event status register (*IEEE 488.2*, section 11.5.1) to be set. One of the following events has occurred:

- A < PROGRAM DATA > element following a header was evaluated by the device as outside of its legal input range or is otherwise inconsistent with the device's capabilities.
- A valid program message could not be properly executed due to some device condition.

Execution errors shall be reported by the device after rounding and expression evaluation operations have taken place. Rounding a numeric data element, for example, shall not be reported as an execution error. Events that generate execution errors shall not generate Command Errors, device-specific errors, or Query Errors.

–200 Execution error

This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error as defined in *IEEE 488.2*, 11.5.1.1.5 has occurred.

–201 Invalid while in local

Indicates that a command is not executable while the device is in local due to a hard local control (see *IEEE 488.2,* 5.6.1.5); for example, a device with a rotary switch receives a message which would change the switches state, but the device is in local so the message can not be executed.

–202 Settings lost due to rtl

Indicates that a setting associated with a hard local control (see *IEEE 488.2*, 5.6.1.5) was lost when the device changed to `LOCS` from `REMS` or to `LWLS` from `RWLS`.

–210 Trigger error

–211 Trigger ignored

Indicates that a `GET`, `*TRG`, or triggering signal was received and recognized by the device but was ignored because of device timing considerations; for example, the device was not ready to respond. Note: a `DT0` device always ignores `GET` and treats `*TRG` as a Command Error.

–212 Arm ignored

Indicates that an arming signal was received and recognized by the device but was ignored.

–213 Init ignored

Indicates that a request for a measurement initiation was ignored as another measurement was already in progress.

–214 Trigger deadlock

Indicates that the trigger source for the initiation of a measurement is set to `GET` and subsequent measurement query is received. The measurement cannot be started until a `GET` is received, but the GET would cause an `INTERRUPTED` error.

–215 Arm deadlock

Indicates that the arm source for the initiation of a measurement is set to `GET` and subsequent measurement query is received. The measurement cannot be started until a `GET` is received, but the `GET` would cause an `INTERRUPTED` error.

–220 Parameter error

Indicates that a program data element related error occurred. This error message should be used when the device cannot detect the more specific errors described for errors `-221` through `-229`.

–221 Setting conflict

Indicates that a legal program data element was parsed but could not be executed due to the current device state (see *IEEE 488.2*, 6.4.5.3 and 11.5.1.1.5.)

–222 Data out of range

Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range as defined by the device (see *IEEE 488.2*, 11.5.1.1.5.)

–223 Too much data

Indicates that a legal program data element of block, expression, or string type was received that contained more data than the device could handle due to memory or related device-specific requirements.

–224 Illegal parameter value

Used where exact value, from a list of possibles, was expected.

–230 Data corrupt or stale

Possibly invalid data; new reading started but not completed since last access.

–231 Data questionable

Indicates that measurement accuracy is suspect.

–240 Hardware error

Indicates that a legal program command or query could not be executed because of a hardware problem in the device. Definition of what constitutes a hardware problem is completely device-specific. This error message should be used when the device cannot detect the more specific errors described for errors `-241` through `-249`.

–241 Hardware missing

Indicates that a legal program command or query could not be executed because of missing device hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device-specific.

–250 Mass storage error

Indicates that a mass storage error occurred. This error message should be used when the device cannot detect the more specific errors described for errors `-251` through `-259`.

–251 Missing mass storage

Indicates that a legal program command or query could not be executed because of missing mass storage; for example, an option that was not installed. Definition of what constitutes missing mass storage is device-specific.

–252 Missing media

Indicates that a legal program command or query could not be executed because of a missing media; for example, no disk. The definition of what constitutes missing media is device-specific.

–253 Corrupt media

Indicates that a legal program command or query could not be executed because of corrupt media; for example, bad disk or wrong format. The definition of what constitutes corrupt media is device-specific.

–254 Media full

Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk. The definition of what constitutes a full media is device-specific.

–255 Directory full

Indicates that a legal program command or query could not be executed because the media directory was full. The definition of what constitutes a full media directory is device-specific.

–256 File name not found

Indicates that a legal program command or query could not be executed because the file name on the device media was not found; for example, an attempt was made to read or copy a nonexistent file. The definition of what constitutes a file not being found is device-specific.

–257 File name error

Indicates that a legal program command or query could not be executed because the file name on the device media was in error; for example, an attempt was made to copy to a duplicate file name. The definition of what constitutes a file name error is device-specific.

–258 Media protected

Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present. The definition of what constitutes protected media is device-specific.

–260 Expression error

Indicates that an expression program data element related error occurred. This error message should be used when the device cannot detect the more specific errors described for errors `-261` through `-269`.

–261 Math error in expression

Indicates that a syntactically legal expression program data element could not be executed due to a math error; for example, a divide-by-zero was attempted. The definition of math error is device-specific.

–270 Macro error

Indicates that a macro-related execution error occurred. This error massage should be used when the device cannot detect the more specific errors described for errors `-271` through `-279`.

–271 Macro syntax error

Indicates that a syntactically legal macro program data sequence, according to *IEEE 488.2*, 10.7.2, could not be executed due to a syntax error within the macro definition (see *IEEE 488.2*, 10.7.6.3.)

–272 Macro execution error

Indicates that a syntactically legal macro program data sequence could not be executed due to some error in the macro definition (see *IEEE 488.2*, 10.7.6.3.)

–273 Illegal macro label

Indicates that the macro label defined in the `*DMC` command was a legal string syntax but could not be accepted by the device (see *IEEE 488.2*, 10.7.3 and 10.7.6.2); for example, the label was too long, the same as a common command header, or contained invalid header syntax.

–274 Macro parameter error

Indicates that the macro definition improperly used a macro parameter placeholder (see *IEEE 488.2*, 10.7.3).

–275 Macro definition too long

Indicates that a syntactically legal macro program data sequence could not be executed because the string or block contents were too long for the device to handle (see *IEEE 488.2*, 10.7.6.1).

–276 Macro recursion error

Indicates that a syntactically legal macro program data sequence could not be executed because the device found it to be recursive (see *IEEE 488.2*, 10.7.6.6).

–277 Macro redefinition not allowed

Indicates that a syntactically legal macro label in the `*DMC` command could not be executed because the macro label was already defined (see *IEEE 488.2*, 10.7.6.4).

–278 Macro header not found

Indicates that a syntactically legal macro label in the `*GMC?` query could not be executed because the header was not previously defined.

–280 Program error

Indicates that a downloaded program-related execution error occurred. This error message should be used when the device cannot detect the more specific errors described for errors `-281` through `-289`.

Note A downloaded program is used to add algorithmic capability to a device. The syntax used in the program and the mechanism for downloading a program is device-specific.

–281 Cannot create program

Indicates that an attempt to create a program was unsuccessful. A reason for the failure might include not enough memory.

–282 Illegal program name

The name used to reference a program was invalid; for example, redefining an existing program, deleting a nonexistent program, or in general, referencing a nonexistent program.

–283 Illegal variable name

An attempt was made to reference a nonexistent variable in a program.

–284 Program currently running

Certain operations dealing with programs may be illegal while the program is running; for example, deleting a running program might not be possible.

–285 Program syntax error

Indicates that a syntax error appears in a downloaded program. The syntax used when parsing the downloaded program is device-specific.

–286 Program runtime error

Query Errors [–399, –300]

An < error/event number > in the range `[-399, -300]` indicates that the instrument has detected an error which is not a command error, a query error, or an execution error; some device operations did not properly complete, possibly due to an abnormal hardware or firmware condition. These codes are also used for self-test response errors. The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register (*IEEE 488.2*, section 11.5.1) to be set. The meaning of positive error codes is device-dependent and may be enumerated or bit mapped; the <error message> string for positive error codes is not defined by SCPI and available to the device engineer. Note that the string is not optional; if the designer does not wish to implement a string for a particular error, the null string should be sent (for example `42," "`). The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register (*IEEE 488.2*, section 11.5.1) to be set. Events that generate device-specific errors shall not generate command errors, execution errors, or query errors; see the other error definitions in this section.

–300 Device-specific error

This is the generic device-dependent error for devices that cannot detect more specific errors. This code indicates only that a Device-Dependent Error as defined in *IEEE 488.2*, 11.5.1.1.6 has occurred.

–310 System error

Indicates that some error, termed "system error" by the device has occurred. This code is device dependent.

–311 Memory error

Indicates that an error was detected in the device memory. The scope of this error is device-dependent.

–312 PUD memory lost

Indicates that the protected user data saved by the `*PUD` command has been lost.

–313 Calibration memory lost

Indicates that nonvolatile calibration data used by the `*CAL?` command has been lost.

–314 Save/Recall memory lost

Indicates that the nonvolatile data saved by the `*SAV?` command has been lost.

–315 Configuration memory lost

Indicates that the nonvolatile data saved by the device has been lost. The meaning of this error is device-specific.

–330 Self-test failed

–350 Queue overflow

A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.

–360 Communication error

This is the generic communication error for devices that cannot detect the more specific errors described for errors `-361` through `-363`.

–361 Parity error in program message

Parity bit not correct when data received for example, on a serial port (for example, a baud rate mismatch).

–362 Framing error in program message

A stop bit was not detected when data was received for example, on a serial port.

–363 Input buffer overrun

Software or hardware input buffer on serial port overflows with data caused by improper or nonexistent pacing.

Query Errors [–499, –400]

An `<error/event number>` in the range `[-499, -400]` indicates that the output queue control of the instrument has detected a problem with the message exchange protocol described in *IEEE 488.2*, Chapter 6. The occurrence of any error in this class should cause the query error bit (bit 2) in the event status register (*IEEE 488.2*, section 11.5.1) to be set. These errors correspond to message exchange protocol errors described in *IEEE 488.2*, section 6.5. One of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

Events that generate query errors shall not generate command errors, execution errors, or device-specific errors; see the other error definitions in this section.

–400 Query error

This is the general query error for devices that cannot detect more specific errors. This code indicates only that a Query Error as defined in *IEEE 488.2*, 11.5.1.1.7 and 6.3 has occurred.

–410 Query INTERRUPTED

Indicates that a condition causing an `INTERRUPTED Query error occurred` (see *IEEE 488.2*, 6.3.2.3); for example, a query followed by `DAB` or `GET` before a response was completely sent.

–420 Query UNTERMINATED

Indicates that a condition causing an `UNTERMINATED Query error occurred` (see *IEEE 488.2*, 6.3.2.2); for example, the device was addressed to talk and an incomplete program message was received.

–430 Query DEADLOCKED

Indicates that a condition causing a `DEADLOCKED Query error occurred` (see *IEEE 488.2*, 6.3.1.7); for example, both input buffer and output buffer are full and the device cannot continue.

–440 Query UNTERMINATED after indefinite response

Indicates that a query was received in the same program message after a query requesting an indefinite response was executed (see *IEEE 488.2*, 6.5.7.5.7.)

Command Index

SCPI Commands