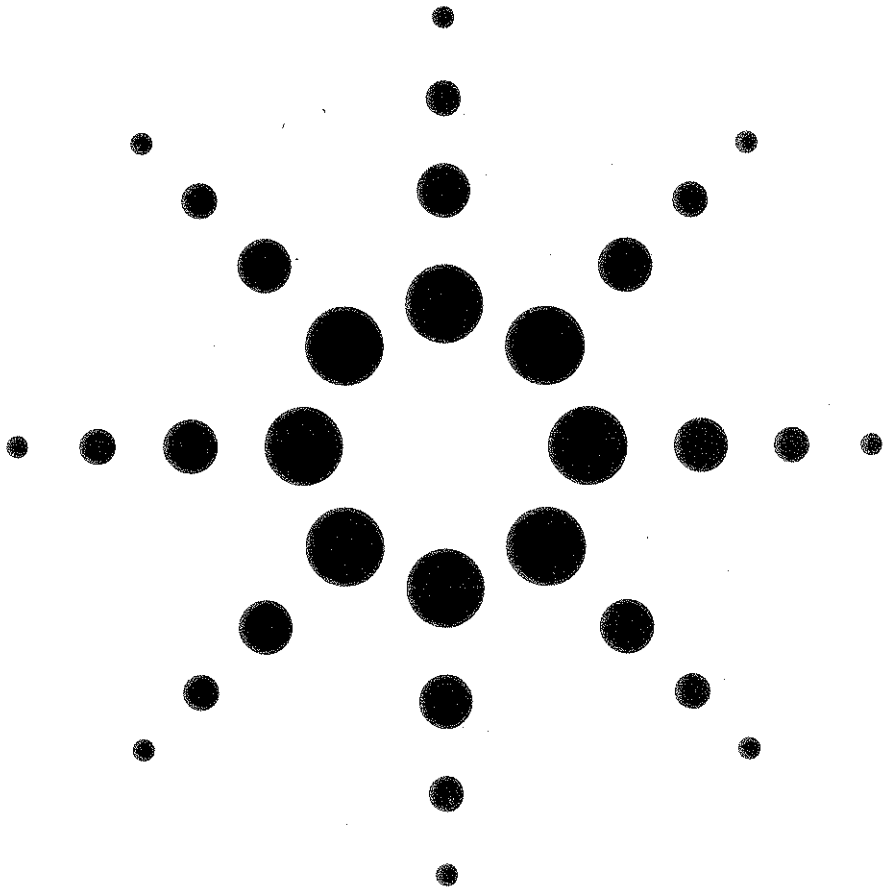
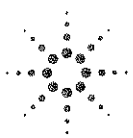
HP 71600B Series of Gbit/s Testers
Programming Manual





Agilent Technologies

HP 71600B Series of Gbit/s Testers Programming Manual

SERIAL NUMBERS

This manual applies directly to:

HP 70841B 0.1-3 Gbit/s Pattern Generator with serial number(s) prefixed 3136U.

HP 70842B 0.1-3 Gbit/s Error Detector with serial number(s) prefixed 3136U.

For important information about serial numbers, refer to SERIAL NUMBER INFORMATION in the HP 71600B Series Installation and Verification manual.

Serial number information for other elements in the system is contained in the following manuals:

Display - see HP 70004A Installation and Verification Manual
Mainframe - see HP 70001A Installation and Verification Manual
Clock Source - see HP 70311A/70312A Operating and Calibration Manual

HEWLETT PACKARD

HP Part No. 71600-90006
Microfiche Part No. 71600-90031
Printed in USA March 1992

Printing History

The Printing History shown below lists all Editions and Updates of this manual and the printing date(s). The first printing of this manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct the current Edition of the manual. Updates are numbered sequentially startingwith Update 1. When a new Edition is created, it contains all the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this printing history page. Many product updates or revisions do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

Edition 1 (71600-90006) March 1992

Contents

4. System Command Reference Section

Figures

Tables

1

Remote Operation

Introduction

This section contains the information to operate the instrument remotely using a suitable Controller. The aspects of remote operation covered are as follows:

- System Configuration
- Interface Types
- Hewlett-Packard Interface Bus
- Connecting the HP 71600B Series to the HP-IB
- Using HP-IB

System Configuration

The HP 71603B error performance analyzer system is factory preset to the following configuration:

Table 1-1. HP 71603B error performance analyzers

Model No.	Description	Default Mode	HP-IB,MS-IB Address
HP 70842B	3 GHz error detector	master	0,17
HP 70841B	3 GHz pattern generator	slave	1,18

Changing the error detector or pattern generator address is simply a matter of changing the setting of a small DIP switch inside the module. Full details of this operation are in the HP 71600B Series Installation and Verification manual, part number 71600-90005.

For more information about master operation, slave operation and MS-IB addressing refer to Section 2 of this manual (Programming the HP 71600B Series).

Note

The HP 71604B pattern generator systems require no change. The pattern generator in these systems is factory preset to be a master module.

Interface Types

There are two communications interfaces used in the HP 71600B Series. The MS-IB (Measurement System Interface Bus) and the HP-IB (Hewlett-Packard Interface Bus).

MS-IB — The Measurement System Interface Bus is the interface used for internal communication between system modules on the Modular Measurement System (MMS).

Operational details of the MS-IB interface can be ignored when the HP 71600B Series is remotely controlled. For further information refer to the HP 71600B Series Installation and Verification manual, part number 71600-90005.

HP-IB — The Hewlett-Packard Interface Bus is the interface used for communication between a controller and external devices such as the HP 71600B Series. The HP-IB conforms to IEEE standard 488-1978, ANSII standard MC 1.1 and IEC Recommendation 625-1.

Note

If you are using the HP-IB or MS-IB interfaces for the first time read this section first. More information about configuring the HP 71600B Series is contained in the Installation and Verification manual, part number 71600-90005.

Hewlett-Packard Interface Bus (HP-IB)

What is the HP-IB?

The Hewlett-Packard Interface Bus (HP-IB) is Hewlett-Packard's implementation of IEEE standard 488-1978, ANSII standard MC 1.1 and IEC Recommendation 625-1.

The HP-IB Interface is easy to use. It allows flexibility in both communicating and controlling data between a controller and the HP 71600B. It is also one of the easiest methods of constructing automatic test systems.

Devices on the bus fall into one of two categories, controller or non-controller. For example, the simplest system (two non-controllers) where one instrument is configured to send data continuously - known as TALKING and the other instrument (such as a printer) is configured to receive data continuously - known as LISTENING. Most devices can perform both roles, TALK or LISTEN, but not simultaneously. Usually a controller controls which instrument TALKS and which instrument LISTENS. The HP 71600B Series can TALK and LISTEN when instructed to do so by a suitable controller. In addition it can operate without a controller when logging results or screen dumping to an external printer configured in LISTEN ALWAYS mode.

The controller may also manage other instruments connected in the same bus configuration, addressing only one instrument, to carry out the data transfer or TALK function.

Further information on HP-IB standards and concepts is available in the following publications:-

- IEEE Interface Standard 488-1978
- ANSII Interface Standard MC 1.1
- Improving Measurements in Engineering and Manufacturing (HP P/N 5952-0078)
- Condensed Description of the Hewlett-Packard Interface Bus (HP P/N 59401-90030)

Connecting the HP 71600B Series to the HP-IB

Cabling Arrangements

There are two possible cabling arrangements when using the HP 71600B Series remotely, depending on whether the system is a pattern generator or an error performance analyzer.

The arrangement for an HP 71604B pattern generator system requires only one HP-IB cable. This is connected from the Controller to the HP-IB port on the display mainframe. The system then appears as an integrated system to the controller.

The other arrangement is for the HP 71603B error performance analyzer system. This arrangement causes the pattern generator and error detector to appear as separate devices to the controller. This configuration requires a cable to the HP-IB port on the display mainframe and to the additional mainframe in the system.

Using HP-IB

You should consider the following when connecting the instrument for operation over the HP-IB.

- Operating distances
- Instrument Mode at Power On
- Address Configuration
- Local and Remote Modes
- Using Local and Remote Commands
- HP-IB Required Commands
- Sending Commands Over the HP-IB
- Using Non-HP Controllers
- Invalid Commands
- Reading Data
- Message Format

Operating Distances

Up to 15 instruments can be connected on a local bus system, but it is important to ensure that the maximum HP-IB cable length between instruments is less than 2 meters. In addition the total cabling should not exceed 20 meters.

Some useful cable part numbers are listed in Table 1-2.

Table 1-2. Part Numbers of HP-IB Cables

Description	HP Part Number
1m	HP 10833A
2m	HP 10833B
4m	HP 10833C
0.5m	HP 10833D

For distances up to 1000m a suitable bus extender such as a HP 37203A or a HP 37201A can be used. Two bus extenders are required, one at the local bus and one at the remote bus. For distances beyond 1000m two HP 37201A bus extenders with suitable modems must be employed.

Note

The 4m cable may be used under certain conditions, usually the driver loading has to be altered to ensure satisfactory operation.

Instrument Mode at Power On

At power on the HP 71600B Series will wake up in the same mode as it was powered down in. Normally, at power on, the HP 71600B Series is ready for either front panel operation or remote operation. If however the instrument was performing a separate operation during power down, for example printing to an external printer or performing the instrument self test, it will require to complete that operation at power on before it is available for front panel or remote operation.

Caution

No HP-IB activity should take place within 15 secs of system power up, as this will effect the system power up routine and may result in system hang up.

Address Configuration

When configuring a HP-IB based system it is essential that each device on the HP-IB has a unique address. The device address can be in the range of 1 to 30. For a controller to communicate with a device over the HP-IB it must send the commands to the appropriate HP-IB device address.

Local and Remote Modes

The HP 71600B Series can be operated in one of two modes: local or remote.

In local operation, all the front controls are responsive and control the instrument.

In remote operation the softkeys which configure the system are inoperative, with exception of the display softkeys, and the instrument is controlled by the HP-IB controller. The front panel display reflects the remote programming commands received.

Using Local and Remote Commands

At power on the instrument is in local mode and is sent to the remote mode by one of two methods.

The first method uses a dedicated command and with HP Basic this is the REMOTE command followed by the instrument address that is REMOTE 717.

The second method is by sending any recognizable command string to the instrument. The instrument will recognize the command string, set itself to the remote mode and then act on that command.

There are three ways to return the instrument back to local mode. The first method is to use the HP Basic command LOCAL plus the instrument address, that is LOCAL 717. The second method is to press the front panel LCL key. The third method is to cycle power to the instrument.

Note

The instrument behaves differently in LOCAL mode if a LOCAL command is asserted on the interface bus by the controller. For example, to assert a local condition at interface 7, the command is simply LOCAL 7. When this condition is present sending a command string to the instrument will not cause it to enter the remote state. It will however act on the command string but remain in the local state.

To cancel the LOCAL 7 state you must use the REMOTE 7 command.

HP-IB Required Commands

The Required Commands perform the most basic remote functions over HP-IB and are common to all HP-IB controllable instruments. The commands are as follows:-

- DEVICE CLEAR
- SERIAL POLL
- REMOTE ENABLE
- LOCAL LOCKOUT
- GO TO LOCAL

Device Clear (CLEAR)

This command initializes the instrument HP-IB hardware.

The command format using HP 200/300 Series Basic is:-

for example. CLEAR 717

Serial Poll (SPOLL)

A serial poll will retrieve the value of the primary status byte. This byte contains useful information about the current state of the instrument.

for example.

SPOLL(717)

Remote Enable (REMOTE)

The Remote command instructs the instrument to enter the REMOTE state and be ready to accept instructions via HP-IB.

When the HP 71600B receives this command it illuminates the front panel REMOTE LED.

for example.

REMOTE 717

Local Lockout (LOCAL LOCKOUT)

It is recommended that the Local Lockout command is sent after the Remote. This disables the front panel local key preventing the return to local mode and thus any interference to the instrument settings.

It should always be preceded by the REMOTE command.

for example.

LOCAL LOCKOUT 7 (configures all the instruments on the bus to the Local Lockout condition.)

Note

If the instrument has been set to the LOCAL LOCKOUT condition, then the front panel LOCAL key is disabled. The instrument can only be returned to LOCAL operation by the controller sending the LOCAL command or by cycling power to the instrument.

Local (LOCAL)

The Local command returns the instrument from Remote operation to local front panel control. for example.

LOCAL 7 or LOCAL 717

Sending Commands Over HP-IB

To send commands over the HP-IB involves sending the command string via the interface select code to the device address. HP Computers use the Basic instruction OUTPUT to send command strings. The structure of a command line is as follows:-

OUTPUT interface select code + device address; "command string"

Note

The semi-colon symbol is the command separator and must be included.

The command string must be enclosed in inverted commas

Using an HP 300 Series Controller with its HP-IB interface set at select code 7 and a device at address 17, a typical command line to reset the instrument would appear as follows:-

```
OUTPUT 717;"*RST"
```

Using Non-HP Controllers

With non-HP controllers it may be necessary to send a suitable command terminator after the data message, the terminator can be:-

- ASCII newline (identical to the linefeed character, LF)
- ASCII carriage return + 1 linefeed, i.e. CR/LF

In most HP controllers the CR/LF is sent automatically when HP Basic OUTPUT statements are used.

Invalid Commands

A command will be rejected if:

- It contains a syntax error
- It cannot be identified
- It has too few or too many parameters
- A parameter is out of range
- It is out of context

All subsequent commands in the same string will be ignored.

Reading Data

It is possible to interrogate the individual settings and status of a device using query commands. Retrieving data is a two stage operation.

The query command is sent from the controller using the OUTPUT statement and the data is read from the device using the ENTER statement. A typical example, using the SCPI IEEE 488.2 Common Command "*IDN?" querying the identity of a device, is given as follows:-

```
OUTPUT 717;"*IDN?"

ENTER 717;Identity$

PRINT Identity$
```

Typically this would display the identity string "HEWLETT-PACKARD,70842B,0,A.01".

Note

When sending strings to the instrument either the double quote (") or the single quote may be used ('), the former being more suited to PASCAL programs which make use of single quote, the latter being more suited to use in BASIC programs, which uses double quote as a delimiter. In this manual the double quote has been used throughout.

Message Format

The HP 71600B Series has the capability of returning data in the following formats:-

- STRING
- NUMERIC
- BOOLEAN
- BLOCK DATA

String

Returns an ASCII string representing the instrument serial number, enclosed in quotes. This should be entered into a string variable.

Example:

```
10 OUTPUT 717;"*IDN?"
20 ENTER 717;Serial$
30 PRINT Serial$
40 END
```

Possible Result = "HEWLETT-PACKARD,70842B,0,A.01"

Numeric

Returns one of three numeric formats and can be entered into a string or numeric variable.

The three formats are:-

- An integer
- A number with embedded decimal point.
- A number with embedded decimal point and exponent.

Integer

Example:-

```
10 OUTPUT 717;"*STB?"
20 ENTER 717;Status_byte$
30 PRINT Status_byte$
40 END
```

Requests the contents of the status byte. Possible Result = +64

A Number with Embedded Decimal Point.

Example:-

```
10 OUTPUT 717;":VOLTAGE:ZOTHRESHOLD?"
20 ENTER 717;Level$
30 PRINT Level$
```

```
40 END
```

Requests the current voltage threshold that the system is operating at. Possible Result = -4.90000000E-001

A Number with Embedded Decimal Point and Exponent.

```
10 OUTPUT 717;"FETCH:ECOUNT?"
20 ENTER 717;Error_count
30 PRINT Error_count
40 END
```

Requests the frequency at which the system is operating.

Possible Result = +9.91000000E+012

Boolean

Boolean parameters are used to indicate whether a condition is true or false. A numeric value is returned where 1 = true and 0 = false.

Block Data

Block data is used when large quantities of related data is being returned. Blocks are returned as definite length blocks.

2

Programming the HP 71600B Series

Introduction

This section gives information on how to begin programming the HP 71600B Series.

The section covers the following topics:-

- The 71600B Series Command Language
- Command Types
- Important Points about SCPI
- SCPI Command Structure
- Master and Slave Operation
- Configuration Required for Remote Operation
- Programming in master/master/slave mode

The 71600B Series Command Language

The HP 71600B Series conforms to the standard language for remote control of instruments. Standard Commands for Programmable Instruments (SCPI) is the universal programming language for instrument control.

SCPI can be subdivided into two distinct command sets.

- Common Commands
- Instrument Control Commands

SCPI IEEE 488.2 Common Commands

This is a common command set which conforms to IEEE 488.2 and which contains general housekeeping commands.

The common commands are always headed by an asterisk. A typical example is the reset command:-

```
OUTPUT 718;"*RST"
```

The IEEE 488.2 command set also contains query commands. Query commands always end with a question mark. A typical example is the command querying the identity of a device at address 717.

```
OUTPUT 717;"*IDN?"
ENTER 718;Identity$
```

A full list of commands can be found in Section 4.

IEEE Mandatory Commands

The following IEEE 488.2 mandatory commands are implemented:

*CLS	Clear Status Command.
*ESE	Standard Event Status Enable Command.
*ESE?	Standard Event Status Enable Query.
*ESR?	Standard Event Status Register Query.
*IDN?	Identification Query.
*OPC	Operation Complete Command.
*OPC?	Operation Complete Query.
*RST	Reset Command.
*SRE	Service Request Enable Command.
*SRE?	Service Request Enable Query.
*STB?	Read Status Byte Query.
*TST?	Self-Test Query.
*WAI	Wait-to-Continue Command.

IEEE Optional Commands

The following optional commands are implemented:

*OPT?	Option Identification Query.
*PSC	Power On Status Clear Command.
*PSC?	Power On Status Clear Query.
*RCL	Recall device setup.
*SAV	Save device setup.

SCPI Instrument Control Commands

SCPI is the command language used to setup and control the HP 71600B Series hardware. It is a powerful command set designed for electronic test and measurement hardware.

SCPI is an extension of IEEE 488.2 and is a standard set of programming commands for all Hewlett-Packard test and measurement instrumentation. This section will explain the implementation of SCPI in the HP 71600B Series. For further information on SCPI refer to the Beginner's Guide to *TMSL, part number H2325-90001.

*TMSL (Test and Measurement System Language) is Hewlett-Packard's original implementation of SCPI.

Note

The response of the instrument to the *RST, *RCL or SYSTEM:PRESET commands may be up to 3 seconds. Any HP-IB program using these commands should have a timeout of greater than 3 seconds.

Important Points about SCPI

There are a number of key areas to consider when using SCPI for the first time.

These are as follows:-

- Instrument Model
- Layered Command Structure
- Command Syntax
- Optional Commands
- Sending Commands
- Command Separators

Instrument Model

SCPI guidelines require that the HP 71600B Series conforms to an instrument model. This ensures that when using SCPI, functional compatibility is achieved between instruments which perform the same tasks. For example, if two different instruments have a programmable clock frequency setting then both instruments would use the same SCPI commands to set their frequency. The instrument model is made up of a number of subsystems. Each subsystem is associated with a particular module in the modular measurement system (MMS).

The sub-system defines a group of functions within a module and has a unique identifier under SCPI which is called the Root Keyword.

For more detail on the instrument model refer to Section 4, the System Command Reference Section.

Layered Command Structure

The SCPI command structure is best explained by equating it with the HP 71600B Series instrument model. The top layer in SCPI identifies a sub-system within the modular measurement system (MMS). The next layer down is a command relating to that module or instrument within the MMS. The bottom layer is any parameter that is associated with that given command.

Command Syntax

Commands can be in short or long form. As we shall discover later in this section using the short form saves time when entering a program, however using long form makes a program more descriptive and easy to understand.

Optional Commands

Some layers in the SCPI command structure are optional. A typical example is where a command is unique to one module. In this case the top layer (that is, the Root Keyword) of the command structure may be omitted.

Sending Commands

Commands are sent over the HP-IB in the same way that HP-IB and IEEE 488.2 common commands are sent. HP controllers use the HP BASIC instruction OUTPUT to send commands strings. The only difference with SCPI is the structure of the command string.

Command Separators

The SCPI command structure is hierarchical and is governed by a number of symbols. For example, a change in the command hierarchy is indicated by a colon, similar level commands are separated by a semi-colon and parameters are separated by a comma. This is explained in more detail in the following section, headed SCPI Command Structure.

SCPI Command Structure

As previously stated the SCPI command set has a hierarchical layered structure.

The structure is as follows:-

Root Keyword + Command Keyword + Parameter(s)

Root Keyword — The Root Keyword is the top layer in the command structure. It identifies a subsystem within a module, which is contained in the modular measurement system.

The subsystems in the HP 71600B Series are given the following root keywords:-

Module	Subsystem	Root Keyword
Pattern Generator	Data Source	[SOURce[1]:]
	Clock Source	SOURce2
	Trigger Source	SOURce3
	Output Subsystem	OUTPut[1]
	System Subsystem	SYSTem
	Status Subsystem	STATus
Error Detector	Data Sense	[SENSe[1]:]
	Clock Sense	SENSe2
	Data Input	INPut[1]
	Measurement Subsystem	FETCh/PFETch
	Display Subsystem	DISPlay
	System Subsystem	SYSTem
	Status Subsystem	STATus

Some root keywords may be optional if the destination of the command is implicit in the Command Keyword.

Command Keyword — The layer below the Root Keyword is the Command Keyword. It describes the feature on the system which is to be changed. It will always be present in any command string and may have additional associated commands.

Parameter — The command parameters are the lowest layer in the SCPI command structure. They may be required by the Command Keyword and are numeric, string, boolean or block data.

Taking one command as an example we can examine this structure further.

Command Structure Example

In the following example we will examine a section of the pattern generator pattern selection command for the HP 71604B pattern generator and HP 71603B error performance analyzer systems.

The pattern command can be illustrated as follows:-

Root Keyword	Command Keyword	Parameter(s)
[SOURce[1]:]	PATTern	
	[:SELect]	PRBS(n)|ZSUB(n)
		MDENsity(n)|UPAT(n)
	[SELect]	PRBS(n)|ZSUB(n)
		MDENsity(n)|UPAT(n)

[SOURce [1]:] — This is the top layer of the command structure and identifies the pattern generator source sub-system.

PATTern	This is the next layer and is the equivalent of setting the front panel pattern selection field.
PRBS(n), ZSUB(n)	These are the parameters required by the PATTern command keyword.

Note

Any optional commands are enclosed in square brackets [ ] and any optional characters are shown in lower case.

A colon indicates a change of level in the command hierarchy. Commands at the same level in the hierarchy may be included in the same command line, if separated by a semi colon.

The bar symbol (|) indicates mutually exclusive commands.

To translate this into a command line you simply follow the same convention, however the command line can be typed in several different ways. This depends on whether longform or shortform is used. The following example gives three possible forms of the command line all of which are perfectly acceptable.

In longform:-

OUTPUT 718;"SOURCE1:PATTERN:SELECT PRBS7"

In shortform:-

OUTPUT 718;"SOUR1:PATT:SEL PRBS7"

With optional commands removed:-

OUTPUT 718;"PATTERN PRBS7"

It can be seen from the examples that longform is the most descriptive form of programming commands in SCPI and will be used for the examples given in this manual.

Master and Slave Operation

Instruments which operate under the control of the Modular Measurement System have two possible modes of operation, master or slave.

When an instrument operates in master mode it acts as a independent device to the controller and is controlled over the HP-IB via its own unique HP-IB address. When an instrument operates in slave mode it is under the control of another master instrument and is not directly addressable over the HP-IB.

Master and Slave Addresses

Each instrument in the modular measurement system has a unique ROW:COLUMN address. The ROW address determines if the module is set for master or slave operation and the column address determines the HP-IB address. Any module with a ROW address of 0 is a master and any module with ROW address 1 or higher is a slave. For more information consult the HP 71600B Series Installation and Verification Manual, part number 71600-90005.

Configuration Required for Remote Operation

The error detector/pattern generator/clock source instruments in the HP 71600B Series are factory configured to master/slave/slave mode. This configuration is designed for manual use of the system, and is not the recommended configuration for remote operation. The recommended configuration for remote operation is master/master/slave mode. The system will however operate in master/slave/slave mode, but requires additional programming commands. Refer to page 2-10 for advice on remote operation in master/slave format.

The required instrument configuration for remote operation is as follows:-

Table 2-1.

Model No.	Description	Instrument Mode	MS-IB,HP-IB Address
HP 70842B	3 GHz Error Detector	master	0,17
HP 70841B	3 GHz Pattern Generator	master	0,18
HP 70312A	3 GHz Clock Source	slave	2,19

Note

In the case of the HP 71600B Series the HP 70312A Clock Source is under the control of the pattern generator. The method of sending commands to the clock source is explained in the following sub-section headed Programming in Master/Master/Slave Mode.

Programming in Master/Master/Slave Mode

When programming in master/master/slave mode the instruments in master mode are addressed directly, however the slave modules receive their commands through the master which controls them. To send a command to a slave instrument, the command string is sent to the master that controls it and the master then passes the command string to the slave instrument over the MS-IB.

To do this the command string must identify which commands are intended for the master and which are for the slave. This is achieved by using an additional command, that is, "SYSTem:PTHRough" command.

Using the SYSTem:PTHRough Command

When a master module receives the SYSTEM:PTHRough command. It does not act on the parameter(s) following the command, instead it passes the parameter(s) to any slave present at a higher MS-IB row address. In the case of the 71600B Series the slave clock source is under the control of the master pattern generator. When the SYSTem:PTHRough command is sent to the pattern generator at address 718, the parameters within the SYSTem:PTHRough command are passed to the clock source at row address 2.

The actual parameter(s) that the SYSTEM:PTHRough command is passing to the slave instrument must be enclosed in inverted commas. For example, suppose we wish to send the *RST command to the slave clock source. The command line is as follows (note HP Basic does not allow double quotes inside double quotes therefore single quotes must be used).

```
OUTPUT 718;"SYSTEM:PTHROUGH '*RST'"
```

Note

The slave clock source is at MS-IB row address 2. If we suppose that another slave instrument is present at MS-IB row address 1, then because that instrument is higher in the slave hierarchy than the clock source and it would receive the parameter(s) in the SYSTem:PTHRough command.

In a configuration like this where two slaves are cascaded at different MS-IB row addresses then, two SYSTem:PTHRough commands would be used to send parameters to the lower order slave. The command form would require the second SYSTem:PTHRough command to be embedded within the first with inverted commas.

Caution

When the frequency of the clock source is changed some settling time is required. The recommendation is after a new frequency has been selected the pattern generator frequency measurement should be queried. When two successive readings are exactly equal, the frequency is stable.

If the error detector eye width measurement is to be made after a new frequency selection, the error detector frequency measurement should be queried for two successive equal frequency readings.

Example Program using Master/Master/Slave

The following short program gives a typical example of programming in Master/Master/Slave mode. The program sets the error detector, pattern generator and clock source to their default settings and then performs an error count measurement at 1 GHz.

```
10    ! Program name   PROG_S2_1
20    !
30    ! This program demonstrates how to program in MASTER/MASTER/SLAVE
40    ! mode.  It sets system to perform a basic error count measurement at
50    ! at 1 GHz.
60    !
70   PRINT CHR$(12)               ! Clear the screen.
80   Error_det=717                ! Assign HP-IB addresses to variables.
90   Pattern_gen=718
100  OUTPUT Error_det;"*RST;*CLS" ! Reset system.
110  OUTPUT Pattern_gen;"*RST;*CLS"
```

```
120  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH '*RST;*CLS'"
130  WAIT 3                          ! Allow settling time.
140  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH 'FREQUENCY 1GHZ'"   ! 1 GHz at +0DbM.
150  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH 'AMPLITUDE +0DBM;AMPLITUDE:STATE ON'"
160 Read_freq: !  This loop ensures the received frequency is stable.
170    OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
180    ENTER Error_det;Frequency1
190    WAIT 1
200    OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
210    ENTER Error_det;Frequency2
220    IF Frequency1<>Frequency2 THEN Read_freq
230  OUTPUT Pattern_gen;"PATTERN:EADDITION ON"  ! Enable error add.
240  OUTPUT Error_det;"GATE:MODE SINGLE"        ! Single shot gating.
250  OUTPUT Error_det;"GATE:PERIOD 5;STATE ON"  ! 5 secs.
260  WAIT 5
270  OUTPUT Error_det;"FETCH:ECOUNT?"           ! Display error count.
280  ENTER Error_det;Error_count
290  PRINT "Error Count is",Error_count
300  END
```

Note

More program examples are given in Section 3, Interrogating the Instrument Status, and in Section 5, Program Examples.

Programming in Master/Slave Mode

It is possible to operate the Pattern Generator as an MMS slave to the Error Detector. When operated in this fashion, the following rules apply:

1. The slave cannot be spoken to directly over HP-IB. Instead, it must use the master as a 'relay' station.

Two HP-IB commands make this possible:

- SYSTem:PTHRough[:STRing] <string>
- SYSTem:PTHRough[:STRing]? <string>?

2. All commands under the master [SENSe[1]:]PATTern node are automatically passed through to the slave. All other commands are not; that is, to reset both the master and slave, two commands are needed:

- OUTPUT 7xx;"*RST"
- OUTPUT 7xx;"SYST:PTHR '*RST'"

3. The slave cannot issue a Service Request directly. Instead, it must notify its master that it had a problem. To achieve this, bit 1 of the master's Status Register Structure is defined as the 'Slave needs service':

Then, upon the control program realizing that bit 1 of the master has been set, it must use the 'SYSTem:PTHRough?' command to interrogate the slave's Status with a combination of:

```
OUTPUT 7xx;"SYST:PTHR '*STB?'"
OUTPUT 7xx;"SYST:PTHR '*ESR?'"
OUTPUT 7xx;"SYST:PTHR 'STAT:OPER:COND?'"
OUTPUT 7xx;"SYST:PTHR 'STAT:FAIL:EVEN?'"
OUTPUT 7xx;"SYST:PTHR 'SYST:ERR?'"
```

4. The MAV (Message Available) Summary message in the slave's Status Byte Register is never set. Instead, the RQS (Request Service) message is set, and this causes the Slave Service bit of the master to be set.

Note

Commands using <block_data> parameters cannot be used with the PTHRough command.

3

Interrogating the Instrument Status

Introduction

This section explains how to use the powerful status reporting features which are contained in the HP 71600B Series.

It explains the structure of the internal registers with examples on how to interrogate them. It also explains the concept of interrupt programming using the Service Request.

The section covers the following topics:-

- HP 71600B Series Status Reporting
- Status Register Group Model
- Pattern Generator Register Model
- Error Detector Register Model
- Description of the Status Registers
- Interrupt Programming and using the Service Request

HP 71600B Series Status Reporting

The HP 71600B Series has powerful status and reporting features which give important information about events and conditions within the instrument, for example flag the end of a measurement or perhaps indicate a command error. To access this information requires interrogating a set of registers using Standard Commands for Programmable Instruments (SCPI).

Internal Registers

The registers contained in the HP 71600B Series are as follows:-

Table 3-1. Internal Registers

Pattern Generator	Error Detector
Status Byte	Status Byte
Standard Event Status	Standard Event Status
Questionable Data Status	Questionable Data Status
	Operation Status
Failure Status	Failure Status

The internal register layout conforms to SCPI standards. This requires that each module in the Modular Measurement System (MMS) conforms to a register model. The pattern generator and error detector register models are explained separately, however the operation of the Status Byte, Failure Status and Standard Event register is identical in both cases and is explained for the pattern generator model only.

The internal registers are read using a combination of SCPI common commands and SCPI status commands. The method of reading each register is explained in the following sub-sections headed Description of the Status Registers.

Generalized Status Register Group Model

SCPI guidelines specifies a register group model which is the building block of the SCPI status reporting system. The SCPI generalized status register group

model is shown in Figure 3-1.

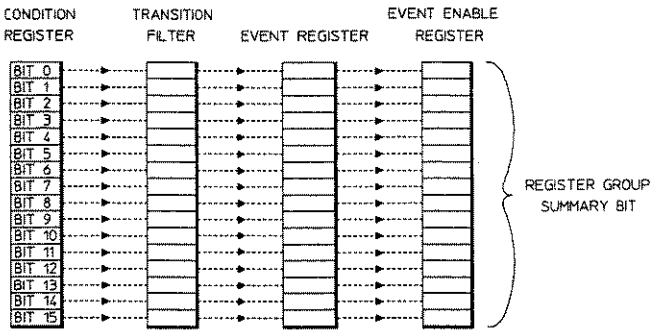


Figure 3-1. Generalized Status Register Group

Condition Register This register monitors the hardware and firmware status of the instrument. There is no latching of conditions in this register, it is updated in real time.

Transition Filter	As the name implies it determines whether positive or negative transitions (true or false) in the Condition register sets the Event register.
Event Register	This register latches the transient states that occur in the Condition register as specified by the Transition Filter.
Enable Register	The 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.

This reporting structure is the basis of generating interrupts that is, service requests, and is explained more fully in the section headed Interrupt Programming.

Pattern Generator Register Model

We will examine the register model by first considering the pattern generator.

The register model is shown in Figure 3-2.

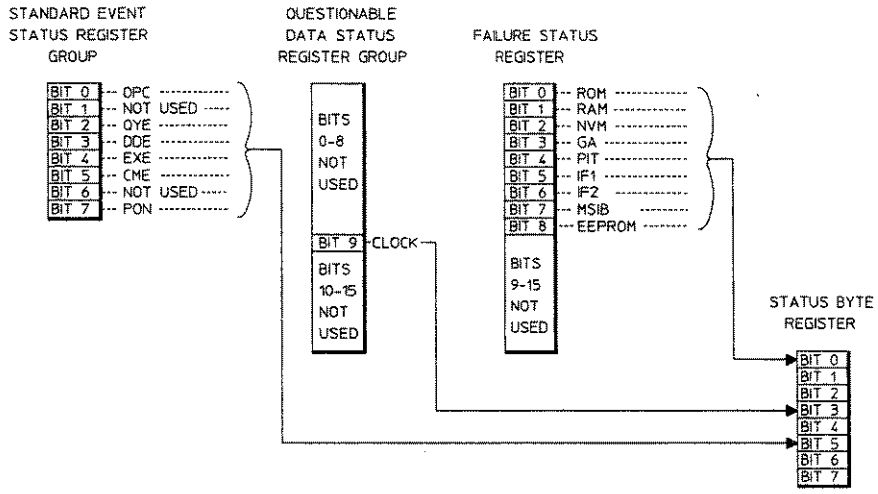


Figure 3-2. Pattern Generator Register Model

Note

SCPI guidelines specify the layout of each register group. The bits in each register are allocated a specific purpose. The HP 71600B Series of instruments only use some of the specified features, therefore the status registers contain a number of unused bits.

The pattern generator register group model consists of the following register groups:- the Standard Event, Questionable Data and Failure Status registers groups. These register groups all report to the Status Byte.

Error Detector Register Model

The error detector register model differs from the pattern generator in two respects.

Firstly, the register model has an additional register group, the Operation Status register group.

Secondly, the Questionable Data register has more information than its counterpart in the pattern generator. The error detector register model is shown in Figure 3-3.

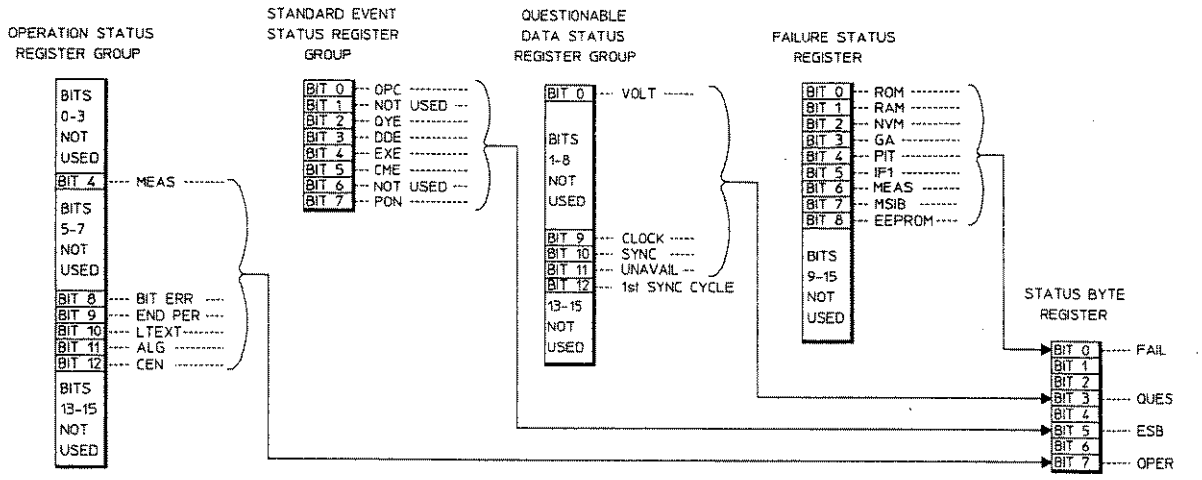


Figure 3-3. Error Detector Register Model

A full description of the different registers given in the following section headed Description of Status Registers.

Description of Status Registers

The following descriptions of the Status Byte register group, Standard Event register group and Failure Status register apply to both the pattern generator and the error detector. The Questionable Data Status register groups differ between instruments and the Operation Status register applies to the error detector only, therefore these are explained separately.

Status Byte Register Group

The Status Byte is the summary register which the other registers report to. Each reporting register is assigned a bit in the Status byte register which it can use to summarize its status.

Table 3-2. Status Byte Register

Bit #	Mnemonic	Description	Bit Value
0	FAIL	Failure Status register summary bit.	1
1	SLAVE	Slave Service bit.	2
2	-	This bit is not used.	
3	QUES	Questionable Data Status register summary bit.	8
4	MAV	Output queue summary bit.	16
5	ESB	Standard Event register summary bit.	32
6	RQS or MSS	SRQ or master status summary bit.	64
7	-	Operation Status register summary bit (error detector only).	128

FAIL Summary Bit — Bit 0, indicates there are bits set in the Failure Status register. This in turn indicates there has been a major hardware failure in the instrument.

SLAVE Service Bit — Bit 1, indicates that a slave instrument is requesting service. This bit is used in master/slave configurations only.

QUES Summary Bit — Bit 3, indicates that a bit has been set in the Questionable Data Status register. The bits in the Questionable Data Status register indicate when a signal is of questionable quality.

MAV Summary Bit — Bit 4, is the message available summary bit. This bit remains set until all the output messages are read from the instrument. The instrument stores its messages in an output queue. These messages are read by addressing the instrument to talk and reading the data. The availability of this data is summarized by the MAV bit.

ESB Summary Bit — Bit 5, indicates that a bit in the Standard Event register has been set.

RQS or MSS Summary bit — Bit 6 of the Status Register has two definitions depending of the method used to access the register. If the value of the register is read using the serial poll bit 6 is referred to as the RQS (request service bit) as this is the means used to inform the active controller that the instrument has set the service request control line (SRQ) i.e. interrupted the controller.

If the register is read using the *STB? common query command, then bit 6 is referred to as the master summary bit or MSS bit. It is this bit which indicates the instrument has requested service. The MSS bit is not cleared when the register is read using the *STB? command. It always reflects the current status of all the instrument's status registers.

Serial Polling

The Status Byte register can be interrogated by serial polling the instrument.

It is easily done using an HP Controller booted up in BASIC. The command for serial polling is SPOLL, as shown in the example program lines below.

```
10 Status_value = SPOLL(717)
20 PRINT Status_value
```

The binary weighted decimal value returned in the variable Status_value is the value of the Status Byte. The Status Byte gives a summary of the state of the reporting registers and will indicate a change of state in any reporting registers.

Another way of reading the value of the Status Byte is by using the *STB? common query command.

```
10 OUTPUT 717;"*STB?"
20 ENTER 717;Status_value
```

In this case the value returned by the variable Status_value is exactly the same as the value returned by carrying out a Serial Poll.

Status Byte Service Request Enable Register

The Service Request Enable register is an 8 bit register which acts as a mask on the Status Byte. The Service Request Enable register is programmed using the SCPI common command *SRE. When the register is programmed with any given value this determines when the instrument may issue a service request. For a service request to be issued the summary bit in the Status Byte must match the bit in the Service Request Enable Register. See Figure 3-4.

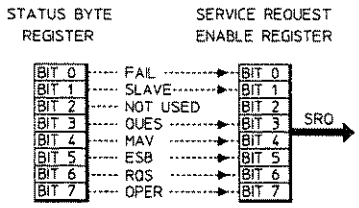


Figure 3-4. Status Byte Register

For example if bit 3 is set in the Service Request Enable register then the instrument only issues a service request when the corresponding bit is set in the Status Byte, that is bit 3, the Questionable Status register summary bit. See the following example:-

```
OUTPUT 717;"*SRE16"
```

This sets bit 3 of the Service Request Enable register.

Note For a more detailed description on service request programming and example programs, refer to the sub-section headed Interrupt Programming.

Standard Event Status Register Group

The Standard Event Status register group is a 16 bit register group which gives general purpose information about the instrument. It is unique in that it is the only reporting register group programmed using SCPI common commands while the other reporting registers are programmed using the SCPI Status command set.

Note

This register conforms, in part, to the generalized status register model. It comprises of an Event and Enable register, but no Condition register or Transition Filter. Therefore all positive (true) states occurring in this register are latched.

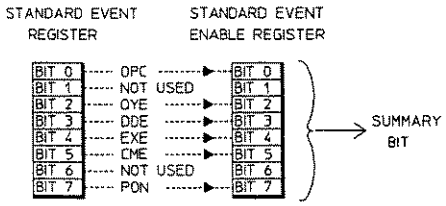


Figure 3-5. Standard Event Status Register

The bits in the Standard Event Status register group are defined as follows:-

Table 3-3. Standard Event Status register

Bit #	Mnemonic	Description	Bit Value
0	OPC	Operation Complete bit.	1
1	-	Not used	
2	QYE	Query Error bit.	4
3	DDE	Device Dependent Error bit.	8
4	EXE	Execution Error bit.	16
5	CME	Command Error bit.	32
6	URQ	User Request bit.	64
7	PON	Power On bit.	128
8-15	-	These bits are not used.	

Operation Complete Bit The operation complete bit, bit 0, is set in response to the *OPC command if the instrument has completed all its pending operations.

Request Control Bit This bit is not used in this instrument.

Query Error Bit The query error bit, bit 2, indicates there is a problem with the output data queue. Either there has been an attempt to read the queue when it is empty or the output data has been lost.

Device Dependent Error Bit	The device dependent error bit, bit 3, is set when an error of some kind has occurred in the instrument.
Execution Error Bit	The execution error bit, bit 4, is set when a command (HP-IB instrument specific) cannot be executed due to an out of range parameter or some instrument condition existing that prevents the execution. for example the instrument is already set to the wrong range.
Command Error Bit	The command error bit, bit 5, is set whenever the instrument detects an error in the format or content of the program message (usually a bad header, missing argument, or wrong data type etc.).
User Request Bit	The user request bit, bit 6, is set whenever the instrument control is changed from remote to local.
Power On Bit	The power on bit, bit 7, is set each time the instrument is powered from off to on.

The Standard Event register can be interrogated using the *ESR? common query command. It is an event register which is cleared after it is read.

```
OUTPUT 717;"*ESR?"

ENTER 717;Event_reg$

PRINT Event_reg$
```

Requests the contents of the Standard Event register. Possible result = +16 The Standard Event Register may also be cleared without having to interrogate it. This is done by using the "*CLS" command.

Standard Event Enable Register

The Standard Event Enable register is a 16 bit register which acts as a mask on the Standard Event Status register. It allows one or more event bits in the Standard Event register to set the ESB summary bit, bit 5, in the Status Byte.

For example, if bit 0 is set in the Standard Event Enable register, then, when the OPC bit in the Standard Event register goes true, the ESB summary bit is set in the Status Byte.

The Standard Event Enable register is set using the "*ESE" command. The following gives an example of setting bit 0, 1 and 2 in the Standard Event Enable register.

```
OUTPUT 718;"*ESE 7"
```

Failure Status register

The Failure Status register is a 16 bit event register, however in the HP 71600B Series only 8 bits are used. The bits in this register are set to indicate a major hardware element of the instrument has failed.

Table 3-4. Failure Status Register

Bit #	Description	Bit Value
0	ROM failure.	1
1	RAM failure.	2
2	Non Volatile memory corrupt.	4
3	Gate array failure.	8
4	PIT failure.	16
5	Interface board #1 failure.	32
6	Interface board #2 failure.	64
7	MSIB failure.	128
8	EPROM failure.	256
9-15	These bits are not used.	

Note

There is no Condition or Enable registers for the Failure Status register. Any failures in the instrument are latched and indicated by this register. The FAIL bit (bit 0) in the Status Byte register is automatically set whenever any bit in the Failure Status register is set. Failures of this type are not recoverable.

Questionable Data Status Group - Pattern Generator

The Questionable Data Status group is a 16 bit register group, however in the pattern generator's case only 1 bit is used. The bits in this register set indicate that a signal is of questionable quality.

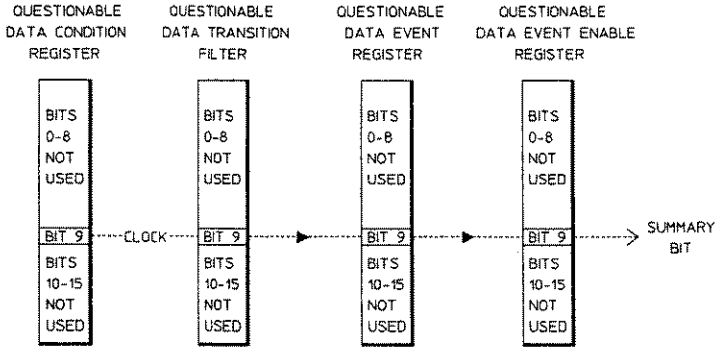


Figure 3-6. Questionable Data Status Register Group - Pattern Generator

The Questionable Data Status register group conforms to the SCPI Status Register model and is defined as follows:-

Table 3-5. Questionable Data Status register

Bit #	Mnemonic	Description	Bit Value
0-8	These bits are not used.		
9	CLOCK	Clock loss.	512
10-15	These bits are not used.		

Interrogating Register Groups

The Questionable Data Status register group is interrogated using SCPI status commands. The command format consists of

"Command identifier:Register group identifier:Register title"

Interrogating the Condition and Event Registers

The Condition and Event registers are interrogated using the :CONDITION? and :EVENT commands. See the following example:-

Condition Register Query

```
OUTPUT 718;"STATUS:QUESTIONABLE:CONDITION?"
ENTER 718;Question_con_reg
PRINT Question_con_reg
```

Event Register Query

```
OUTPUT 718;"STATUS:QUESTIONABLE:EVENT?"
ENTER 718;Question_evt_reg
PRINT Question_evt_reg
```

Transition Filter

The Transition Filter state is set using the ":PTRANSITION " and ":NTRANSITION" commands. The Transition Filter can be set to pass either positive transitions, negative transitions or both.

The default setting of the Transition Filter is to pass positive transitions only. To also pass a negative transition on bit 9, that is detect clock gain, from the Condition register to the Event register the command is as follows:-

```
OUTPUT 718;"STATUS:QUESTIONABLE:NTRANSITION 1024"
```

To reset the Transition Filter to pass only positive transitions at bit 9, the command is as follows:-

```
OUTPUT 718;"STATUS:QUESTIONABLE:NTRANSITION 0"
```

Questionable Data Event Enable Register

The Questionable Data Event Enable register acts as a mask on the Questionable Data Event register. It is enabled by sending the command ":ENABLE". The following example allows one or more event bits in the Questionable Data Event register to set the QUES summary bit in the Status Byte.

```
OUTPUT 718;"STATUS:QUESTIONABLE:ENABLE 512"
```

This enables bit 9, Clock Loss. Whenever a clock loss condition occurs in the condition register the QUES summary bit (bit 3) is set in the Status Byte register.

Questionable Data Status Group - Error Detector

The error detector Questionable Data Status register group is a 16 bit register set however in the error detector only 4 bits are used. The bits in this register set indicate when a signal is of questionable quality.

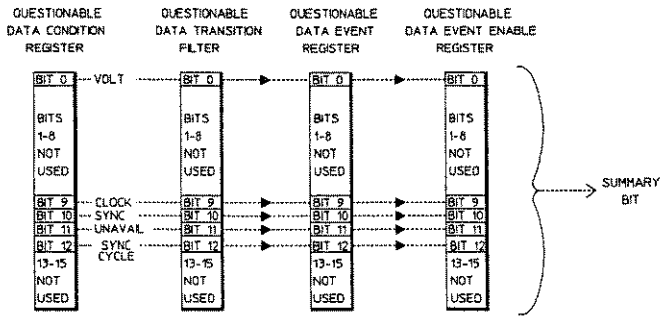


Figure 3-7. Questionable Data Status Register Group

Table 3-6. Questionable Data Status register group

Bit #	Mnemonic	Description	Bit Value
0	VOLTage	Indicates data loss.	1
1-8	These bits are not used.		
9	CLOCK	Clock loss.	512
10	SYNC	Sync Loss.	1024
11	UNAV	Signal is unavailable.	2048
12	Inst. dependent	1st SYNC CYCLE	4096
13-15	These bits are not used		

Interrogating the Condition and Event Registers

The error detector Questionable Data register group is interrogated/programmed by the same method as the pattern generator. The following example gives the commands for interrogating the Condition and Event registers.

Condition Register Query: `OUTPUT 717;"STATUS:QUESTIONABLE:CONDITION?"`

Event Register Query: `OUTPUT 717;"STATUS:QUESTIONABLE:EVENT?"`

Questionable Data Transition Filter

The Questionable Data Transition Filter state is set in the same way as in the pattern generator that is, the ":PTRANSITION" and ":NTRANSITION" commands. A typical example would be as follows:-

OUTPUT 717;"STATUS:QUESTIONABLE:PTRANSITION 8"

OUTPUT 717;"STATUS:QUESTIONABLE:NTRANSITION 8"

This sets the Transition Filter to pass positive and negative transitions at bit 3.

Questionable Data Status Enable Register

The Questionable Data Status register acts as a mask on the Questionable Data Event register. It is enabled by sending the command ":ENABLE". The following example allows one or more event bits in the Questionable Data Status register to set the QUES summary bit in the Status Byte.

OUTPUT 718;"STATUS:QUESTIONABLE:ENABLE 4"

sets bit 2, TIME

or OUTPUT 718;"STATUS:QUESTIONABLE:ENABLE 516"

sets bit 2, TIME - and bit 9, CLOCK LOSS.

If bit 2 is set in the Question Status Enable Register, then, when the TIME bit in the Question Status register goes true, the QUES summary bit is set in the Status Byte.

Operation Status Register Group - Error Detector

The Operation Status register is a sixteen bit register group unique to the error detector of which only 6 bits are used. This register group conforms to the SCPI register model and gives information about the current operation the instrument is performing. The Operation Status register group is defined as follows:-

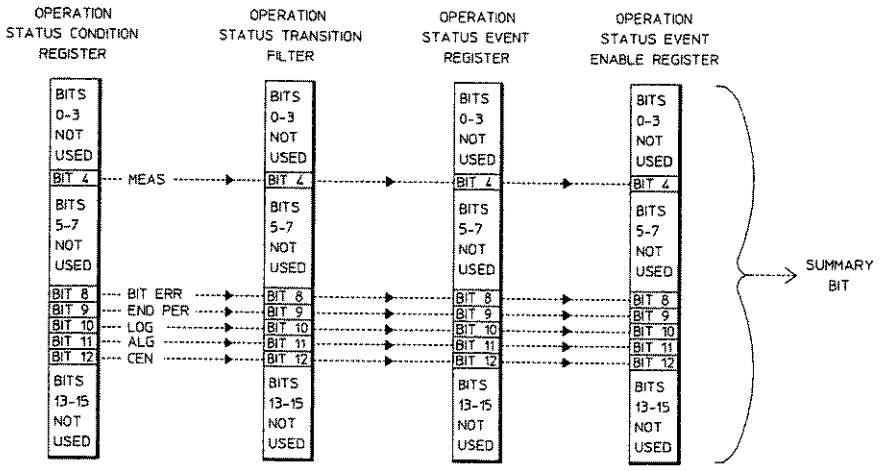


Figure 3-8. Operation Status Register Group

The bit in the Operation Status register group are defined as follows:-

Table 3-7. Operation Status register

Bit #	Mnemonic	Description	Bit Value
0-3	These bits are not used.		
4	MEAS	Measuring (Gating)	16
5-7	These bits are not used.		
8	ERR	Bit Error	256
9	END	End of repetitive measurement period	512
10	LOG	Instrument logging	1024
11	ALIG	Aligning Clock and Data	2048
12	CENT	Centering Clock and Data	4096
13-15	These bits are not used.		

Interrogating the Condition and Event Registers

The Error Detector Operation Status register group set is interrogated/programmed by the same method as the Questionable Data Status register group, except the register group identifier is changed. The following example gives the commands for interrogating the Condition and Event registers.

Condition Register Query: `OUTPUT 718;"STATUS:OPERATION:CONDITION?"`

Event Register Query: `OUTPUT 718;"STATUS:OPERATION:EVENT?"`

Operation Status Transition Filter

The Operation Status register group Transition Filter value is set in the same way as the Questionable Data Status register group that is, the "PTRANSITION" and "NTRANSITION" commands.

A typical example would be as follows:-

```
OUTPUT 718;"STATUS:OPERATION:PTRANSITION 8"
```

This sets the Transition Filter to pass a positive transition from the Condition register at bit 3.

Operation Event Enable Register

The Operation Event Enable register is enabled by sending the SCPI command shown in the example below. This allows one or more event bits in the Operation Status register to set the OPER summary bit in the Status Byte register.

```
OUTPUT 718;"STATUS:OPERATION:ENABLE 512"
```

If bit 9 is set in the Operation Event Enable register and the END PERIOD bit in the Operation Event register goes true, then the OPER bit is set in the Status Byte.

Interrupt Programming and using the Service Request

The method of interrogating the reporting registers is to read the register using SCPI status commands. This method is perfectly adequate for most applications, however should you wish to detect when a particular event occurs this would require the register to be continually polled. This problem is solved using interrupts.

Interrupts allow the instrument to interrupt the controller when a particular register has changed. The controller can then suspend its present task, service the instrument and then return to its initial task. It is more convenient and more efficient to get the instrument to issue a service request (SRQ) when an event or condition occurs, rather than continually monitor the instrument.

The basic steps involved in generating a service request (SRQ) are as follows:-

- Decide which particular event in a given status register you wish to trigger the service request.
- Set the Transition Filter to pass the chosen transition of that event.
- Set the Enable register from that register group to pass that event to set the summary bit in the Status Byte register.
- Set the Status Byte Enable register to generate an SRQ on the chosen summary bit being set.

The process is best explained by looking at an actual example. The following example generates an SRQ from an event in the Operating Status group.

Generating a Service Request from the Operating Status Register

The following example causes the error detector to generate a service request at the end of a measurement period using bit 4 of the Operation Status. See Figure 3-9.

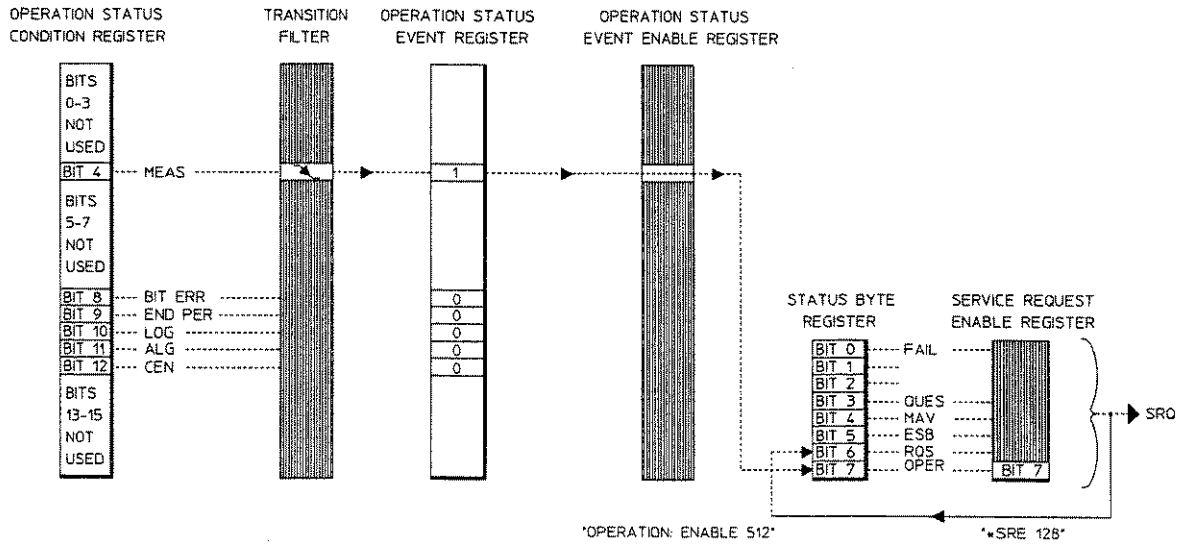


Figure 3-9. Service Request Illustration

Note

The SRQ enable bit, bit 6, of the Status Byte is the master status summary bit and will automatically be set on the occurrence of a service request.

The basic steps involved in setting the instrument to generate this service request are as follows:-

Step 1 — Set the Transition Filter to pass the chosen condition, either when it is true (positive) or when it is false (negative). The default value of the Transition Filter is that all positive (true) conditions are passed.

Step 2 — Program the Operation Enable Event register to allow bit 9 in the Event register to set the summary bit in the Status Byte register.

Step 3 — Program the Service Request Enable register to generate a service request when the Operation Status summary bit (OPER) is set in the Status Byte register.

Translating these three steps into to SCPI command lines it appears as follows:-

```
10 Error_det=717
20 OUTPUT Error_det;"STATUS:OPERATION:PTRANSITION 16"
30 OUTPUT Error_det;"STATUS:OPERATION:ENABLE 16"
40 OUTPUT Error_det;"*SRE 128"
50 END
```

Line 10 assigns the error detector address to variable Error_det.

Line 20 sets bit 9 the Operation Status Enable register which causes the start or end of the measurement period to set the OPER summary bit in the Status Byte register.

Line 30 enables the Transition Filter to pass a positive transition in bit 4 of the Operation Status Condition register. The default state is to pass positive transitions, therefore this step is in fact unnecessary.

Line 40 sets the Service Request Enable register to produce a service request if bit 7 (OPER) is set in the Status Byte register.

Line 50 terminates the program.

A more practical example of using the service request in a program would be to detect the end of the measurement period and then collect and display results. The following program gives an example of this.

```
10    ! Program name   PROG_S4_1
20    !
30    ! This program demonstrates the use of the service request.  It
40    ! sets up the Error Performance Analyzer system to perform a basic
50    ! error count measurement at 1 GHz and uses the service request
60    ! to indicate the end of the measurement period.  On recipt of
70    ! the service request the program then collects the error count
80    ! result from the sysytem.
90    !
100  PRINT CHR$(12)              ! Clear the screen.
```

```
110  Error_det=717                ! Assign HP-IB addresses to variables.
120  Pattern_gen=718
130  COM Error_det
140  A=SPOLL(Error_det)
150  ON INTR 7 CALL Init_serv     ! Assign interrupt branch.
160  ENABLE INTR 7;2              ! Arm the computer to recognize an SRQ.
170  DISP "RE-SETTING ERROR DETECTOR"
180  OUTPUT Error_det;"*RST;*CLS" ! Reset system.
190  DISP "RE-SETTING PATTERN GENERATOR"
200  OUTPUT Pattern_gen;"*RST;*CLS"
210  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH '*RST;*CLS'"
220  DISP "SETTLING TIME 3 SECONDS"
230  WAIT 3                       ! Allow settling time.
240   ! Configure Status registers:
250  OUTPUT Error_det;"STATUS:OPERATION:PTRANSITION 0;NTRANSITION 16"
260  OUTPUT Error_det;"STATUS:OPERATION:ENABLE 16"
270  OUTPUT Error_det;"*SRE 128"
280  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH 'FREQUENCY 1GHZ'"
290  OUTPUT Pattern_gen;"SYSTEM:PTHROUGH 'AMPLITUDE +0DBM;AMPLITUDE:STATE ON'"
300  DISP "WAITING FOR FREQ TO SETTLE"
310  CALL Check_freq
320  DISP "STARTED GATING"
330  OUTPUT Error_det;"GATE:MODE SINGLE"
340  OUTPUT Error_det;"GATE:PERIOD 5;STATE ON"
350  CALL Waiting
360  END
370  SUB Init_serv   ! This is the interrupt service routine which reads the
380                  ! Status Byte, clearing the SRQ, and then reads the
390                  ! Operation Status register to confirm that the measurement
400                  ! has ended.  On confirming this, it then gets the results.
410     COM Error_det
420     Status_byte=SPOLL(Error_det)
430     OUTPUT Error_det;"STATUS:OPERATION:EVENT?"
440     ENTER Error_det;Event_reg
450     IF BIT(Event_reg,4) THEN
460        DISP "SRQ Received - fetching results"
470        BEEP 600,1.5
480        CALL Fetch_results
490     ELSE
500        DISP "Result Invalid"
510     END IF
520     DISP " "
530  SUBEND
540  SUB Fetch_results   ! This routine simply displays error count and ratio.
550    COM /Eom/ Measurement_end
560     COM Error_det
570     PRINT CHR$(12)
580     OUTPUT Error_det;"FETCH:ECOUNT?"
590     ENTER Error_det;Error_count
600     OUTPUT Error_det;"FETCH:ERATIO?"
610     ENTER Error_det;Error_ratio
620     PRINT "Error Count is",Error_count
630     PRINT "Error Ratio is",Error_ratio
640     Measurement_end=1
650  SUBEND
```

```
660  SUB Check_freq   ! This subroutine ensures the received frequency from
670 Read_freq:        ! the signal Generator is stable.
680     COM Error_det
690     OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
700     ENTER Error_det;Frequency1
710     DISP "FREQUENCY =";Frequency1
720     WAIT 1
730     OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
740     ENTER Error_det;Frequency2
750     DISP "FREQUENCY =";Frequency2
760     IF Frequency1<>Frequency2 THEN Read_freq
770  SUBEND
780  SUB Waiting
790  COM /Eom/ Measurement_end
800  Elapsetime=TIMEDATE
810 Wait_loop:       !  This routine simply marks time.
820  DISP "RUNNING TIME SO FAR = ";TIMEDATE-Elapsetime
830  IF Measurement_end THEN SUBEXIT
840  GOTO Wait_loop
850  !
860  SUBEND
```

Figure 3-10. Service Request Example Program

Service Requests and Master mode

Setting a master instrument to generate a service request is a straightforward exercise and the principle is the same in each case.

When a service request occurs, tracing the source of the service request requires that the Status Byte in each instrument be tested.

Service Requests and Slave mode

To set a slave instrument to generate a service request requires that the instruments registers be accessed using the SYSTEM:PTHROUGH command. For more information on the SYSTEM:PTHROUGH command refer to Section 2.

Tracing the source of a service request in a slave instrument is a little different. If a slave instrument issues a service request it not only sets bit 6 (RQS bit) of its own Status Byte register, it also sets a bit 1 of its masters Status Byte register. Bit 1 in the Pattern Generator's Status Byte register is designated the slave service bit and it indicates that the slave Signal Generator instrument has requested service.

4

System Command Reference Section

About this Chapter

This chapter describes the command syntax of the pattern generator and error detector.

Syntax Diagrams

Syntax diagrams accompany the description of each root command. Commands in solid boxes are required commands and those in dashed boxes are implied commands. Parameters in solid boxes are required parameters and those in dashed are optional parameters.

Implied Commands

Implied commands appear in dashed boxes in the syntax diagrams and in square brackets in the text. If a command immediately preceding an implied command is used, the system assumes the implied command is also being used and expects the required parameters of the implied command.

Query Commands

Any value that can be set can also be queried, therefore the query form of each command is not shown in the syntax diagrams or in the text. Where a command ending in a question mark does appear, it is a query only command.

Command Abbreviations

Commands may be up to twelve characters long but a short-form version is also available which has a preferred length of four characters. In this document the long-form and short-form versions are shown as a single word with the short-form being shown in upper-case letters. For example, the long-form node command `SOURce` has the short-form `SOUR`.

Note

Accessing large patterns can take several minutes. Control programs must be prepared for I/O timeouts of this order.

Instrument Configuration

The pattern generator and error detector that comprise the system can be configured as separate modules or as a composite instrument with the pattern generator as a slave to the error detector. This configuration is done by setting the addresses of the two modules appropriately[3].

There is a `SOURce` node for each of the outputs from the pattern generator and a `SENSe` node for each of the inputs to the error detector. These nodes are differentiated from each other by adding a number to the end of the node name, thus the data output is `SOURce1`, the clock output `SOURce2` etc.

SCPI requires the node `SOURce` to be allowed as an alias for `SOURce1`. SCPI permits instruments that are primarily sources to allow the `SOURce` node to be optional, similarly instruments that are primarily sensors may allow the `SENSe` node to be optional.

Behavior at Power On

At power-on, the state of the registers and filters will be:

In normal operation,	the enable state of the registers and transition filters will be preserved through a power fail.
On virgin power-on,	all registers and filters are disabled except: 1) the PON, CME and EXE bits of the Standard Event Status Register and its summary bit in the Status Byte, 2) all the assigned bits of the FAILURE register and its summary bit in the Status Byte. In this way, the naive user will not be swamped by SRQs. An SRQ will only be generated if the instrument receives invalid commands or queries, or a major hardware failure occurs. The transition filters will be set to allow all conditions and events to pass.

The event registers and the error queue are cleared at each and every power-up.

Pattern Generator Commands

The pattern generator is primarily a source device. There is one source node for each of the outputs present; they are numbered as follows:

`SOURce1`	*The data output. This node may also be referred to as* `SOURce`.
`SOURce2`	*The clock output.*
`SOURce3`	*The trigger output.*

SOURce1 - The Data Source

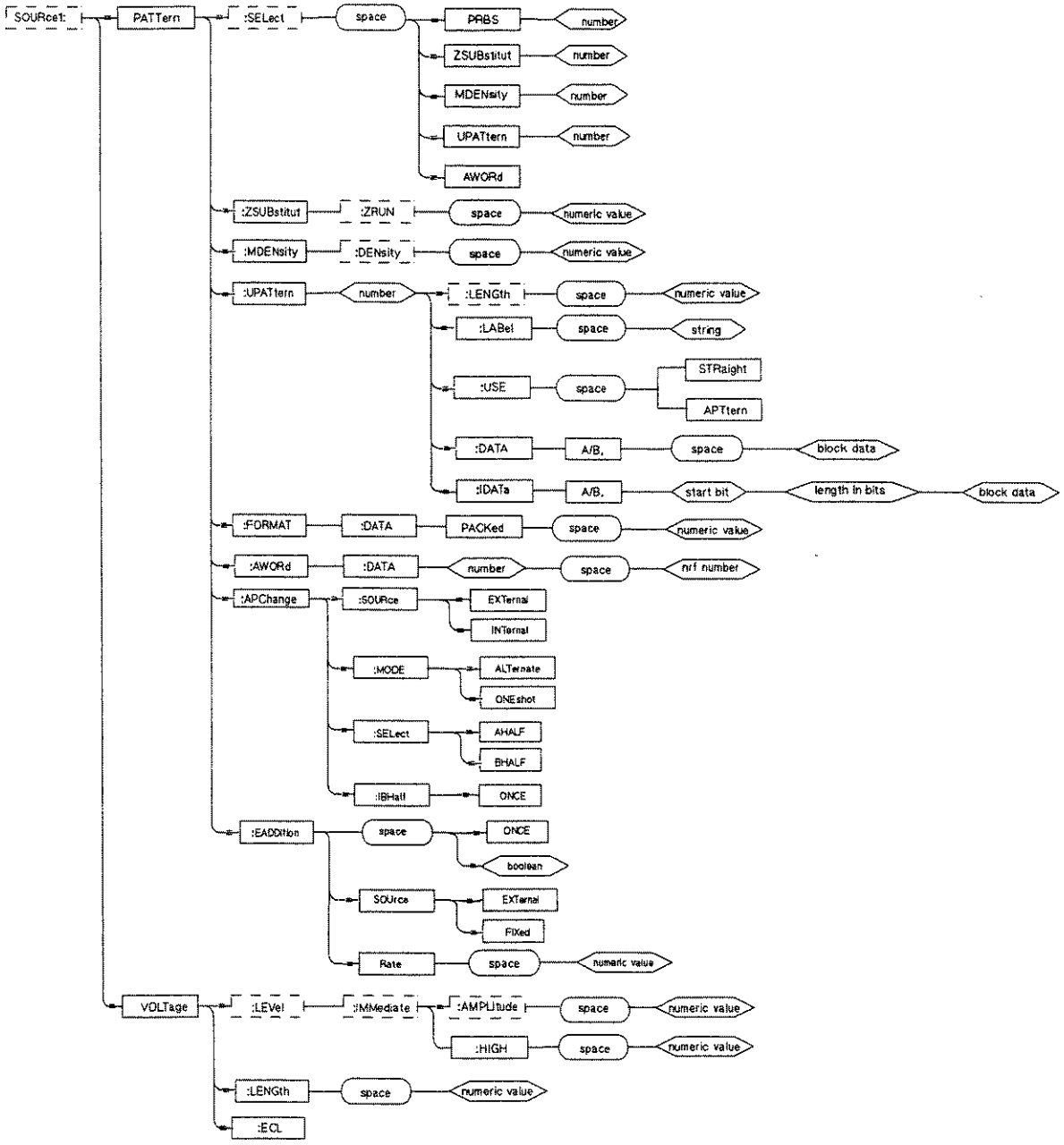


Figure 4-1. SOURce1 Syntax Diagram

:PATTern

This node selects and defines the attributes of the pattern being generated.

[:SELect] <character data>

This node defines the type of pattern being generated. The <character data> is one of:

PRBS<n>	*<n> = 7,10,15,23 or 31*
ZSUBstitut<n>	*<n> = 7,10,11, or 13*
MDENsity<n>	*<n> = 7,10,11, or 13*
UPATtern<n>	*<n> = 0,1,2,3,4,5,6,7,8,9,10,11, or 12*
AWORd	

Note that if a user pattern is selected and the [:SELECT]? command is used, then the response is "UPAT"; no <n> is appended.

The *RST selection is PRBS23.

:ZSUBstitut

This is a contraction of the phrase: *Zero SUBstitution* and is used for defining patterns in which a block of bits is replaced by a block of zeros.

[:ZRUN] <numeric value>. This is a contraction of the phrase: *Zero RUN*, and is the length, in bits, of the longest run of zeros in the pattern. The zeros that are added for the Zero Substitution function replace the bits that immediately follow this longest run of zeros and the length of the overall block of zeros is the value set by the ZRUN command. The range of values is:

MINimum	*The length of the longest run of zeros in the unmodified pattern. (eg for a 2^7 pattern this value is 7.)*
MAXimum	*The length of the pattern minus one.*

The *RST selection is 13.

:MDENsity

This is a contraction of the phrase: *Mark DENsity* and is used for defining a pattern in which the density of marks may be set by the user.

[:DENSity] <numeric value>. Sets the density of marks in the output pattern. The mark density may be varied in eighths, from one to seven eighths, (but excluding $\frac{3}{8}$ and $\frac{5}{8}$).

The *RST selection is $\frac{4}{8}$.

:UPATtern<n>

This is a contraction of the phrase: *User PATtern* and is used to define the contents of a pattern store. The value <n> must be in the range 1 through 12.
The commands under this node affect the storage of information as defined in the table below:

<n> = 0	Current pattern
<n> = 1 thru 4	Non-volatile RAM storage
<n> = 5 thru 12	Disc storage

[:LENGth] <numeric value>. This command sets the length of the pattern that is to be generated. When an alternate user-defined pattern is selected, the :LENGth refers to each half of the pattern. The pattern length has the following constraints:

1. 1 bit to 32 kbits in 1 bit steps,
2. 32 kbits to 64 kbits in 2-bit steps,
3. 64 kbits to 128 kbits in 4-bit steps,
4. 128 kbits to 256 kbits in 8-bit steps,
5. 256 kbits to 512 kbits in 16-bit steps,
6. 512 kbits to 1 Mbits in 32-bit steps,
7. 1 Mbit to 2 Mbits in 64-bit steps,
8. 2 Mbits to 4 Mbits in 128-bit steps,

The *RST command leaves this selection unchanged.

Note

Accessing large patterns can take several minutes. Control programs must be prepared for I/O timeouts of this order.

:LABel <string>. Defines a character string of up to 14 characters that is associated with the pattern. This is to make it easy for the user to comprehend the purpose of the particular pattern without having to refer to a lookup table.

The character data values of MINimum, MAXimum and DEFault are not defined for the label.

The *RST command leaves this selection unchanged.

:USE STRaight|APATtern. Defines the use of a user-defined pattern. When STRaight is selected the whole of the pattern is repeatedly output. When APATtern is selected the pattern is considered to be composed of two halves. The ":APCHange" command controls how these two halves are output.

The "USE" command also resets the selected pattern store. If "USE STRaight" is used, the store is set to have a length of 1 bit; this data bit is set to zero and the trigger bit is set to zero. If "USE APATtern" is used, the store is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A-B changeover.

For user-patterns used in the *STRaight* mode, the recommended sequence of issuing commands is:

PATTern:UPATtern<n>:USE	STRaight
PATTern:UPATtern<n>[:LENGth]	<numeric value>
PATTern:UPATtern<n>:DATA	<block data>
SOURce3:TRIGger:UPATtern<n>	<numeric value>

For user-patterns used in the *APATtern* mode, the recommended sequence of issuing commands is:

PATTern:UPATtern<n>:USE	APATtern
PATTern:UPATtern<n>[:LENGth]	<numeric value>
PATTern:UPATtern<n>:DATA	A,<block data>
PATTern:UPATtern<n>:DATA	B,<block data>
SOURce3:TRIGger:UPATtern<n>	ABCHange|SOPattern

The *RST command leaves this selection unchanged.

:DATA [A|B,] <block_data>. Sets the bits of the pattern. The bits are sent as an arbitrary block diagram data element. The data may be sent 1 bit/byte or 8 bits/byte, under the control of the :FORMAT[:DATA] command. If 1 bit/byte is selected numeric values of either binary 1 or binary 0 only are allowed. If 8 bits/byte is selected the left-most bit of the first byte received forms the first bit of the pattern.

If ":USE APATtern" is selected, then the first parameter indicates which half pattern is to receive the data. If "USE STRaight " is selected, either "A" or no first parameter are acceptable.

The length of the <block data> embedded in the header refers always to the length in bytes irrespective of the current setting of the [:DATA] PACKed, <numeric value> command.

The character data values of MINimum, MAXimum and DEFault are not defined for the data.

To be consistent with the behavior of the pattern editor, more bits may be sent than are specified by the "LENGth" command, in which case the extra bits will be ignored and will not appear as part of the pattern. If the pattern length is subsequently extended the extra bits are filled with zeros. If fewer bits than specified by the "LENGth" command are sent, then the bits in the store beyond the length sent remain unchanged.

The pattern stores 1 through 4 have an overall length of 8192 bits, and pattern store 0 and 5 through 12 have an overall length of 4,194,304 bits

The following rules apply:

1. If 'PATTern:FORMat PACKed,1' is selected and data is sent with the ':UPATtern:DATA' command, then:
   block length = pattern length
2. If 'PATTern:FORMat PACKed,1' is selected and data is sent with the ':UPATtern:IDATa' command, then:
   block length = number of relevant bits in block
   (start bit + block size) <= pattern length
   block size >= 1
3. If 'PATTern:FORMat PACKed, 8' is selected and data is sent with the ':UPATtern:DATA' command, then:
   block size =((pattern length - 1) DIV 8) + 1
4. If 'PATTern:FORMat PACKed,8' is selected and data is sent with the ':UPATtern:IDATa' command, then:
   block size = ((number of relevant bits in block - 1) DIV 8) + 1
   (start bit + block size) = ((pattern length - 1) DIV 8 + 1) * 8
   block size >= 1

An arbitrary block program data element is a method of sending large quantities of data from a controller to an instrument. It comes in two forms; an *indefinite length* format when the length of the transmission is not known, and a *definite length* format when the length is known. In the application here, the *definite length* format is used.

A definite length arbitrary block program data element is composed of two parts; a header and the data itself. The header is made up from three parts:

1. The first part is the ASCII character #.
2. The second part is a single non-zero ASCII digit. The magnitude of this digit equals the number of digits in the third part of the header.
3. The third part is composed of between 1 and 9 ASCII digits. The value of these digits taken together as a decimal integer equal the number of 8-bit data bytes which follow.

The data part is composed of a number of 8-bit data bytes.

As an example, if a user-pattern of length 7986 bits is to be set up, then the header would be #47986.

The *RST command leaves this selection unchanged.

:IDATa [A|B,] <start_bit>, <length_in_bits>, <block_data>

This command is similar to the :DATA command. The header is short for Incremental Data and the command is used to download just part of a user-defined pattern.

If ":USE APATtern" is selected, then the first parameter indicates which half pattern is to receive the data. If "USE STRaight " is selected, either "A" or no first parameter are acceptable.

The length of the <block data> embedded in the header refers always to the length of the data in bytes.

The first parameter defines the starting position within the overall pattern of the first bit of the transmitted pattern. The first bit is counted as bit zero. The second parameter defines how many bits are to be transmitted and the third parameter provides the data itself.

The query form of the command is of the format ":IDATa? <start bit> ,<length in bits>". The second parameter defines the length (in bits) of the data block to be output.

Example 1——Use of the :DATA command

Set user-defined pattern store 5 to a length of 9 bits. Let the new data bits be 1, 0, 0, 1, 1, 0, 1, 1, 1 (binary). Then query the contents of this pattern store.

```
Method 1:  using data packed 1 bit per byte.


     PATT:FORM         PACK,1
     PATT:UPAT5        9
     PATT:UPAT5:DATA   #19<data>

        where  # = the start of the header
               1 = the number of decimal digits to follow forming the length
               9 = the length of the data block that follows
         <data> = 9 data bytes containing binary 00000001
                                                 00000000
                                                 00000000
                                                 00000001
                                                 00000001
                                                 00000000
                                                 00000001
                                                 00000001
                                                 00000001


     PATT:UPAT5:DATA?

        would return   #19<data>

        where  # = the start of the header
               1 = the number of decimal digits to follow forming the length
               9 = the length of the data block that follows
         <data> = 9 data bytes containing binary 00000001
                                                 00000000
                                                 00000000
                                                 00000001
                                                 00000001
                                                 00000000
                                                 00000001
                                                 00000001
                                                 00000001
```

Method 2: using data packed 8 bits per byte.

```
PATT:FORM         PACK,8
PATT:UPAT5        9
PATT:UPAT5:DATA   #12<data>
```

where # = the start of the header
1 = the number of decimal digits to follow forming the length
2 = the length of the data block that follows
<data> = 2 data bytes containing binary 10011011 and 1xxxxxxx

```
PATT:UPAT5:DATA?
```

would return #12<data>

where # = the start of the header
1 = the number of decimal digits to follow forming the length
2 = the length of the data block that follows
<data> = 2 data bytes containing binary 10011011 and 10000000

Example 2: Use of the :IDATa command

Update 9 bits of store number 5 starting at bit 3. Let the new data bits be 1, 0, 0, 1, 1, 0, 1, 1, 1 (binary). Then query these 9 bits.

```
Method 1:  using data packed 1 bit per byte.

      PATT:FORM         PACK,1
      PATT:UPAT5:IDAT   3,9,#19<data>

         where  3 = the start bit
                9 = the number of bits
                # = the start of the header
                1 = the number of decimal digits to follow forming the length
                9 = the length of the data block that follows
          <data> = 9 data bytes containing binary 00000001
                                                  00000000
                                                  00000000
                                                  00000001
                                                  00000001
                                                  00000000
                                                  00000001
                                                  00000001
                                                  00000001

      PATT:UPAT5:IDAT?  3,9

         would return   #19<data>

         where  # = the start of the header
                1 = the number of decimal digits to follow forming the length
                9 = the length of the data block that follows
          <data> = 9 data bytes containing binary 00000001
                                                  00000000
                                                  00000000
                                                  00000001
                                                  00000001
                                                  00000000
                                                  00000001
                                                  00000001
                                                  00000001
```

```
Method 2:  using data packed 8 bits per byte.

      PATT:FORM        PACK,8

      PATT:UPAT5:IDAT   3,9,#12<data>

         where  3 = the start bit
                9 = the number of bits
                # = the start of the header
                1 = the number of decimal digits to follow forming the length
                2 = the length of the data block that follows
          <data> = 2 data bytes containing binary 10011011 and
                                                  1xxxxxxx


      PATT:UPAT5:IDAT?  3,9

         would return   #12<data>

         where  # = the start of the header
                1 = the number of decimal digits to follow forming the length
                2 = the length of the data block that follows
            <data> = 2 data bytes containing binary 10011011 and
                                                  10000000
```

:FORMat:

This command controls the format of data transfer for the :PATTern:UPATtern<n>:DATA and :PATTern:UPATtern<n>:IDATa commands.

[:DATA] PACKed,<numeric value>. This command permits the packing of bits within a byte to be set. The First parameter must be *PACKed*. The <numeric value> parameter may be set to either 1 or 8.

Note that if a user pattern is selected and the [:SELect]? command is issued, then the response if "UPAT"; no '<n>' is appended.

The *RST selection is "PACKed,1".

:AWORd

This is a contraction of the phrase: *Alternate WORd* and is used to set the pattern used by the two alternating words.

:DATA<n> <NRf>{,<NRf>}. Since the length of the alternate word is fixed at 16, then failure to send exactly sixteen parameters will cause an error. The value <n> may be either 0 or 1, and corresponds to the word that is emitted and the value of the " AUX INPUT" input that controls it.

The *RST selection is 0101010101010101 0000000011111111.

:APCHange

This is a contraction of the phrase *alternate pattern change* and is used to control how user-defined patterns are output when set to be used as alternate patterns.

:SOURce EXTernal|INTernal. This command control the source of control for the alternate pattern output. When EXTernal is selected the pattern is controlled by the rear-panel Auxiliary Input socket. When INTernal is selected the pattern is controlled by the user, either from the front-panel or from HP-IB using other commands from within this group.

The *RST selection is "EXTernal".

:MODE ALTernate|ONEShot. This command controls the mode of operation of the alternate pattern output. If ALTernate is selected and the source is set to EXTernal, then the polarity of the signal at the Auxiliary Input socket governs which half of the pattern is output. If the source is set to INTernal, then the :APCHange:SELect command control which half of the pattern is output.

If the MODE is set to ONEShot and the source is set to EXTernal, then one instance of half B of the pattern is output for each rising edge of the Auxiliary Input. If the source is set to INTernal, then the :APCHange:IBHalf command is used to insert one instance of half B of the pattern.

The *RST selection is "ALTernate".

:SELect AHALf|BHALf. This command controls whether half A or half B of the alternate pattern is output. It is valid only when :APCHange:SOURce is set to INTernal and :APCHange:MODE is set to ALTernate.

The *RST selection is "AHALf".

:IBHalf ONCE:. This command is short for *Insert B Half*. It causes the single insertion of a number of instances of half B of the alternate pattern to be inserted. It is valid only when :APCHange:SOURce is set to INTernal and :APCHange:MODE is set to ONSHot. It is an event command, and as such has no query form. The number of half 'B' insertions is equal to the smallest integral multiple of the pattern length that divides exactly by 128.

:EADDition ONCE|<boolean>

This is a contraction of the phrase: *Error ADDition* and is used to control the addition of errors into the generated pattern. The parameter ONCE causes a single bit error to be added to the pattern. It also turns off the constant rate error addition.
A boolean parameter enables/disables the addition of errors at a fixed rate.

The *RST selection is OFF.

:SOURce EXTernal|FIXed. This command controls the source of injected errors. When set to EXTernal (and :EADDition[:STATe] is ON), each pulse at the External Errors socket causes an error to be added to the data stream. When set to FIXed (and :EADDition[:STATe] is ON), repetitive errors are internally added to the data stream. The rate of error addition is controlled by the :EADDition:RATE command.

The *RST selection is "FIXed".

:RATE <numeric value>:. This command controls the rate of internal, fixed error addition. Values between 1e-3 and 1e-9 in decade steps are permitted.

The *RST selection is 1e-6.

:VOLTage

This node sets the values of the data output electrical levels.

[:LEVel][:IMMediate][:AMPLitude] <numeric value>

Sets the peak to peak value of the data signal, in units of Volts.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 500 mV.

[:LEVel][:IMMediate]:HIGH <numeric value>

This is used to set the DC high output level, in units of Volts.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 0 V.

:ATTenuation_<numeric_value>

Specifies, in decibels, the value of external attenuation on the output. This causes the entered/displayed values to be modified so as to reflect the value of the output on the far side of the attenuator.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 0 dB.

:ECL

Sets the output AMPLitude and HIGH values to those used for the ECL family.

There is no query form of this command.

SOURce2 - The Clock Source

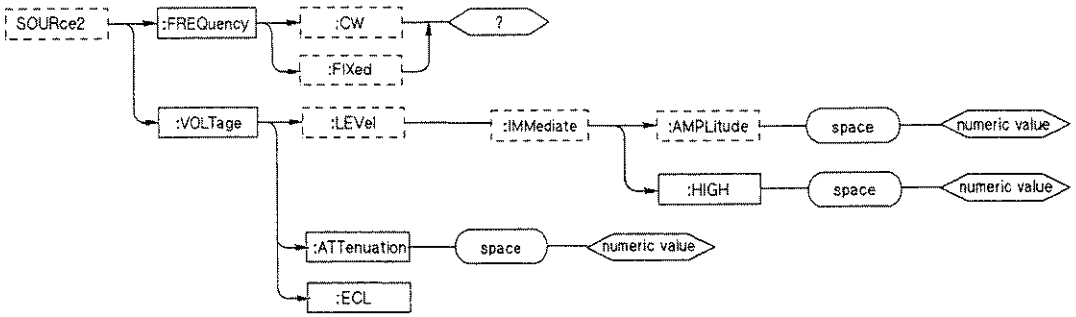


Figure 4-2. SOURce2 Syntax Diagram

:FREQuency[:CW|:FIXed]? <numeric value>

Queries the bit rate of the measured clock frequency at the input of the Pattern Generator.

:VOLTage

This node sets the values of the clock output electrical levels.

[:LEVel][:IMMediate][:AMPLitude] <numeric value>

Sets the peak to peak value of the clock signal, in units of Volts.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 500 mV.

[LEVel][:IMMediate]:HIGH <numeric value>

This is used to set the DC high output level, in units of Volts.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 250 mV.

:ATTenuation <numeric value>

Specifies, in decibels, the value of external attenuation on the output. This causes the entered/displayed values to be modified so as to reflect the value of the output on the far side of the attenuator.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 0 dB.

:ECL

Sets the output "AMPLitude" and "HIGH" values to those used for the ECL family. There is no query form for this command.

SOURce3 - The Trigger Source

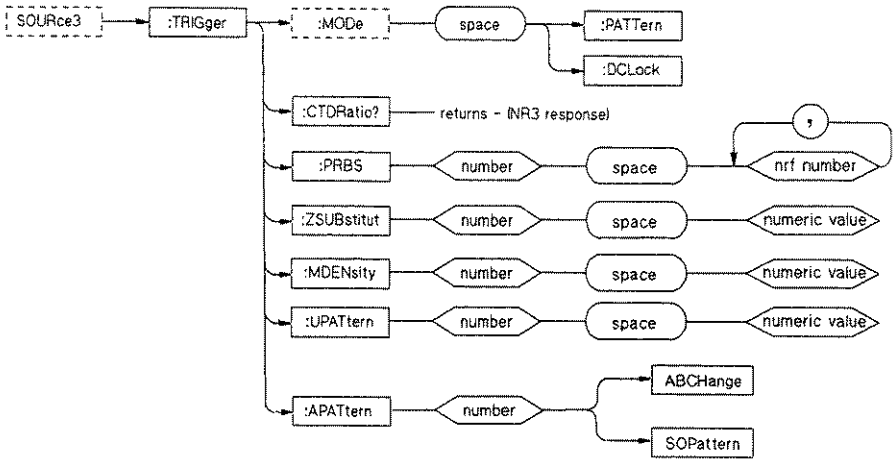


Figure 4-3. SOURce3 Syntax Diagram

:TRIGger

This node permits the attributes of the trigger output to be set up.

[:MODe] PATTern|DCLock

This node is for specifying the mode of the trigger output. The possible modes are:

`Pattern`	*The trigger pulse is output coincident with the occurrence, in the data output stream, of a particular pattern of bits.*
`Divided Clock`	*The trigger pulse is simply the input data clock divided by a fixed value.*

The `*RST` selection is PATTern.

:CTDRatio? <NR3>. The command is short for Clock to Trigger Division ratio. It gives the ratio between the frequency of the clock output and the frequency of the pulses on the trigger output for the currently selected pattern.

If alternate patterns are selected and the trigger is set to occur *on input*, then no division ratio is available and this command responds with Not-A-Number (NAN, 9.91xE+37).

:PRBS<n> <NRf>{,<NRf>}

This command sets the pattern, the occurrence of which causes a trigger pulse to be output. The number `n` must one of `7, 10, 15` or `23`. The number of parameters depends on the pattern length, and is the minimum that can define a unique place in the overall pattern, for example a pattern of length 2^{n-1} the number of parameters is `n`. The parameter values are either 1 or 0. An *all-ones* pattern is disallowed.

The `*RST` selection is ALL ZEROS for n = 1 through 4.

:ZSUB<n> <numeric value>

This command selects the position within the PRBS at which the trigger pulse is to be output whenever a Zero Substitution PRBS is selected. The number 'n' must be one of 7, 10, 11 and 13. The parameter must be in the range 0 through (pattern length - 1).

The *RST selection is 0 for n = 7, 10, 11 and 13.

:MDEN<n> <numeric value>

This command selects the position within the PRBS at which the trigger pulse is to be output whenever a Mark Density PRBS is selected. The number 'n' must be one of 7, 10, 11 and 13. The parameter must be in the range 0 through (pattern length - 1).

The *RST selection is 0 for n = 7, 10, 11 and 13.

:UPAT<n> <numeric value>

This command selects the position within the PRBS at which the trigger pulse is to be output whenever a Zero Substitution PRBS is selected. The number 'n' must be in the range 0 thru 12. The parameter must be in the range 0 through (pattern length - 1).

The commands under this node affect the storage of information as defined in the following table:

<n> = 0	Current pattern
<n> = 1 thru 4	Non-volatile RAM storage
<n> =5 thru 12	Disc storage

The *RST selection is 0 for n = 0 through 4.

:APATtern<n> ABCHange|SOPpattern. This command control the trigger output when an alternate pattern is selected for output. If SOPpattern (short for Start Of Pattern) is selected, then a trigger pulse is output at the start of the pattern. If ABCHange (short for A-B change) is selected, then the trigger output changes as the alternate halves change.

The commands under this node affect the storage of information as defined in the table below:

<n> = 0	Current pattern
<n> = 1 thru 4	Non-volatile RAM storage
<n> = 5 thru 12	Disc storage

The *RST selection is ABCHange.

OUTPut1 - The Data Output

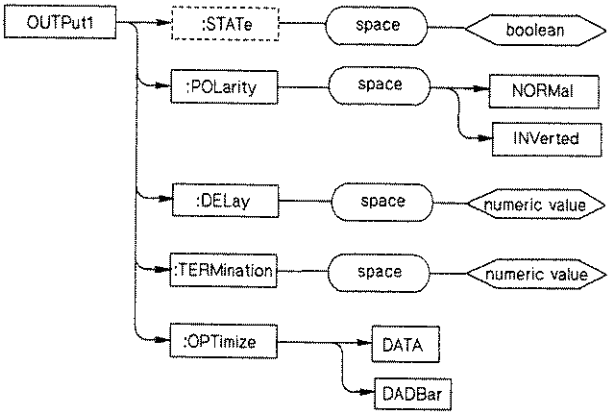


Figure 4-4. OUTPut1 Syntax Diagram

This node is used to control electrical characteristics of the data output.

[:STATe] <boolean>

This node controls the data output. When OFF, the output is set to 0V.

The *RST selection is ON.

:POLarity NORMal|INVerted

Sets the polarity of the data output.

The *RST selection is NORMal.

:DELay <numeric value>

Sets the delay of the active edge of the clock output relative to the data output. The units are seconds. The value is rounded to the nearest one picosecond.

The *RST selection is 0 ps.

:TERMination <numeric value>

Enables the data termination level to be selected as 0 Volts or −2 Volts.

See the section entitled *Handling Coupled Parameters*.

The *RST selection is 0 V.

:OPTimize DATA|DADBar

This node controls the optimization used for the data output signal eye crossover. When DATA is selected the crossover symmetry is optimized for the normal data signal alone. When DADBar (short for data and data bar) is selected the crossover symmetry is optimized as a compromise between the normal and inverted signals.

The *RST selection is DATA.

OUTPut2 - The Clock Output

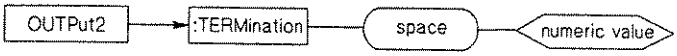


Figure 4-5. OUTPut2 Clock Output

This node is used to control electrical characteristics of the clock output.

:TERMination <numeric value>

Enables the clock termination level to be selected as 0 Volts or −2 Volts.
See the section entitled *Handling Coupled Parameters*.
The *RST selection is 0 V.

MMEMORY

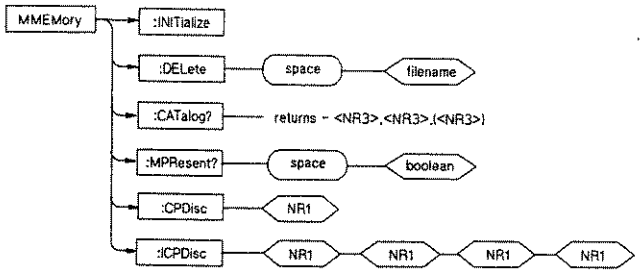


Figure 4-6. Memory

This subsystem is used for controlling the floppy disc used as mass memory with the instrument. It is recommended that a floppy disc is reserved for sole use by the HP 70841B.

Note: disc accesses by a slaved pattern generator should not be performed whilst the master error detector is gating.

INITialize

The INITialize command is used to initialize the floppy disc mass storage medium. Both 720 kbyte DOS and 1400 kbyte DOS formats are supported. The instrument automatically detects the disc capacity and formats accordingly.

This command is an event and has no *RST condition.

DELete <file name>

The DELete command removes a file from the floppy disc. The <file name> parameter specifies the file name to be removed. It is a string parameter. File names are 'HPPATxx.DAT,' where xx ranges from 05 through 12. For example to delete disc pattern 7. the command would be "MMEM:DEL 'HPPAT 07.DAT'".

This command is an event and has no *RST condition or query form.

CATalog? <NR3>,<NR3>{,<file entry>}

The CATalog? command is query-only and returns information on the current contents and state of the floppy disc. Upon a CATalog? query, the instrument reads the floppy disc and returns its directory information. The information returned is composed of two numeric parameters followed by as many strings as there are files in the directory list. The first parameter indicates the total amount of storage currently used in bytes. The second parameter indicates the total amount of storage available, also in bytes. The <file entry> is a string. Each <file entry> indicates the name, type and size of one file in the directory list:

<file name>,<file type>,<file size>

The <file size> is returned in bytes. The number of <file entry> items that is returned is limited to eight.

MPResent? <boolean>

This command is short for "Media Present". It returns a boolean indicating whether a floppy disc is present.

CPDisc <NR1>

The mnemonic ICPDisc is short for *Copy Pattern to Disc*. The parameter provides the destination store number, and must be between 5 and 12 inclusive.

ICPDisc <NR1>,<NR1>,<NR1>,<NR1>

The mnemonic ICPDisc is short for *Incremental Copy Pattern to Disc*. It is used to copy just a portion of the current edit buffer to disc. If used on alternate patterns then the pattern half needs to be specified. The four parameters are:

Parameter No. 1:	The destination store number, between 5 and 12.
Parameter No. 2:	The pattern half;
	For a straight pattern= 0
	For an alternate pattern= 0 for half A and 1 for half B.
Parameter No. 3:	The first bit of the block to copy to disc.
Parameter No. 4:	The last bit of the block to copy to disc.

SYSTem

This subsystem is mostly defined by SCPI for functions that are not related to instrument performance.

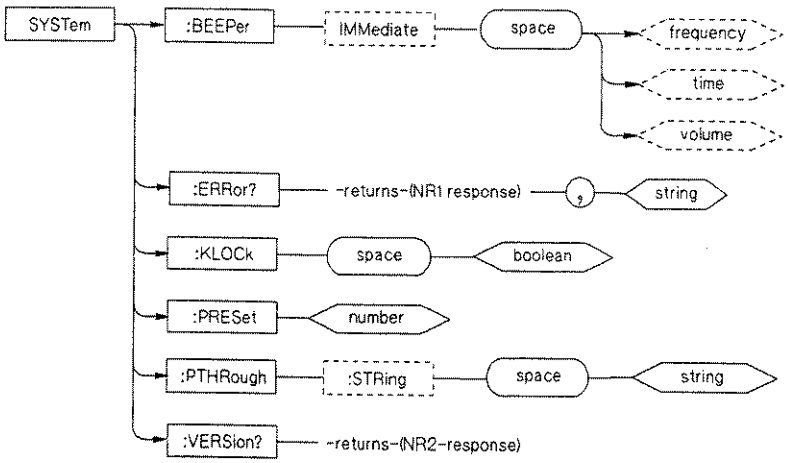


Figure 4-7. SYSTem Syntax Diagram

:BEEPer

This subsystem controls the audible beeper.

The `:BEEPer` command produces a beep only if a Display unit is connected and the Pattern Generator has been assigned a window.

[:IMMediate] [<freq>[,<time>[,<vol>]]]

Causes an audible tone to be generated. The optional parameters `<freq>`, `<time>` and `<vol>` are intended (in SCPI) to set the frequency, duration and volume of the beep; however, they are not implemented. A fixed beep is always generated.

The pattern generator accepts these parameters, but ignores them.

There is no query form of this command.

:ERRor? <NR1>,<string>

This query-only command will pull the next error from the error queue, and return the error number and a string describing the error. The error queue is of depth ten.

Note

SCPI-defined errors are all negative. The positive error numbers are device defined. The SCPI Messages section at the rear of this manual describes the SCPI-defined errors and Appendix C of the Operating Manual contains details of the device-defined errors.

:KLOCk <boolean>

This locks the instrument's keyboard. When locked, the user may not modify any of the instrument's configuration; although those keys that merely affect the display are still usable.

The *RST selection is OFF.

:PRESet|:PRESet<n>

Sets the pattern generator to a pre-defined *local operation* state. The choice of <n> is 0 through 2. PRESet and PRESet 0 both have the same effect as the front-panel (INSTR PRESET) key. PRESet 1 and PRESet 2 have the same effect as the front-panel recall setup, PRESET 1 and PRESET 2 keys respectively.

:PTHRough

The Pass-Through command allows a remote programming command to be passed through an MMS master module to a slave module. All valid remote commands for the slave can be passed through in this way. The command intended for the slave is given as a string parameter to the PTHRough command. The master pays no attention to the contents of this parameter.

[:STRing] <string>

The <string> parameter is passed through in its entirety to an MMS slave. The format of the parameter must be identical to the HP-IB command that would have been given to the slave had it been setup as an independent master. If the slave module detects an error in the <string> parameter passed through to it, it will generate an error in its error queue and set bit 1 of the master's Status Byte.

If no slave is present when the pass-through command is received, the error "slave needs service" is generated.

Example:

To set the frequency of an HP 70312A clock source (slaved to a master pattern generator) to 2.5 GHz, the following command could be issued:

```
OUTPUT pgen ; "SYST:PTHR ""FREQ 2.5 GHZ"""
```

Note

The string parameter itself needs to be enclosed within quotation marks. Since the whole output string is already enclosed within double quotation marks, then the inner-most set must be represented by a quadruple set. Alternatively, a single set of single quotation marks may be used:

```
OUTPUT pgen ; "SYST:PTHR  'FREQ 2.5 GHZ'"
```

[:STRing]? <string>?. This command permits a query command to be passed through to the slave. Note that a question mark is needed to terminate the header; and another to terminate the <string> since the string is itself a header when received by the slave.

If no slave is present when the pass-through query command is received, an error is generated and a null string is returned.

Example:

To query the current frequency setting of an HP 70312A clock source, the following commands could be used:

```
OUTPUT pgen ; "SYST:PTHR? 'FREQ?'"
ENTER pgen ; freq
```

Note

If the string parameter sent to the slave contains an error one of the following can occur:

- The slave will generate an error in its error queue and set bit 1 of the master's status byte.
- **For Query commands only:** the instrument can hang-up; no error message is reported. The instrument must be powered down, then powered up, and the string parameter checked/corrected.

:VERSion?

This command queries the version of the SCPI programming Language that the pattern generator conforms to. The command currently returns " 1990.0".

STATus

The status conditions that the pattern generator needs to report are partly covered by the pre-defined status registers of IEEE 488.2 and SCPI.

Two additional status registers covers instrument hardware failure conditions and MSIB slaves.

```
:FAILure
  [:EVENt]?        <NR1>

:SSERvice
  [:EVENt]?        <NR1>
  :ENABle          <NRf>
  :ENABle?         <NR1>
```

Status Byte

If an MMS slave module detects an error that would have caused an SRQ to be generated when operating as an independent master, then bit 1 of the Status Byte of the master MMS module is set. This bit is called 'Slave service'. In this way, the master module can relay the error situation to the external controller. The controller can then serial poll the master, detect that it is the slave module that has an error, and interrogate the slave's Status Byte and error queue.

:OPERation

The OPERation register (and summary bit) is defined by SCPI, but is unused in the pattern generator.

:QUEStionable

The bits in this register indicate that a signal is of questionable quality. The usage by the pattern generator is as follows:

SCPI MEANING	pattern generator usage
0 - VOLTage	-
1 - CURRent	-
2 - TIME	-
3 - POWer	-
4 - TEMPerature	-
5 - FREQuency	-
6 - PHASe	-
7 - MODulation	-
8 - CALibration	-
9 - Instr dependent	Clock loss.
10 - Instr dependent	-

11 - Instr dependent	-
12 - Instr dependent	-
13 - Instr summary	-
14 - Unexpected param	-
15 - Zero	-

:PRESet

The PRESet command is an event that configures the SCPI and device dependent status data structures, such that the device dependent events are reported at a higher level through the mandatory part of the status reporting structures.

The PRESet command affects only the enable register and the transition filter registers for the SCPI mandated and device dependent status data structures. PRESet does not affect either the *status byte* or the *standard event status* as defined by *IEEE 488.2*. PRESet does not clear any of the event registers. The *CLS command is used to clear all event registers in the device status reporting mechanism.

For the device dependent status data structures, the PRESet command sets the enable register to all ones and the transition filter to recognize both positive and negative transitions. For the SCPI mandatory status data structures, the PRESet command sets the transition filter registers to recognize only positive transitions and sets the enable register to zero.

:FAILure

The bits in this register indicate that a major hardware element of the instrument has failed. No capability is provided to query the condition register, setup the enable register, nor setup the positive or negative transition filters. This is because failures within this category are non-recoverable, and as such the enable registers are pre-defined.

The usage by the pattern generator is as follows:

BIT	pattern generator usage
0	ROM failure
1	RAM failure
2	Non-volatile memory corrupt
3	Gate Array failure
4	PIT failure
5	Interface board #1 failure
6	Interface board #2 failure
7	MSIB failure
8	EEPROM failure
9	Undefined
10	Undefined
11	Undefined

12	Undefined
13	Undefined
14	Undefined
15	Undefined

:SSERvice

The bits in this register indicate that an MSIB slave module is requesting service. No capability is provided to query the condition register nor setup the positive or negative transition filters. This is because failures within this category are events.
The usage by the pattern generator is as follows:

```
BIT            pattern generator usage
 0             slaved clock source or signal generator
 1 thru 15     undefined
```

Handling Coupled Parameters

There are two groups of four commands within the pattern generator's command set that are coupled together, and their correct use requires an understanding of this coupling. These commands are:

```
[SOURce[1]:]VOLTage[:LEVel][:IMMediate][:AMPLitude]
[SOURce[1]:]VOLTage[:LEVel][:IMMediate][:HIGH]
[SOURce[1]:]VOLTage[:ATTenuation]
OUTPut[1]:TERMination

[SOURce[2]:]VOLTage[:LEVel][:IMMediate][:AMPLitude]
[SOURce[2]:]VOLTage[:LEVel][:IMMediate][:HIGH]
[SOURce[2]:]VOLTage[:ATTenuation]
OUTPut[2]:TERMination
```

The restrictions on the parameter values that these commands can take and the order in which the commands need to be issued are as follows:

1. If a new value of attenuation is issued, then this needs to be followed with new values for the amplitude and high-level.
2. If a new value of termination is issued, then this needs to be followed with a new value for the high-level.
3. Whilst the attenuation is set to 0 dB, if the termination voltage is set to 0 Volts, then the maximum value of high-level is 1 Volt. If the termination voltage is set to −2 Volts, then the maximum value of the high-level is 0 Volts.
4. Values of high-level below −2 Volts are restricted according to the value of amplitude. This is shown in the accompanying Figure.
5. If new values of amplitude and high-level are issued, then care needs to be exercised as described below. Consider first the legal values that the amplitude and high-level. These are best described with the aid of a diagram.

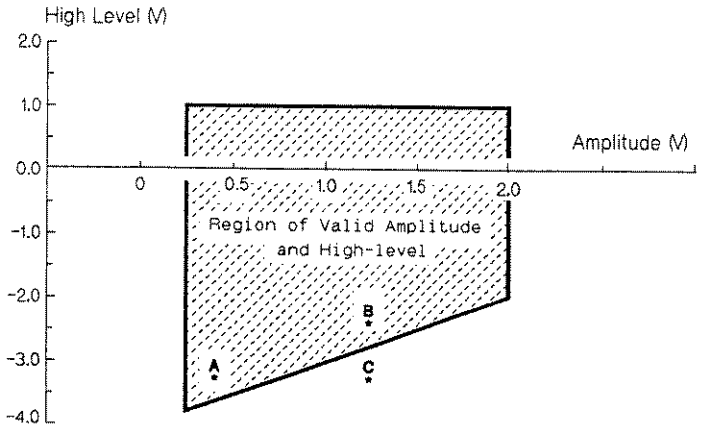


Figure 4-8. Data Amplitude and High-Level Relationship

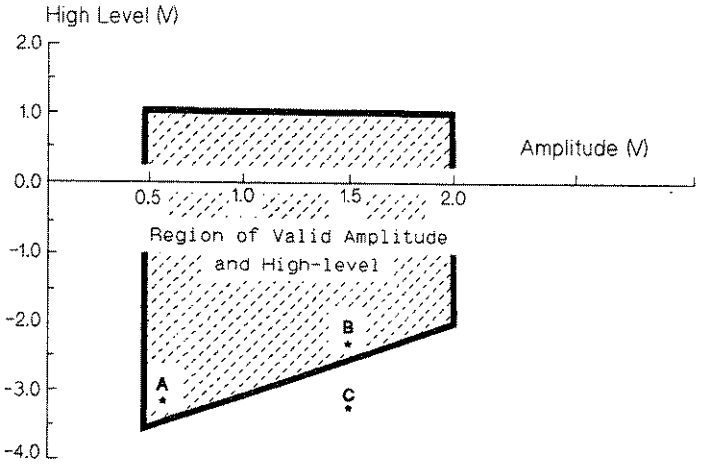


Figure 4-9. Clock Amplitude and High-Level Relationship

The Figures shows the region of valid amplitude and high-level. To move from the point A to the point B, for example, requires some care. This arises because, if the amplitude is first moved, followed by the new value of data high-level, then the intermediate state will be at point C and this would generate an error message.

There are two methods of overcoming this problem:

1. The order of issuing the amplitude and high-level may be used to prevent the bottom sloping line being crossed. The algorithm would be:

   If the new amplitude is greater than the old amplitude, then send the new high-level first, followed by the new amplitude.

   If the new amplitude is smaller than the old amplitude, then send the amplitude first, followed by the high-level.

2. A dummy move of the high-level to a value between 0 V and −2.0 V is followed by the new value of amplitude, followed by the new value of high-level.

Error Detector Commands

The error detector is primarily a sensing device. There is one sense node for each of the inputs present; they are numbered as follows:

`SENSe1`	*The data input. This node may also be referred to as* `SENSe`.
`SENSe2`	*The clock input.*

SENSe1 - The Data Sense

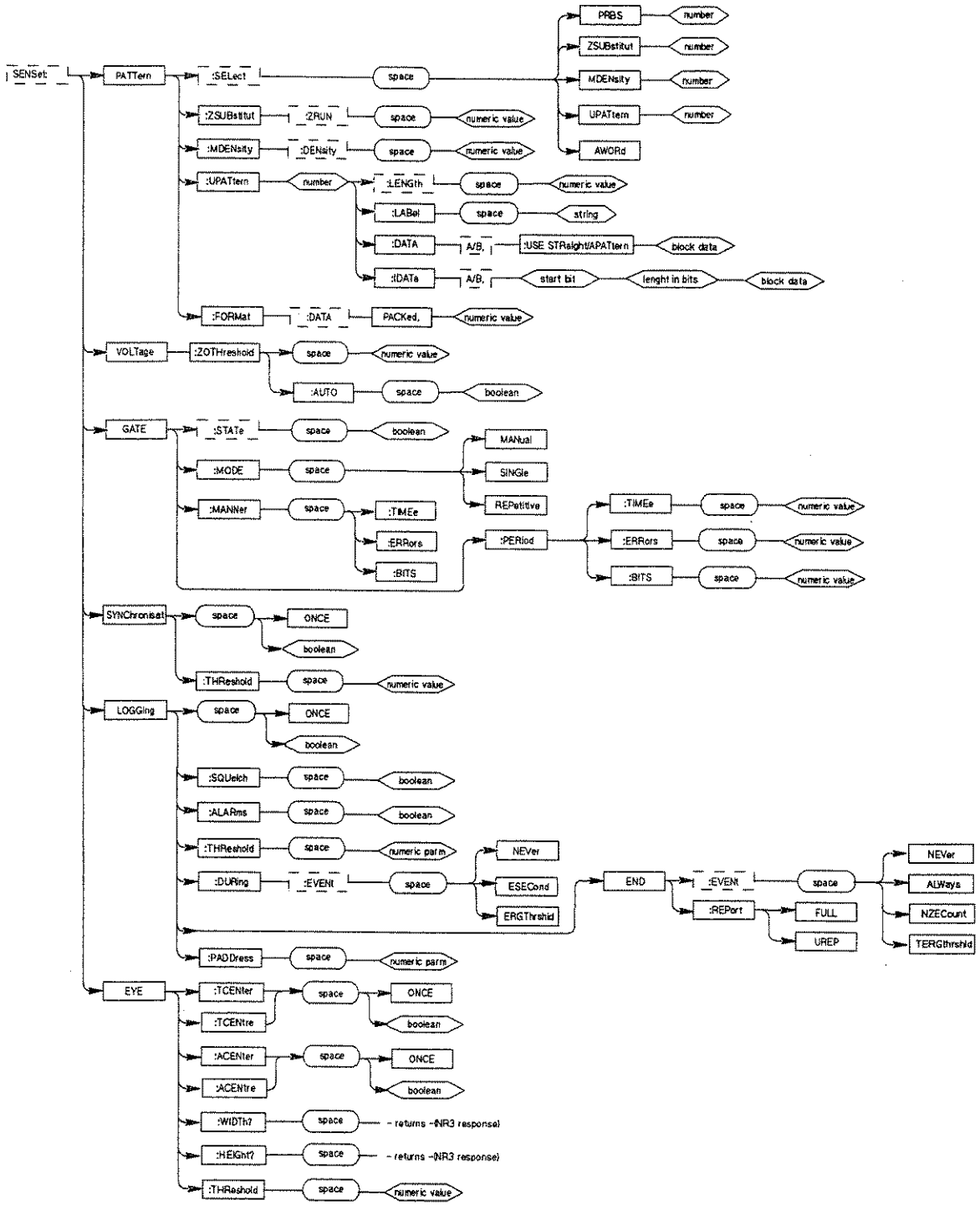


Figure 4-10. SENSe1 Syntax Diagram

:PATTern

This node selects and defines the attributes of the pattern being generated. It is essentially the same as the "PATTern" node in the pattern generator, with the exception that the error detector has neither an Alternate Word mode, nor the error addition capability.

[:SELect] <character data>

This node defines the type of pattern being generated. The <character data> is one of:

PRBS<n>	*<n> = 7,10,15,23 or 31*
ZSUBstitut<n>	*<n> = 7,10,11, or 13*
MDENsity<n>	*<n> = 7,10,11, or 13*
UPATtern<n>	*<n> = 0,1,2,3,4,5,6,7,8,9,10,11, or 12*

Note that if a user pattern is selected and the [:SELECT]? command is used, then the response is "UPAT"; no <n> is appended.

The *RST selection is PRBS23.

:ZSUBstitut

This is a contraction of the phrase: *Zero SUBstitution* and is used for defining patterns in which a block of bits is replaced by a block of zeros.

[:ZRUN] <numeric value>. This is a contraction of the phrase: *Zero RUN*, and is the length, in bits, of the longest run of zeros in the pattern. The zeros that are added for the Zero Substitution function replace the bits that immediately follow this longest run of zeros and the length of the overall block of zeros is the value set by the ZRUN command. The range of values is:

MINimum	*The length of the longest run of zeros in the unmodified pattern. (eg for a 2^7 pattern this value is 7.)*
MAXimum	*The length of the pattern minus one.*

The *RST selection is 13.

:MDENsity

This is a contraction of the phrase: *Mark DENsity* and is used for defining a pattern in which the density of marks may be set by the user.

[:DENSity] <numeric value>. Sets the density of marks in the output pattern. The mark density may be varied in eighths, from one to seven eighths, (but excluding $\frac{3}{8}$ and $\frac{5}{8}$).

The *RST selection is $\frac{4}{8}$.

:UPATtern<n>

This is a contraction of the phrase: *User PATtern* and is used to define the contents of a pattern store. The value <n> must be in the range 1 through 12.
The commands under this node affect the storage of information as defined in the table below:

<n> = 0	Current pattern
<n> = 1 thru 4	Non-volatile RAM storage

[:LENGth] <numeric value>. This command sets the length of the pattern that is to be generated. When an alternate user-defined pattern is selected, the :LENGth refers to each half of the pattern. The pattern length has the following constraints:

1. 1 bit to 32 kbits in 1 bit steps,
2. 32 kbits to 64 kbits in 2-bit steps,
3. 64 kbits to 128 kbits in 4-bit steps,
4. 128 kbits to 256 kbits in 8-bit steps,
5. 256 kbits to 512 kbits in 16-bit steps,
6. 512 kbits to 1 Mbits in 32-bit steps,
7. 1 Mbit to 2 Mbits in 64-bit steps,
8. 2 Mbits to 4 Mbits in 128-bit steps,

The *RST command leaves this selection unchanged.

:LABel <string>. Defines a character string of up to 14 characters that is associated with the pattern. This is to make it easy for the user to comprehend the purpose of the particular pattern without having to refer to a lookup table.

The character data values of MINimum, MAXimum and DEFault are not defined for the label.

The *RST command leaves this selection unchanged.

:USE STRaight|APATtern. Defines the use of a user-defined pattern. When STRaight is selected the whole of the pattern is repeatedly output. When APATtern is selected the pattern is considered to be composed of two halves. The ":APCHange" command controls how these two halves are output.

The "USE" command also resets the selected pattern store. If "USE STRaight" is used, the store is set to have a length of 1 bit; this data bit is set to zero and the trigger bit is set to zero. If "USE APATtern" is used, the store is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A-B changeover.

For user-patterns used in the *STRaight* mode, the recommended sequence of issuing commands is:

PATTern:UPATtern<n>:USE	STRaight
PATTern:UPATtern<n>[:LENGth]	<numeric value>
PATTern:UPATtern<n>:DATA	<block data>
SOURce3:TRIGger:UPATtern<n>	<numeric value>

For user-patterns used in the *APATtern* mode, the recommended sequence of issuing commands is:

PATTern:UPATtern<n>:USE	APATtern
PATTern:UPATtern<n>[:LENGth]	<numeric value>
PATTern:UPATtern<n>:DATA	A,<block data>
PATTern:UPATtern<n>:DATA	B,<block data>

SOURce3:TRIGger:UPATtern<n> ABCHange|SOPattern

The *RST command leaves this selection unchanged.

:DATA [A|B,] <block_data>. Sets the bits of the pattern. The bits are sent as an arbitrary block diagram data element. The data may be sent 1 bit/byte or 8 bits/byte, under the control of the :FORMAT[:DATA] command. If 1 bit/byte is selected numeric values of either binary 1 or binary 0 only are allowed. If 8 bits/byte is selected the left-most bit of the first byte received forms the first bit of the pattern.

If ":USE APATtern" is selected, then the first parameter indicates which half pattern is to receive the data. If "USE STRaight " is selected, either "A" or no first parameter are acceptable.

The length of the <block data> embedded in the header refers always to the length in bytes irrespective of the current setting of the [:DATA] PACKed, <numeric value> command.

The character data values of MINimum, MAXimum and DEFault are not defined for the data.

To be consistent with the behavior of the pattern editor, more bits may be sent than are specified by the "LENGth" command, in which case the extra bits will be ignored and will not appear as part of the pattern. If the pattern length is subsequently extended the extra bits are filled with zeros. If fewer bits than specified by the "LENGth" command are sent, then the bits in the store beyond the length sent remain unchanged.

The pattern stores 1 through 4 have an overall length of 8192 bits, and pattern store 0 and 5 through 12 have an overall length of 4,194,304 bits

The following rules apply:

1. If 'PATTern:FORMat PACKed,1' is selected and data is sent with the ':UPATtern:DATA' command, then:
   block length = pattern length
2. If 'PATTern:FORMat PACKed,1' is selected and data is sent with the ':UPATtern:IDATa' command, then:
   block length = number of relevant bits in block
   (start bit + block size) <= pattern length
   block size >= 1
3. If 'PATTern:FORMat PACKed, 8' is selected and data is sent with the ':UPATtern:DATA' command, then:
   block size =((pattern length - 1) DIV 8) + 1
4. If 'PATTern:FORMat PACKed,8' is selected and data is sent with the ':UPATtern:IDATa' command, then:
   block size = ((number of relevant bits in block - 1) DIV 8) + 1
   (start bit + block size) = ((pattern length - 1) DIV 8 + 1) * 8
   block size >= 1

An arbitrary block program data element is a method of sending large quantities of data

from a controller to an instrument. It comes in two forms; an *indefinite length* format when the length of the transmission is not known, and a *definite length* format when the length is known. In the application here, the *definite length* format is used.

A definite length arbitrary block program data element is composed of two parts; a header and the data itself. The header is made up from three parts:

1. The first part is the ASCII character #.
2. The second part is a single non-zero ASCII digit. The magnitude of this digit equals the number of digits in the third part of the header.
3. The third part is composed of between 1 and 9 ASCII digits. The value of these digits taken together as a decimal integer equal the number of 8-bit data bytes which follow.

The data part is composed of a number of 8-bit data bytes.

As an example, if a user-pattern of length 7986 bits is to be set up, then the header would be #47986.

The *RST command leaves this selection unchanged.

:IDATa [A|B,] <start_bit>, <length_in_bits>, <block_data>

This command is similar to the :DATA command. The header is short for Incremental Data and the command is used to download just part of a user-defined pattern.

If ":USE APATtern" is selected, then the first parameter indicates which half pattern is to receive the data. If "USE STRaight " is selected, either "A" or no first parameter are acceptable.

The length of the <block data> embedded in the header refers always to the length of the data in bytes.

The first parameter defines the starting position within the overall pattern of the first bit of the transmitted pattern. The first bit is counted as bit zero. The second parameter defines how many bits are to be transmitted and the third parameter provides the data itself.

The query form of the command is of the format ":IDATa? <start bit> ,<length in bits>". The second parameter defines the length (in bits) of the data block to be output.

:FORMat:

This command controls the format of data transfer for the :PATTern:UPATtern<n>:DATA and :PATTern:UPATtern<n>:IDATa commands.

[:DATA] PACKed,<numeric value>. This command permits the packing of bits within a byte to be set. The First parameter must be *PACKed*. The <numeric value> parameter may be set to either 1 or 8.

Note that if a user pattern is selected and the [:SELect]? command is issued, then the response if "UPAT"; no '<n>' is appended.

The *RST selection is "PACKed,1".

:VOLTage

This node sets the values of the data input electrical levels.

:ZOTHreshold <numeric value>

This node allows the level at which the error detector discriminates between a zero and a one to be configured.

A numeric value parameter sets the level to a given value. It also sets :ZOTHreshold:AUTO OFF.

When in :ZOTHreshold:AUTO OFF, the query form of the :ZOTHreshold command returns the last user-entered value. When in :ZOTHreshold:AUTO ON, the query form returns the value automatically determined by the hardware.

If input termination and zero-to-one level are to be set up, then the input termination should be set up first.

The *RST selection is −1.3 V.

:AUTO <boolean>. This command enables an automatic mode in which the zero-to-one threshold level is set to the mean of the input signal.

The query form of this command returns the current setting of the hardware discrimination circuit.

The *RST selection is ON.

:GATE

This node controls the gating of the measurement.

[:STATe] <boolean>

Turns gating on or off.

The GATE[:STATe] ON command when in GATE:MODE SINGle is an overlapped command.

The *RST selection is OFF.

:MODE MANual|SINGle|Repetitive

Sets the gating period mode to either Manual, Single, or Repetitive.

This command causes all past results to be labelled as invalid.

Note This is not changeable while the error detector is gating.

The *RST selection is MANual.

:MANNer TIME|ERRors|BITS

This node control the manner by which the gating period is controlled. When TIME is selected the error detector performs SINGLE and REPETITIVE gating periods that are controlled by elapsed time. When the selected time has accumulated, the gating period ends.

When ERRors is selected the error detector performs SINGLE and REPETITIVE gating periods that are controlled by the accumulation of bit errors. When the selected number of bit errors have been accumulated, the gating period ends.

When BITS is selected the error detector performs SINGLE and REPETITIVE gating periods that are controlled by the accumulation of clock bits. When the selected number of clock periods have been accumulated, the gating period ends.

The *RST selection is TIME.

:PERiod

This node controls the period of SINGLE and REPETITIVE gating periods.

:PERiod[:TIME] <numeric value>

When GATE:MANNer is set to TIME, this sets the duration of the gating period in seconds. Neither a value less than 1 second not a value greater than 99 days, 23 hours, 59 minutes and 59 seconds is permitted.

This command causes all past results to be labelled as invalid.

Note This is not changeable while the error detector is gating.

The *RST selection is 1 minute.

:PERiod:ERRors <numeric value>

When GATE:MANNer is set to TIME, this sets the duration of the gating period in bit errors. Values of 10, 100 and 1000 are permitted.

This command causes all past results to be labelled as invalid.

Note This is not changeable while the error detector is gating.

The *RST selection is 100.

:PERiod:BITS <numeric value>

When GATE:MANNer is set to TIME, this sets the duration of the gating period in clock bits (or periods). Values of 1e7 through 1e15 in decade steps are permitted.

This command causes all past results to be labelled as invalid.

Note This is not changeable while the error detector is gating.

The *RST selection is 1e10.

:SYNChronisat ONCE|<boolean>

This node is for configuring the settings that control synchronization of the reference pattern to the incoming pattern.

A <boolean> parameter this turns automatic resynchronization off or on. If ONCE is selected a resynchronization is begun.

The *RST selection is ON.

:THReshold <numeric value>

This sets the threshold level of error ratio at which synchronization is deemed to be lost. Valid values are in the range 0 to 1.

Note The valid values are 1e-01 thru 1e-08 in decade steps.

The *RST selection is 1e-1.

:LOGGing ONCE|<boolean>

This node controls the output of logging information to an external controller. The use of these commands is permitted only when the error detector is set up to not be an HP-IB Controller. These commands control when a line of text is generated and made available to be read by the controller. An SRQ is asserted when a line of text is available.

The LOGGing ONCE command is equivalent to the front-panel Log On Demand key.

The LOGGing <boolean> command enables and disables the logging capability.

The *RST selection is OFF.

:SQUelch <boolean>

This command controls the logging squelch command. When enabled, further output of logged text is inhibited if triggered for ten consecutive seconds.

The *RST selection is OFF.

:ALARms <boolean>

This command controls the output of alarm conditions.

The *RST selection is OFF.

:THReshold <numeric parm>

This command permits a threshold to be set against which logging conditions are compared to decide when some logged information is output.

The *RST selection is 1.00e-3.

:DURing[:EVENt] NEVer|ESECond|ERGThrshld

This command selects which of three conditions apply when deciding when to log output during a gating period. The choices are between 'never ', 'on the occurrence of an error second' and 'when the error ratio over a second is greater than the threshold '.

The *RST selection is ESECond.

:END[:EVENt]NEVer| ALWays|NZECount|TERGthrshld

This command selects which of four conditions apply when deciding when to log output at the end of a gating period. The choices are between 'never', 'always', 'only on non-zero error count' and 'total error ratio greater than the threshold '.

The *RST selection is ALWays.

:END:REPort FULL|UREP. This command selects what to output at the end of a gating period. The choices are between 'FULL' that is, Main Results plus Interval Results plus G.821 Analysis, and 'UREP', that is results currently part of the User's Page.

The *RST selection is FULL.

:EYE

This node controls the automatic data/clock delay and automatic zero-one-threshold setting.

:TCENter|:TCENtre ONCE|<boolean>

The command :TCENter|:TCENtre ONCE causes the initiation of a search for the value of data/clock delay that puts the active edge in the center of the data eye, defined as between two points with a BER greater than EYE:THReshold. The command :TCENter|:TCENtre ON has the same effect. If successful, the command leaves the data/clock delay at this value and the center of the eye can be found by querying the data delay value. If unsuccessful, the EYE:WIDth? will return NAN (Not-A-Number). The command :TCENter|:TCENtre OFF aborts a previously started search.

Note

The clock/data align feature (used to center the sampling point in the data input eye) uses information derived from the input clock frequency.

For the clock/data align feature to work properly the input frequency must be stable during the measurement. The frequencies at the start and end of the measurement are compared and if they differ by more than 10% the measurement fails.

When a source clocking the instrument changes frequency it will take time for the new frequency to settle and an additional 1 second for the frequency measurement of the error detector module to settle. If a clock/data align is performed before the frequency has settled it may fail.

To overcome this potential problem, ensure the frequency is stable before starting a clock/data align, either by waiting a set amount of time or by querying the measured frequency until it is stable.

The command `:TCENter|:TCENtre` is an overlapped command.

:ACENter|:ACENtre ONCE|<boolean>

The command `:ACENter|:ACENtre ONCE` causes the initiation of a search for the value of data amplitude that puts the zero-to-one threshold at the midpoint between the upper and lower bounds of amplitude at which the bit error ratio exceeds the value setup by the `:EYE:THReshold` command. The command `:ACENter|:ACENtre ON` has the same effect. If successful, the command leaves the zero-one-threshold at this value and the center of the eye can be found by querying the zero-one-threshold value. If unsuccessful, the `EYE:HEIGht?` will return NAN (Not-A-Number). The command `:ACENter|:ACENtre OFF` aborts a previously started search.

The command `:ACENter|:ACENtre` is an overlapped command.

:WIDTh? <NR3>

This command interrogates the eye width found by the most recent search for the value of data/clock delay that put the active edge in the center of the data eye.

If the result is not available or the search was unsuccessful, then the number 9.91×E+37 (Not-A-Number, NAN) will be returned.

:HEIGht? <NR3>

This command interrogates the eye height found by the most recent search for the value of data amplitude that puts the zero-to-one threshold level midway between the upper and lower bounds at which the error ratio exceeds the threshold value set up by the `:EYE:THReshold` command.

If the result is not available or the search was unsuccessful, then the number 9.91eE+37 (Not-A-Number, NAN) will be returned.

:THReshold <numeric value>

This command sets the threshold to be used in the determination of the edges of the eye.

The *RST selection is 1.00e-3.

SENSe2 - The Clock Sense

This node is to setup the clock input.

Figure 4-11. SENSe2 Syntax Diagram

:VOLTage

This node is for setting electrical values of the clock input.

:EDGE POSitive|NEGative

Sets the active edge of the clock, that is, the one which will cause the input data to be sampled.

The *RST selection is POSitive.

INPut1 - The Data Input

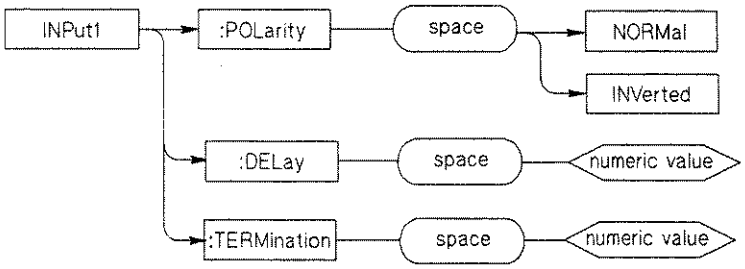


Figure 4-12. INPut1 Syntax Diagram

This node is for setting electrical characteristics of the data input.

:POLarity NORMal|INVerted

Sets the polarity of the detected data signal.

The *RST selection is NORMal.

:DELay <numeric value>

Sets the delay of the sampling of the data input relative to the active clock edge. The units are in seconds. The value is rounded to the nearest one picosecond.

The *RST selection is 0 ps.

:TERMination <numeric value>

This node permits the input termination level to be set to 0 volts (ground) or −2 volts.

If input termination and zero-to-one threshold level are to be set up, then the input termination should be set up first.

The *RST selection is 0 V.

INPut2 - The Clock Input

```
INPut2
   :TERMination      <numeric value>
   :TERMination?     <NR1>
```

This node is for setting electrical characteristics of the clock input.

:TERMination <numeric value>

This node permits the input termination level to be set to 0 volts (ground) or -2 volts.

If input termination and zero-to-one threshold level are to be set up, then the input termination should be set up first.

The *RST selection is 0 V.

Measurement Functions

The FETCh|PFETch command is used to return measurement values from the error detector. The FETCh command returns the results for the current gating period, and the PFETch (Previous FETch) returns the results for the previous gating period. The PFETch command is valid only in Repetitive Timed gating periods.

If any result is not available, then the number 9.91×E+37 (Not-A-Number, NAN) will be returned.

All results are updated every 0.2 seconds, except for the two DELTa? commands which are updated every 0.1 seconds.

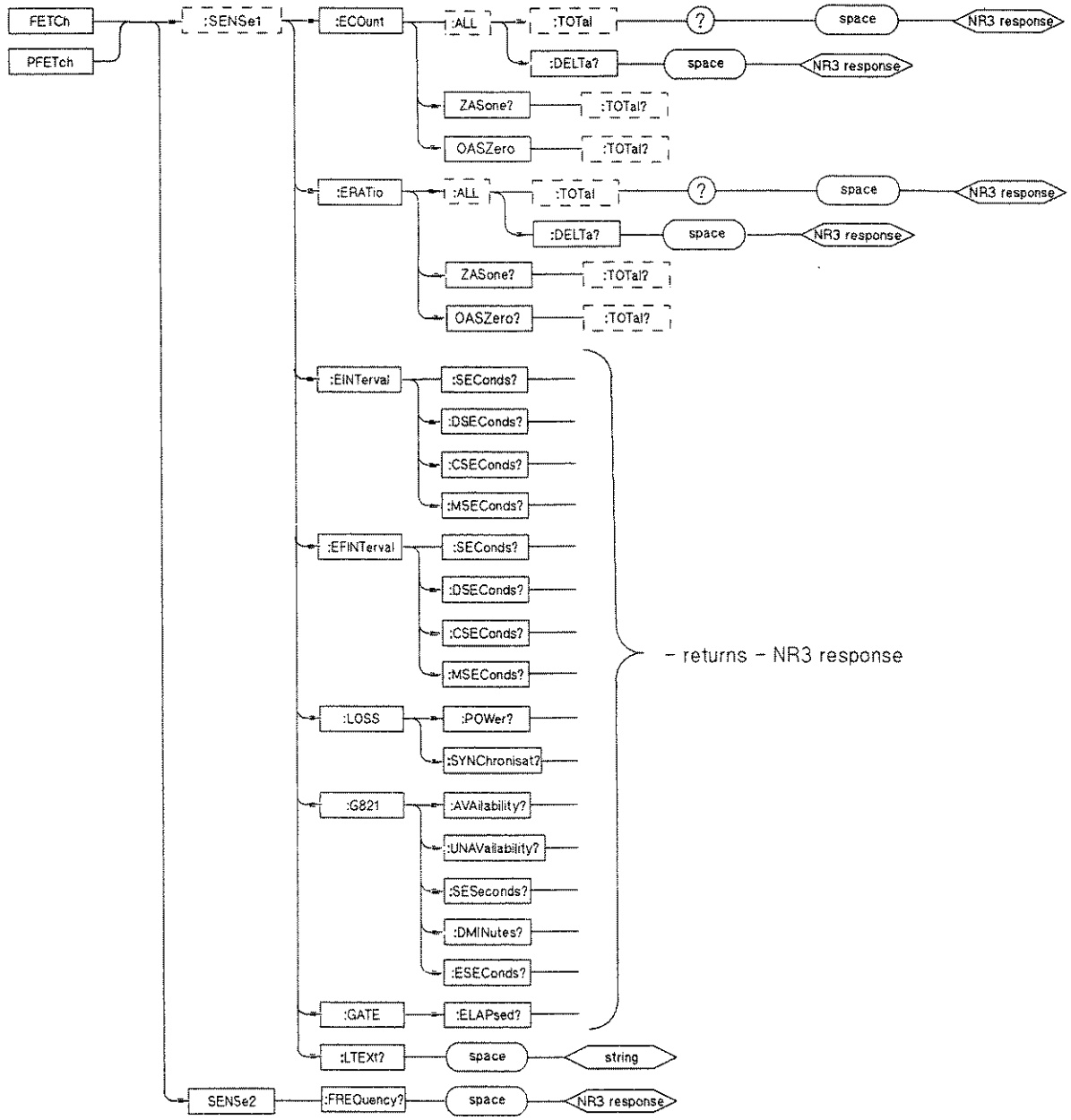


Figure 4-13. FETCh Syntax Diagram

[SENSe[1]]

This node (the default node) is for measurements on the data input.

:ECOunt

This is a contraction of the phrase `Error COUnt` and returns the number of errors counted in a time specified by the next level in the command. The next level is:

`[:ALL][:TOTal]? <NR3>`	*The total number of errors accumulated since the start of the gating period.*
`[:ALL]:DELTa? <NR3>`	*The number of errors accumulated in the last decisecond. This is intended to give a result that corresponds to the "instantaneous" error count. This value is available even when gating is turned off.*
`:ZASone[:TOTal]? <NR3>`	*This is a contraction of the phrase* Zero received AS One. *The command returns the number of errors accumulated since the start of the gating period, where each error is a true data zero received as a data one.*
`:OASZero[:TOTal]? <NR3>`	*This is a contraction of the phrase* One received AS Zero. *The command returns the number of errors accumulated since the start of the gating period, where each error is a true data one received as a data zero.*

:ERATio

This is a contraction of the phrase "Error RATio" and is the ratio of the number of errors to the number of bits received in a time interval, specified by the next level in the command. The next level is:

`[:TOTal]? <NR3>`	*The error ratio calculated from the total counts accumulated since the start of the gating period.*
`:DELTa? <NR3>`	*The "instantaneous" error ratio calculated from the counts obtained in the last decisecond. This value is available even when gating is turned off.*
`:ZASone[:TOTal]? <NR3>`	*This is a contraction of the phrase* Zero received AS One. *The command returns the error ratio calculated from a count of errors, where each error is a true data zero received as a data one.*
`:OASZero[:TOTal]? <NR3>`	*This is a contraction of the phrase* One received AS Zero. *The command returns error ratio calculated from a count of errors, where each error is a true data one received as a data zero.*

:EINTerval

This is a contraction of the phrase `Errored INTerval` and returns a count of the number of time intervals, the duration of which is selected by the next node, in which one or more errors were detected. The four time interval classifications are:

`:SEConds? <NR3>`	*One second*
`:DSEConds? <NR3>`	*One decisecond*
`:CSEConds? <NR3>`	*One centisecond*
`:MSEConds? <NR3>`	*One millisecond*

:EFINterval

This is a contraction of `Error Free INterval` and returns a count of the number of time intervals, the duration of which is selected by the next node, in which no error was detected. The four time interval classifications are:

`:SEConds? <NR3>`	*One second*
`:DSEConds? <NR3>`	*One decisecond*
`:CSEConds? <NR3>`	*One centisecond*
`:MSEConds? <NR3>`	*One millisecond*

:LOSS

This node returns a count of the number of seconds for which some characteristic was lost.

:POWer? <NR3>. This is the count of the number of seconds for which power was lost, since the start of the gating period.

If the Error Detector is not connected to an HP70004A Display, then this measurement is not available. If received, then it will return 9.91×E+37 (Not-A-Number, NAN).

:SYNChronisat? <NR3>. This is the count of the number of seconds for which the incoming pattern was not synchronized to the reference pattern, during the gating period.

:G821

This node returns a percentage of seconds that have been classified according to the CCITT's G.821 specification[8]. The subordinate nodes, representing the classifications, are:

`:AVAilability? <NR3>`	*% Availability*
`:UNAVailabili? <NR3>`	*% Unavailability*
`:SESeconds? <NR3>`	*% Severely Errored Seconds*
`:DMINutes? <NR3>`	*% Degraded MINutes*
`:ESEConds? <NR3>`	*% Errored SEConds*

:GATE

This node is used to return information about the gating period.

:ELAPsed? <NR3>. This node returns information about the degree to which the gating period has progressed. If GATE:MANNer TIME is selected, then this command returns the elapsed time into the gating period in units of seconds. If GATE:MANNer ERRors is selected, then this command returns the elapsed errors into the gating period. IF GATE:MANNer BITS is selected, then this command returns the elapsed clock bits into the gating period.

:LTEXt?

This command returns one line of log output. If a line of text is not currently available, then the message *No text currently available* is returned.

SENSe2

This node is for measurements on the clock input.

:FREQuency? <NR3>

This returns the current frequency of the signal on the clock input. This measurement is independent of the gating period.

DISPlay

This subsystem defines the usage of the display.

The available commands are:

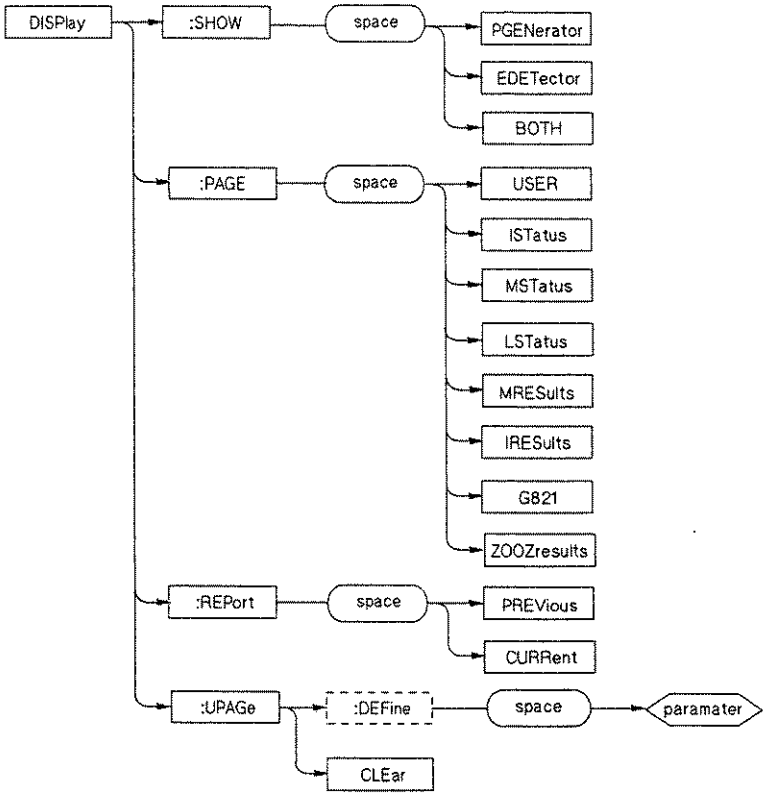


Figure 4-14. DISPlay Syntax Diagram

:SHOW PGENerator|EDETector|BOTH

This command controls the allocation of the display window. If the error detector has been allocated a window that is less than a full screen in size, then that window is allocated to either the pattern generator or error detector as appropriate. If the `DISPlay:SHOW BOTH` command is received then an error is generated. If the error detector has been allocated a window that is the full size of the screen, then that window is allocated to either the pattern generator or error detector as appropriate, and the `DISPlay:SHOW BOTH` command causes the window to be shared, with the error detector getting the upper half.

If the error detector does not have a slave pattern generator, then the receipt of the command `:DISPlay:SHOW` causes an error.

The `*RST` selection is:

full window - BOTH
half window - EDETector

:PAGE USER|ISTatus|MSTatus|LSTatus|MRESults|IRESults|G821|ZOOZ results

This command selects the display page to be viewed. The choices are:

USER	*USER-configurable page.*
ISTatus	*Input STatus page.*
MSTatus	*Main STatus page.*
LSTatus	*Logging STatus page.*
MRESults	*Main RESults page.*
IRESults	*Interval RESults page.*
G821	*CCITT's G.821 analysis page.*
ZOOZ results	*Zeros as one, one as zeros results*

The *RST selection is MRES.

:REPort PREVious|CURRent

This subsystem configures the result displays to show answers relating to either the previous gating period or current gating period. It has no effect on the results returned following the FETCh or PFETch commands.

The *RST selection is PREV.

:UPAGe[:DEFine] <parameter>

This subsystem configures the user-defined page to hold particular results or status information. As each command is received, the chosen parameter is added to the next vacant location in the User's Page.

Note that a separate User's Page exists for full and half sized display windows.

The single parameter is chosen from the following:

BECount	*big error count*
BEDCount	*big error delta count*
BERatio	*big error ratio*
BEDRatio	*big delta error ratio*
BGELapsed	*big gating elapsed*
OEcount	*one-as-zero count*
ZECount	*zero-as-one count*
OERatio	*zero-as-one ratio*
DTIMe	*date - time*
ECOunt	*error count*
DCOunt	*delta count*
ERATio	*error ratio*
DRATio	*delta error ratio*
PIDentity	*pattern identity*
CFRequency	*clock frequency*
ERSeconds	*error seconds*
EDSeconds	*error deciseconds*
ECSeconds	*error centiseconds*
EMSeconds	*error milliseconds*
PLSeconds	*power-loss seconds*
SLSeconds	*sync-loss seconds*

EFSeconds	*error-free seconds*
EFDSeconds	*error-free deciseconds*
EFCSeconds	*error-free centiseconds*
EFMSeconds	*error-free milliseconds*
BLANk	*blank line*
AVAilability	*G.821 availability*
UNAVailabili	*G.821 unavailability*
SESeconds	*G.821 severely errored seconds*
ERDSeconds	*G.821 errored seconds*
DMINutes	*G.821 degraded minutes*
SMODe	*synchronization mode*
ZOTMode	*zero-to-one mode*
DTERmination	*data termination (for backwards compatibility)*
TERMination	*clock and data termination DPOLarity << data polarity*
DIDelay	*data input delay*
CEDGe	*clock edge*
GMODe	*gating mode*
GREPort	*gating report*
GPERiod	*gating period*
GELapsed	*gating elapsed*
PLSeconds	*power-loss seconds*
SLSeconds	*sync-loss seconds*
LGSTatus	*logging status*
ALOGging	*alarms logging*
LDTRigger	*log during-trigger*
LETRigger	*log end-trigger*
LEReport	*log end-report*
LTHReshold	*logging threshold*
SSTatus	*squelch status*
HCONtroller	*HP-IB controller*
EETHreshold	*eye edge threshold*
EWIDth	*eye width*
EHEight	*eye height*

The *RST selection is given by the table below:

The default values of the User's Display Page when allocated half a screen are:

	INSTR PRESET	PRESET 1 & PRESET 2
Line 1:	pattern, line #1	B I G error delta count
Line 2:	pattern, line #2	B I G error delta count
Line 3:	error count	B I G error delta count
Line 4:	error ratio	B I G error delta count
Line 5:	gating mode	error count
Line 6:	gating period	0/1 threshold mode
Line 7:	0/1 threshold mode	data input delay
Line 8:	data input delay	gating elapsed

The default values of the User's Display Page when allocated a whole screen are:

	INSTR PRESET	PRESET 1 & PRESET 2
Line 1:	pattern, line #1	B I G error delta count
Line 2:	pattern, line #2	B I G error delta count
Line 3:	error count	B I G error delta count
Line 4:	delta error count	B I G error delta count
Line 5:	error ratio	B I G error count
Line 6:	error seconds	B I G error count
Line 7:	error free seconds	B I G error count
Line 8:	clock frequency	B I G error count
Line 9:		
Line 10:	sync mode	
Line 11:	sync loss seconds	0/1 threshold mode
Line 12:	gating mode	data input delay
Line 13:	gating period	eye width
Line 14:	gating elapsed	clock frequency
Line 15:		
Line 16:	0/1 threshold mode	gating elapsed
Line 17:	clock & data termination	
Line 18:	data polarity	error seconds
Line 19:	data input delay	error milliseconds
Line 20:	clock edge	sync loss seconds

The query form of the command returns a <boolean> to indicate whether a particular item is currently contained within the User's Page.

:UPAGe:CLEar

Clears the contents of the user-defined page.

SYSTem

This subsystem is mostly defined by SCPI for functions that are not related to instrument performance.

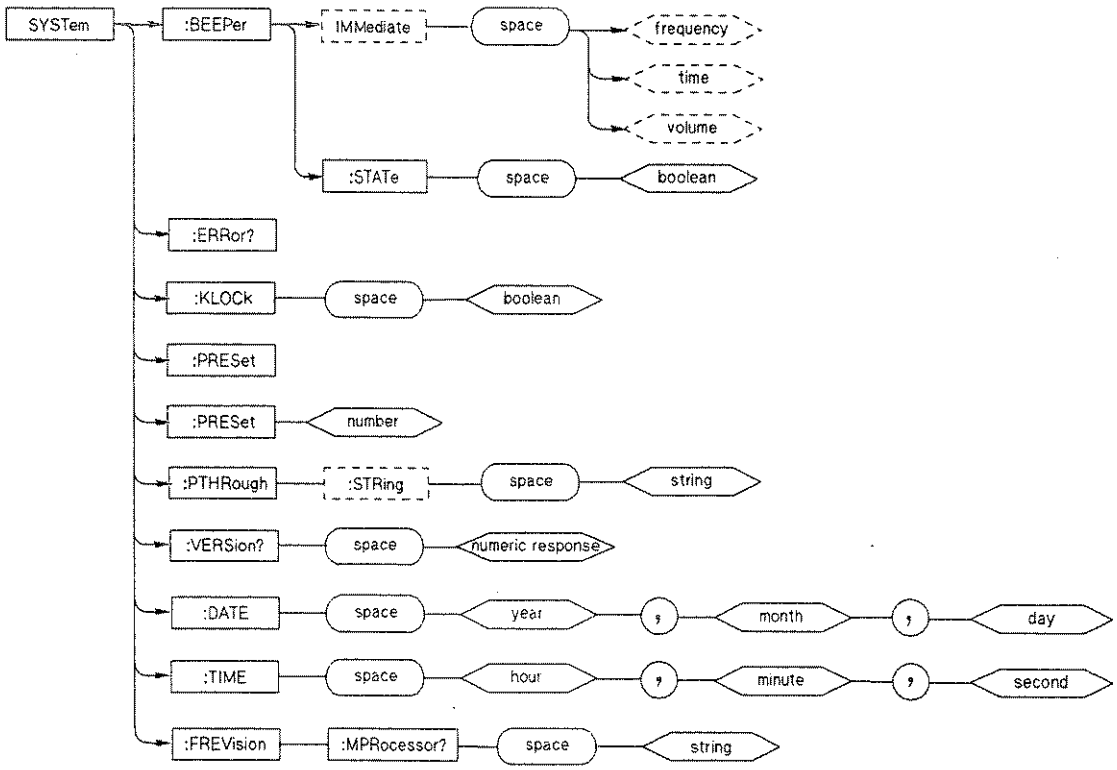


Figure 4-15. SYSTem Syntax Diagram

:BEEPer

This subsystem controls the audible beeper.

The `:BEEPer` command produces a beep only if a Display unit is connected and the error detector has been assigned a window.

[:IMMediate] [<freq>[,<time>[,<vol>]]]

Causes an audible tone to be generated. The optional parameters <freq>, <time> and <vol> are intended (in SCPI) to set the frequency, duration and volume of the beep; however, they are not implemented. A fixed beep is always generated.

The error detector accepts these parameters, but ignores them.

There is no query form of this command.

:STATe <boolean>

Controls whether the error detector beeps when an error is detected. (In this context "error" means a erroneous data bit on the data input not an internal instrument error nor an HP-IB message error.)

The `*RST` selection is OFF.

:ERRor?

This query-only command will pull the next error from the error queue, and return the error number and a string describing the error. The error queue is of depth ten.

Note SCPI-defined errors are all negative. The positive error numbers are available to be defined. The SCPI Messages section at the rear of this manual contains a list of error numbers.

:KLOCk <boolean>

This locks the instrument's keyboard. When locked, the user may not modify any of the instrument's configuration; although those keys that merely affect the display are still usable.

The `*RST` selection is OFF.

:PRESet|:PRESet<n>

Sets the error detector to a pre-defined "local operation" state. The choice of <n> is 0 through 2. `PRESet` and `PRESet0` both have the same effect as the front-panel (INSTR PRESET) key. `PRESet1` and `PRESet2` have the same effect as the front-panel `recall setup PRESET 1` and `PRESET 2` keys respectively.

This command causes all past results to be labelled as invalid.

Note If `PRESet2` is selected whilst an external controller is connected, then an error message is given because the instrument is attempting to take over HP-IB whilst it is already under control of the controller.

:PTHRough

The Pass-Through command allows a remote programming command to be passed through an MMS master module to a slave module. All valid remote commands for the slave can be passed through in this way. The command intended for the slave is given as a string parameter to the "PTHRough" command. The master pays no attention to the contents of this parameter.

Note The Pass-Through command would not be used to access the floppy disc of a slaved pattern generator whilst the error detector is gating.

[:STRing] <string>

The <string> parameter is passed through in its entirety to an MMS slave. The format of the parameter must be identical to the HP-IB command that would have been given to the slave had it been setup as an independent master. If the slave module detects an error in the <string> parameter passed through to it, it will generate an error in its error queue and set bit 1 of the master's Status Byte.

If no slave is present when the pass-through command is received, the error *slave needs service* is generated.

Example:

To set the frequency of an HP 70312A clock source (slaved to a master pattern generator, which is itself slaved to a master error detector) to 2.5 GHz, the following command could be issued:

```
OUTPUT clock ; "SYST:PTHR ""SYST:PTHR """"FREQ 2.5 GHZ""""""""
```

Note

The string parameter itself needs to be enclosed within quotation marks. Since the whole output string is already enclosed within double quotation marks, then the inner-most set must be represented by a quadruple set. Alternatively, a single set of single quotation marks may be used:

```
OUTPUT clock ; "SYST:PTHR ""SYST:PTHR 'FREQ 2.5 GHZ'"""
```

[:STRing]? <string>?

This command permits a query command to be passed through to the slave. Note that a question mark is needed to terminate the header; and another to terminate the <string> since the string is itself a header when received by the slave.

If no slave is present when the pass-through query command is received, an error is generated and a null string is returned.

Example:

To query the current frequency setting of a slaved HP 70312A clock source, the following commands could be used:

```
OUTPUT pgen ; "SYST:PTHR? 'FREQ?'"
ENTER pgen ; freq
```

Note

If the string parameter sent to the slave contains an error one of the following can occur:

- The slave will generate an error in its error queue and set bit 1 of the master's status byte.
- **For Query commands only:** the instrument can hang-up; no error message is reported. The instrument must be powered down, then powered up, and the string parameter checked/corrected.

:DATE <year>,<month>,<day>

This command permits the date of the real-time clock within the error detector to be set up. The command is invalid if another MMS module, with an HP-MSIB address to the left of and on the same row or lower than the error detector, exists and supports a valid date.

The range of valid <year> is 1990 through 2049.

:TIME <hour>,<minute>,<second>

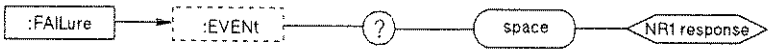
This command permits the time of the real-time clock within the error detector to be set up. The command is invalid if another MMS module, with an HP-MSIB address to the left of and on the same row or lower than the error detector, exists and supports a valid time.

:FREVision:MPRocessor?<string>

This command permits the revision code of the measurement processor to be interrogated.

STATus

The status conditions that the error detector needs to report are partly covered by the pre-defined status registers of *IEEE 488.2* and *SCPI*.



Two additional status registers covers instrument hardware failure conditions and MSIB slaves.

```
:FAILure
    [:EVENt]?   <NR1>
:SSERvice
    [:EVENt]?   <NR1>
     :ENABle    <NRf>
     :ENABle?   <NR1>
```

Status Byte

If an MMS slave module detects an error that would have caused an SRQ to be generated when operating as an independent master, then bit 1 of the Status Byte of the master MMS module is set. This bit is called *slave service*. In this way, the master module can relay the error situation to the external controller. The controller can then serial poll the master, detect that it is the slave module that has an error, and interrogate the slave's Status Byte and error queue.

:OPERation

The bits in this register permit the operational status of the error detector to be interrogated. All bits are CONDITION bits, except for BIT ERROR and END PERIOD which are EVENT bits.

The usage by the error detector is as follows:

SCPI MEANING	error detector usage
0 - CALibrating	-
1 - SETTling	-
2 - RANGing	-
3 - SWEeping	-
4 - MEASuring	The instrument is actively gating.
5 - Waiting for TRIG	-
6 - Waiting for ARM	-
7 - CORRecting	-
8 - Bit ERRor	A data bit-error has occurred.
9 - End PERiod	A timed, repetitive gating-period has ended.
10 - Logged TEXt	A line of logged text is available to be read. A negative transition occurs after every line of text.
11 - Clock-Data ALign	The clock-to-data align search is in process.
12 - 0/1 THR center	The zero-to-one threshold center search is in process.
13 - Reserved for SCPI definition	-
14 - PROGram running	-
15 - Zero	-

:QUEStionable

The bits in this register indicate that a signal is of questionable quality. The usage by the Error Detector is as follows:

SCPI MEANING	Error Detector usage
0 - VOLTage	Data loss.
1 - CURRent	-
2 - TIME	-
3 - POWer	-
4 - TEMPerature	-
5 - FREQuency	-
6 - PHASe	-
7 - MODulation	-

8 - CALibration	-
9 - Instr dependent	Clock loss.
10 - Instr dependent	Sync loss.
11 - Instr dependent	The signal is Unavailable according to the CCITT G.821 definition.
12 - Instr dependent	-Re-synchronization is in progress on the first pass of all possible pattern alignments.
13 - Instr summary	-
14 - Unexpected param	-
15 - Zero	-

:PRESet

The `PRESet` command is an event that configures the SCPI and device dependent status data structures, such that the device dependent events are reported at a higher level through the mandatory part of the status reporting structures.

The `PRESet` command affects only the enable register and the transition filter registers for the SCPI mandated and device dependent status data structures. `PRESet` does not affect either the "status byte" or the "standard event status" as defined by IEEE 488.2. `PRESet` does not clear any of the event registers. The `*CLS` command is used to clear all event registers in the device status reporting mechanism.

For the device dependent status data structures, the `PRESet` command sets the enable register to all **one's** and the transition filter to recognize both positive and negative transitions. For the SCPI mandatory status data structures, the `PRESet` command sets the transition filter registers to recognize only positive transitions and sets the enable register to zero.

:FAILure

The bits in this register indicate that a major hardware element of the instrument has failed. No capability is provided to query the condition register, setup the enable register, nor setup the positive or negative transition filters. This is because failures within this category are non-recoverable, and as such the enable registers are pre-defined.

The usage by the error detector is as follows:

BIT	error detector usage
0	ROM failure
1	RAM failure
2	Non-volatile memory corrupt
3	Gate Array failure
4	PIT failure
5	Interface board #1 failure
6	Measurement processor board failure
7	MSIB failure

8	EEPROM failure
9	Undefined
10	Undefined
11	Undefined
12	Undefined
13	Undefined
14	Undefined
15	Undefined

:SSERvice

The bits in this register indicate that am MSIB slave module is requesting service. No capability is provided to query the condition register nor setup the positive or negative transition filters. This is because failures within this category are events.

The usage by the Pattern Generator is as follows:

BIT	pattern generator usage
0	Slaved clock source or signal generator
1 thru 15	Undefined

IEEE Mandatory Commands

The following IEEE 488.2 mandatory commands are implemented:

*CLS	Clear Status Command.
*ESE	Standard Event Status Enable Command.
*ESE?	Standard Event Status Enable Query.
*ESR?	Standard Event Status Register Query.
*IDN?	Identification Query.
*OPC	Operation Complete Command.
*OPC?	Operation Complete Query.
*RST	Reset Command.
*SRE	Service Request Enable Command.
*SRE?	Service Request Enable Query.
*STB?	Read Status Byte Query.
*TST?	Self-Test Query.
*WAI	Wait-to-Continue Command.

IEEE Optional Commands

The following optional commands are implemented:

*OPT?	Option Identification Query.
*PSC	Power On Status Clear Command.
*PSC?	Power On Status Clear Query.
*RCL	Recall device setup.
*SAV	Save device setup.

IEEE Std 488.2-1987 System Related Specifications

Unless otherwise stated, the following specifications apply to both the Pattern Generator and Error Detector.

Device/Controller Synchronization Techniques

Overlapped and Sequential Commands

All commands are sequential commands, except for the following:

`GATe[:STATe] ON` for SINGLE TIMED repetitive periods in the Error Detector.

`EYE:TCENter|EYE:TCENtre ONCE|ON`

`EYE:ACENter|EYE:ACENtre ONCE|ON`

Operation Complete Messages

The following functional criteria are met when an operation complete message is generated:

A SINGLE TIMED gating period has expired.

The automatic eye time-centering operation has expired.

The automatic eye amplitude-centering operation has expired.

References

1.

2.

3. *HP 70000 MMS Communication Protocol Design Guide* (1988-11) - explains the addressing of MMS modules and the communications links between them.

4. ANSI/IEEE Std 488.1-1987 - *IEEE Standard Digital Interface for Programmable Instrumentation* - defines the electrical behavior of the HP-IB interface.

5. ANSI/IEEE Std 488.2-1987 - *IEEE Standard Codes, Formats, Protocols, and Common Commands for use with ANSI/IEEE Std 488.1-1987* - defines the allowable syntax of the messages that may be sent over the HP-IB interface.

6. *Standard Commands for Programmable Instruments, SCPI, Syntax and Style - Revision 1.0* - describes the underlying concepts and style guidelines of SCPI.

7. *Standard Commands for Programmable Instruments, SCPI Manual - Version 1990.0* - defines the grammar and vocabulary of SCPI commands.

8. CCITT G.821 (Red Book) - *Error Performance of an International Digital Connection forming part of an Integrated Services Digital Network*

9. CCITT O.151 - *Specification for Instrumentation to Measure Error Performance on Digital Systems*

5

Program Examples

This chapter provides example programs which perform the following functions:

- Log a summary of results to a disc file on completion of a measurement.
- Give a basic demonstration of the Error Performance Analyzer's measurement capability.
- Check command syntax.
- Identify the configuration of the system connected to the HP-IB.
- How to recover the user pattern data in Block format. Also to retransmit the block data and if necessary display or print the entire pattern.

```
10    ! Program name  LOGGER_1
20    !
30    ! This program logs the summary of results to a string on completion
40    ! of a measurement.
50    !
60    ! Run the program before setting the instrument to measure.  The program
70    ! will prompt the user when it is time to configure the instrument.
80    !
90   CALL Clr_crt                    ! Clear screen.
100  Error_det=717                   ! Assign variable name.
110  COM Error_det,Text$[80]
120  COM /Log_files/ Files$(200)[80]
130  A=SPOLL(Error_det)              ! Clear out any existing SRQ
140  OUTPUT Error_det;"*CLS"         ! Clear instrument.
150  ON INTR 7 CALL Read_data        ! Identify branch routine.
160  ENABLE INTR 7;2                 ! Enable computer.
170  CALL Config_reg                 ! Call register configure routine.
180  CALL Clear_buffer               ! Call instrument buffer clear routine.
190  CALL Config_instr               ! Call user configure routine.
200 Wait_loop:   !                   ! This loop simply waits for the SRQ.
210  DISP "Waiting for end of measurement"
220  GOTO Wait_loop
230  END
240  SUB Config_reg                  ! This routine sets the instrument to
250     CALL Clr_crt                 ! generate an SRQ at the end of gating.
260     COM /Log_files/ File$(*)
270     COM Error_det,Text$
280     OUTPUT Error_det;"STATUS:OPERATION:EVENT?"! Always clear Event register.
290     ENTER Error_det;Event_reg
300     OUTPUT Error_det;"STATUS:OPERATION:PTRANSITION 0"
310     OUTPUT Error_det;"STATUS:OPERATION:PTRANSITION 512" ! Pass pos bit 9.
320     OUTPUT Error_det;"STATUS:OPERATION:NTRANSITION 16" ! Pass neg bit 4.
330     OUTPUT Error_det;"STATUS:OPERATION:ENABLE 528" ! Enable bits 4 & 9.
340     OUTPUT Error_det;"*SRE 128"  ! Enable SRE on bit 7.
350  SUBEND
360  SUB Clear_buffer   ! This routine clears out any left over data from a
370                     ! previous measurement.
380     COM Error_det,Text$
390     DISP "Clearing Buffer"
400     WAIT 1
410     REPEAT
420        OUTPUT Error_det;"FETCH:LTEXT?"  ! Fetch a line of text.
430        ENTER Error_det;Text$
440  !     PRINT Text$
450     UNTIL Text$="No text currently available"  ! End of results message.
460     DISP
470  SUBEND
480  SUB Config_instr  ! This routine simply prompts the user to set the
490                    ! instrument manually.  The routine could be substituted
500                    ! by a much more sophisticated one.
510     COM /Log_files/ File$(*)
520     BEEP 600,.5
530     LOCAL 717
540     PRINT "Instrument returned to LOCAL mode"
550     PRINT
```

```
560     PRINT "Please set up measurement manually, set a 5 or 10 sec gating period,"
570     PRINT "set the logging on and press RUN GATING  to start the measurement."
580     PRINT "and then hit the RETURN/ENTER key to continue the program"
590 Get_key:   !
600     INPUT Key$
610     IF Key$<>"" THEN Get_key
620     CALL Clr_crt
630     ENABLE INTR 7;2
640  SUBEND
650  SUB Read_data  ! This routine gets the results from the Error Detector.
660     CALL Clr_crt
670     COM /Log_files/ File$(*)
680     COM Error_det,Text$
690     A=SPOLL(717)
700     DISP "Storing text"
710     REPEAT      ! This loop fetches all the available lines of text.
720        OUTPUT Error_det;"FETCH:LTEXT?"
730        ENTER Error_det;Text$
740        File$(Pointer)=Text$
750        Pointer=Pointer+1
760        WAIT .1
770        DISP "Storing text line",Pointer
780        IF Pointer>200 THEN    ! Maximum allowable length is 200 lines.
790           DISP "FILE FULL"
800           BEEP 400,1
810           SUBEXIT
820        END IF
830     UNTIL Text$="No text currently available"
840     FOR Lines=1 TO Pointer-2    ! Miss out last part of results message.
850        PRINT File$(Lines)
860     NEXT Lines
870     OUTPUT Error_det;"STATUS:OPERATION:EVENT?"
880     ENTER Error_det;Event_reg
890     CALL Config_instr
900  SUBEND
910  SUB Clr_crt
920     OUTPUT 2 USING "#,K";CHR$(255)&"K"
930  SUBEND
```

```
10  ! Program name  REMOTE_PNL
20  !
30  ! This program gives a basic demonatration of the Error Performance
40  ! Analyzer's measurement capability.  It allows control of the basic
50  ! generator/detector/clock parameters, the type of measurement gating
60  ! and it returns all the basic results.
70  !
80  ! It may be used as a stand alone program or the routines may be
90  ! expanded to produce a more sophisticated program.
100 !
110  CALL Clr_crt                      ! Call clear screen routine.
120  Pattern_gen=718                   ! Assign addresses to variables.
130  Error_det=717
140  COM Pattern_gen,Error_det         ! Assign variables to common memory area.

150  CALL Default                      ! Call instrument default settings.
160  CALL Clr_crt
170  REPEAT                            ! Set repeat loop around main menu.
180     CALL Main_menu
190  UNTIL Infinity
200  END
210  SUB Default
220 ! This subroutine sets the modules to their default settings.
230     CALL Clr_crt
240     DISP "Initialising System"
250     COM Pattern_gen,Error_det
260     CALL Clear_instr(Pattern_gen)
270     CALL Clear_instr(Error_det)
280     OUTPUT Pattern_gen;"SYSTEM:PTHROUGH '*CLS;*SRE 0;*ESE 0;*RST'"
290     WAIT 3
300  SUBEND
310  SUB Main_menu
320 ! The program revolves round this sub.  It displays the options available.

330     CALL Clr_crt
340     COM Pattern_gen,Error_det              ! Get variables from memory.
350     PRINT TABXY(35,1),"OPTIONS"            ! TABXY prints at x,y coordinate.

360     PRINT TABXY(1,3)
370     PRINT "1 - Summary of results"
380     PRINT "2 - Display/alter setup - Pattern Generator"
390     PRINT "3 - Display/alter setup - Signal Generator"
400     PRINT "4 - Display/alter setup - Error Detector"
410     PRINT "5 - Measurement setup"
420 Make_selection:        !
430     PRINT TABXY(1,30);"Selection"
440     INPUT Choice
450     IF Choice<1 OR Choice>5 THEN Make_selection     ! Check selection is OK.

460     SELECT Choice
470     CASE 1
480        CALL Result_sum                     ! Call results summary routine.

490     CASE 2
500        CALL Pgen_setup                     ! Alter Pattern Generator setup.

510     CASE 3
```

```
520     !  CALL Sgen_setup                    ! Alter Signal Generator setup.

530     PRINT "NOT AVAILABLE WITH HP 70311A AND HP 70312A CLOCK SOURCES "
540     PRINT "EDIT THIS PROGRAM TO USE SUB FOR HP 70322 AND HP 70320
SYNTHESIZERS"
550     PRINT "PRESS RETURN TO GO BACK TO MAIN MENU"
560     INPUT Anthing
570     CASE 4
580        CALL Edet_setup                    ! Alter Error Detector setup.
590     CASE 5
600        CALL Meas_setup                    ! Alter measurement setup.
610     END SELECT
620  SUBEND
630  SUB Result_sum
640 ! This subroutine display all the possible results.
650     CALL Clr_crt
660     COM Pattern_gen,Error_det
670     OUTPUT Error_det;"FETCH:ECOUNT?"
680     ENTER Error_det;Error_count
690     OUTPUT Error_det;"FETCH:ERATIO?"
700     ENTER Error_det;Error_ratio
710     OUTPUT Error_det;"FETCH:EINTERVAL:SECONDS?"
720     ENTER Error_det;Error_secs
730     OUTPUT Error_det;"FETCH:EFINTERVAL:SECONDS?"
740     ENTER Error_det;Error_free_secs
750     OUTPUT Error_det;"FETCH:G821:AVAILABILITY?"
760     ENTER Error_det;Availablity
770     OUTPUT Error_det;"FETCH:G821:UNAVAILABILI?"
780     ENTER Error_det;Unavailabilty
790     OUTPUT Error_det;"FETCH:G821:SESECONDS?"
800     ENTER Error_det;Sev_err_secs
810     PRINT TABXY(30,1),"SUMMARY OF RESULTS"
820     PRINT TABXY(1,5),"Error Count =";Error_count
830     PRINT "Error Ratio =";Error_ratio
840     PRINT "Errored Secs =";Error_secs
850     PRINT "Error Free Secs =";Error_free_secs
860     PRINT
870     PRINT "Availablity =";Availability
880     PRINT "Unavailability =";Unavailability
890     PRINT "Severely Errored Secs =";Sev_err_secs
900     PRINT TABXY(1,18);"Enter number to change parameter or"
910     PRINT "Hit RETURN key for main menu"
920     INPUT Continue
930  SUBEND
940  SUB Pgen_setup   ! This routine alters the Pattern Generator setup.
950     CALL Clr_crt
960     COM Pattern_gen,Error_det
970     PRINT "Pattern Generator"
980     PRINT "-----------------"
990     PRINT
1000    PRINT "The current configuration is as follows:-"
1010    PRINT
1020    OUTPUT Pattern_gen;"PATTERN?"
1030    ENTER Pattern_gen;Pgen_pattern$
1040    OUTPUT Pattern_gen;"PATTERN:EADDITION?"
```

```
1050    ENTER Pattern_gen;Error_add
1060    OUTPUT Pattern_gen;"VOLTAGE?"
1070    ENTER Pattern_gen;Data_amp
1080    PRINT "1 - Pattern is ";Pgen_pattern$
1090    PRINT "2 - Error Add is ";Error_add
1100    PRINT "3 - Data O/P Amplitude is ";Data_amp;"V"
1110    PRINT TABXY(1,18);"Enter number to change parameter or"
1120    PRINT "hit RETURN key for main menu."
1130    INPUT Key$
1140    SELECT Key$
1150    CASE "1"
1160       INPUT "PRBS Length",Prbs_length$
1170       Prbs_length$="PATTERN PRBS"&Prbs_length$
1180       OUTPUT Pattern_gen;Prbs_length$
1190 Error_add:          !
1200    CASE "2"
1210       INPUT "Error Add enabled ?  Y or N",Choice$
1220       IF Choice$="Y" THEN OUTPUT Pattern_gen;"PATTERN:EADDITION 1"
1230       IF Choice$="N" THEN OUTPUT Pattern_gen;"PATTERN:EADDITION 0"
1240       IF Choice$<>"Y" OR Choice$<>"N" THEN Error_add
1250 Amplitude_set:    !
1260    CASE "3"
1270       INPUT "Enter Data amplitude in volts",Data_amp
1280       OUTPUT Pattern_gen;"VOLTAGE";Data_amp
1290    END SELECT
1300 SUBEND
1310 SUB Sgen_setup    ! This routine alters the stup of the Signal Generator.

1320  !
1330  ! ******************************************************
1340  ! THIS ROUTINE IS FOR HP 70320A  AND HP 70322A ONLY    *
1350  ! ******************************************************
1360    CALL Clr_crt
1370    DIM
Command$[50],Reply$[50],Sig_gen_freq$[50],Sig_gen_amp$[50],Sig_gen_op$[50]
1380    COM Pattern_gen,Error_det
1390    PRINT "Signal Generator"
1400    PRINT "----------------"
1410    PRINT
1420    PRINT "The current configuration is as follows:-"
1430    PRINT
1440    CALL Sig_gen("':FREQUENCY?'",Reply$)
1450    Sig_gen_freq$=Reply$
1460    CALL Sig_gen("':AMPLITUDE?'",Reply$)
1470    Sig_gen_amp$=Reply$
1480    CALL Sig_gen("':AMPLITUDE:STATE?'",Reply$)
1490    Sig_gen_op$=Reply$
1500    PRINT "1 - Frequency is ";Sig_gen_freq$;"Hz"
1510    PRINT "2 - Amplitude is ";Sig_gen_amp$;"dBm"
1520    PRINT "3 - Output is ";Sig_gen_op$
1530    PRINT TABXY(1,18);"Enter number to change paramter or"
1540    PRINT "hit RETURN key for main menu."
1550    INPUT Key$
1560    SELECT Key$
1570    CASE "1"
```

```
1580        INPUT "Enter fequency value and units i.e. 1GHZ",Sig_gen_freq$
1590        Command$="'"&":FREQUENCY "&Sig_gen_freq$&"'"
1600        CALL Sig_gen(Command$)
1610     CASE "2"
1620        INPUT "Enter amplitude value and units i.e. +1DBM",Sig_gen_amp$
1630        Command$="'"&":AMPLITUDE "&Sig_gen_amp$&"'"
1640        CALL Sig_gen(Command$)
1650     CASE "3"
1660        Command$="':AMPLITUDE:STATE ON'"
1670        CALL Sig_gen(Command$)
1680        CALL Freq_stable
1690     END SELECT
1700 SUBEND
1710 SUB Edet_setup  ! This routine alters the setup of the Error Detector.
1720     CALL Clr_crt
1730     COM Pattern_gen,Error_det
1740     PRINT "Error Detector"
1750     PRINT "--------------"
1760     PRINT
1770     PRINT "The current configuration is as follows:-"
1780     PRINT
1790     OUTPUT Error_det;"PATTERN?"
1800     ENTER Error_det;Pgen_pattern$
1810     OUTPUT Error_det;"SYNCHRONISAT?"
1820     ENTER Error_det;Sync_state
1830     IF Sync_state=1 THEN
1840        Sync_state$="AUTO"
1850     ELSE
1860        Sync_state$="MANUAL"
1870     END IF
1880     OUTPUT Error_det;"VOLTAGE:ZOTHRESHOLD?"
1890     ENTER Error_det;Data_threshold
1900     PRINT "1 - Pattern is ";Pgen_pattern$
1910     PRINT "2 - Sync State is ";Sync_state$
1920     PRINT "3 - Data Threshold is ";Data_threshold;"V"
1930     PRINT TABXY(1,18);"Enter number to change parameter or"
1940     PRINT "hit RETURN key for main menu."
1950     INPUT Key$
1960     SELECT Key$
1970     CASE "1"
1980        INPUT "PRBS Length",Prbs_length$
1990        Prbs_length$="PATTERN PRBS"&Prbs_length$ ! Remove blank in string.

2000        OUTPUT Error_det;Prbs_length$
2010 Sync_mode:          !
2020     CASE "2"
2030        INPUT "Auto Synchronization mode - Y or N",Sync_mode$
2040        IF Sync_mode$="Y" THEN OUTPUT Error_det;"SYNCHRONISAT 1"
2050        IF Sync_mode$="N" THEN OUTPUT Error_det;"SYNCHRONISAT 0"
2060        IF Choice$<>"Y" OR Choice$<>"N" THEN Sync_mode
2070 Amplitude_set:   !
2080     CASE "3"
2090        INPUT "Enter Data amplitude in volts  (-5V to 1V)",Data_threshold
2100        OUTPUT Error_det;"VOLTAGE:ZOTHRESHOLD ";Data_threshold
2110     END SELECT
```

```
2120 SUBEND
2130 SUB Meas_setup   ! This routine alters the measurement setup.
2140    CALL Clr_crt
2150    COM Pattern_gen,Error_det
2160 Gate_loop:   !
2170    OUTPUT Error_det;"GATE:MODE?"
2180    ENTER Error_det;Meas_mode$
2190    Meas_mode$=Meas_mode$[1,4]
2200    OUTPUT Error_det;"GATE:PERIOD?"
2210    ENTER Error_det;Meas_per
2220    OUTPUT Error_det;"GATE:STAT?"
2230    ENTER Error_det;Meas_state$
2240    IF Meas_state$="1" THEN
2250       Meas_state$="ON"
2260    ELSE
2270       Meas_state$="OFF"
2280    END IF
2290    PRINT TABXY(30,1);"MEASURE SETUP"                          ! Display
alter options.
2300    PRINT TABXY(1,5);"1 - Measurement Mode  ";Meas_mode$
2310    PRINT "2 - Measurement Period ";Meas_per;"secs"
2320    PRINT "3 - Measurement Start/Stop ";Meas_state$
2330    PRINT TABXY(1,18);"Enter number to change parameter or"
2340    PRINT "Hit RETURN key for main menu"
2350    INPUT Choice$
2360    SELECT Choice$
2370    CASE "1"
2380       INPUT "Enter the preferred measurement mode  MAN/SING/REP",Meas_mode$

2390       OUTPUT Error_det;"GATE:MODE ";Meas_mode$
2400    CASE "2"
2410       INPUT "Enter the required measurement period in seconds",Meas_per
2420       OUTPUT Error_det;"GATE:PERIOD ";Meas_per
2430    CASE "3"
2440       IF Meas_state$="ON" THEN OUTPUT Error_det;"GATE:STATE 0"
2450       IF Meas_mode$="MAN" OR Meas_mode$="REP" THEN
2460          CALL Man_meas
2470          SUBEXIT
2480       ELSE
2490          CALL Auto_meas
2500          SUBEXIT
2510       END IF
2520    CASE ""
2530       SUBEXIT
2540    END SELECT
2550 SUBEND
2560 SUB Clr_crt  ! This routine clears the screen.
2570    OUTPUT 2 USING "#,K";CHR$(255)&"K"
2580 SUBEND
2590 SUB Clear_instr(Module)   ! This routine sets the system to the recommended

2600                           ! IEEE 488.2 default.
2610  !
2620  !
2630    CLEAR Module
```

```
2640    OUTPUT Module;"*CLS; *SRE 0;  *ESE 0"
2650    OUTPUT Module;"STATUS:PRESET"
2660 SUBEND
2670  !
2680 SUB Sig_gen(Command$,OPTIONAL Reply$) ! This is a utility routine for
2690                                       ! setting the Signal Generator.
2700    DIM Rev_command$[50]
2710    COM Pattern_gen,Error_det
2720    Rev_command$=REV$(Command$)
2730    IF Rev_command$[2,2]="?" THEN
2740       OUTPUT Pattern_gen;"SYSTEM:PTHROUGH? ";Command$
2750       ENTER Pattern_gen;Reply$
2760    ELSE
2770       OUTPUT Pattern_gen;"SYSTEM:PTHROUGH ";Command$
2780    END IF
2790 SUBEND
2800 SUB Freq_stable    ! This routine ensures that the Error Detector is
2810                    ! measuring a stable frequency before attempting
2820                    ! any measurements.
2830    COM Pattern_gen,Error_det
2840 Freq_stable:   !
2850    OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
2860    ENTER Error_det;Frequency1
2870    WAIT 1
2880    OUTPUT Error_det;"FETCH:SENSE2:FREQUENCY?"
2890    ENTER Error_det;Frequency2
2900    DISP "Waiting for frequency to settle";Frequency1,Frequency2
2910    IF Frequency1<>Frequency2 THEN Freq_stable
2920    DISP
2930 SUBEND
2940 SUB Man_meas   ! This routine is called when manual/repetitive gating
2950                ! has been chosen.  It prompts the user to hit the
2960                ! RETURN key to halt gating.
2970 ! The subroutine is called when manual/repetative gating has been chosen.

2980 ! It prompts the user to hit the 'RETURN' key to halt gating.
2990    CALL Clr_crt
3000    COM Pattern_gen,Error_det
3010    OUTPUT Error_det;"GATE:STAT ON"
3020    PRINT "Instrument ";CHR$(129);"measuring";CHR$(128)
3030    PRINT
3040    PRINT TABXY(1,18);"Hit RETURN key to halt measurement and return to main
menu"
3050    ON KBD GOTO Continue
3060 Time_loop:  !
3070    PRINT TABXY(1,5);"Elapsed time = ";Elapsed_time
3080    WAIT 1
3090    Elapsed_time=Elapsed_time+1
3100    GOTO Time_loop
3110 Continue:  !
3120    OUTPUT Error_det;"GATE OFF"
3130    SUBEXIT
3140 SUBEND
3150 SUB Auto_meas  ! This routine is called when single shot gating is chosen.
```

```
3160                ! It sets the Error Detector to issue a Service Request
3170                ! to indicate the end of a gating period.
3180    CALL Clr_crt
3190    COM Patt_gen,Error_det
3200    ON INTR 7 GOTO Abandon
3210    OUTPUT Error_det;":STAT:OPER:ENAB 16"
3220    OUTPUT Error_det;":STAT:OPER:NTR 16"
3230    OUTPUT Error_det;"*SRE 128"
3240    OUTPUT Error_det;"GATE:STATE ON"
3250    OUTPUT Error_det;":STAT:OPER:EVEN?"
3260    ENTER Error_det;Value
3270    Svalue=SPOLL(Error_det)
3280    ENABLE INTR 7;2
3290    PRINT "Waiting for completion of measurement"
3300 Time_loop:     ! Set a loop to indicate elapsed gating time.
3310    PRINT TABXY(1,5);"Elapsed time = ",Elap_time,"secs"
3320    WAIT 1
3330    Elap_time=Elap_time+1
3340    GOTO Time_loop
3350    A=SPOLL(Error_det)
3360 Abandon:       !
3370 SUBEND
```

```
10  ! PROGRAM NAME:  SYNTAX_CHK
20  !
30  ! PROGRAM DESCRIPTION:
40  ! This program can be used as a tutorial to check command syntax.
50  ! It will generate a Service Request when incorrect commands are
60  ! entered and flag a command error.  It could be used for checking
70  ! dubious command syntax when creating program.
80  !
90   CALL Clr_crt                  ! Clear screen.
100  DIM Answer$[50]               ! This string will be used for data returned
110  !                             ! from the instrument.
120  Pattern_gen=718               ! Assign HP-IB addresses to variables.
130  COM Pattern_gen               ! Store variable Pattern_gen in common memory.
140  A=SPOLL(Pattern_gen)          ! Clear any existing SRQ.
150  OUTPUT Pattern_gen;"*CLS"     ! Establish default state.
160  ON TIMEOUT 7,2 CALL Int_serv
170  ON INTR 7 CALL Int_serv       ! Set interupt branch routine.
180  ENABLE INTR 7;2               ! Enable computer to recognize SRQ.
190  CALL Set_event_srq            ! Call the routine to set the reason for SRQ.
200 Command_enter:  !
210  REPEAT                        ! This repeat loop is simply to allow the
220     CALL Clr_crt
230     PRINT TABXY(30,1);"Syntax Checker"
240     Command$=""                ! user to enter commands.  To trigger the
250     INPUT "Command ?",Command$! service request, enter an illegal command
260     OUTPUT Pattern_gen;Command$! string.
270     Reverse$=REV$(Command$)
280     IF Reverse$[1,1]="?" THEN ! This little extra routine is optional.  It
290        ENTER Pattern_gen;Answer$! checks to see if the command entered is an
300        PRINT TABXY(1,25);Answer$! interrogation command and if so asks for a
310        WAIT 2
320     END IF                     ! reply.
330     WAIT .1
340  UNTIL Command$=""             ! Hitting RETURN key with no characters
350  DISP "End of program"         ! entered will end the progam.
360  END
370  SUB Set_event_srq
380     COM Pattern_gen
390   ! This rountine allows the user to define the conditions in the Standard
400   ! Event Register that will generate a service request.
410     OUTPUT Pattern_gen;"*ESE 32"! Assign value of mask, enabling bit 5.
420     OUTPUT Pattern_gen;"*SRE 32"! Set Service Request Enable register to
430    !                            ! generate an SRQ when the Standard Event
440    !                            ! register summary bit (ESB) is set.
450  SUBEND
460  SUB Int_serv
470     COM Pattern_gen
480     PRINT TABXY(1,10)
490     BEEP 100,.75
500     Status_byte=SPOLL(Pattern_gen)      ! Clear SRQ bit in Status Byte register.
510     IF BIT(Status_byte,5) THEN
520        PRINT "Service Request received"
530        PRINT "*****Command Error*****"
540     END IF
550     INPUT "Hit RETURN key to resume",Key$
```

```
560     OUTPUT Pattern_gen;"*CLS" ! Clear instrument.
570     ENABLE INTR 7;2           ! Re-enable interrupt detect on controller.
580   SUBEND
590   SUB Clr_crt                 ! This routine clears the screen.
600     OUTPUT 2 USING "#,K";CHR$(255)&"K"
610   SUBEND
```

```
10    ! Program name:   SYSTEM_ID
20    !
30    ! This program identifies the configuration of the system that is
40    ! connected to the HP-IB.
50    !
60   CALL Clr_crt
70   DIM Identity$[50]
80   ON TIMEOUT 7,.2 GOTO Recovery
90   PRINT TABXY(20,1);"HP 71600 Series - System Configuration"
100  PRINT
110  PRINT
120  FOR Address=702 TO 730  ! Search through all possible addresses.
130     DISP Address
140     A=SPOLL(Address)
150     OUTPUT Address;"*IDN?"
160     ENTER Address;Identity$
170     PRINT Identity$;" at address";Address
180     IF Address=718 THEN CALL Check_for_slave(Address,Identity$)
190 Recovery:   !
200  NEXT Address
210  DISP "End of search"
220  END
230  SUB Check_for_slave(Address,Identity$) ! Check for the presence of a slave.
240     OUTPUT Address;"SYSTEM:PTHROUGH? '*IDN?'"
250     ENTER Address;Identity$
260     PRINT Identity$;" slave present at address";Address
270 Quit:      !
280  SUBEND
290  SUB Clr_crt   ! Clear screen subroutine.
300     OUTPUT 2;CHR$(255)&"K"
310  SUBEND
```

```
1     REM *************USER PATTERN STORAGE PROGRAM*************************
2     !
3     ! This program shows how to recover the user pattern data in Block
4     ! format. It also show how to retransmitt the block data and if necessary
5     ! display or print the entire pattern. Additional SUB routines to build
6     ! your own Pattern are included. Delete the !! at the CALL's to adapt
7     ! the program. You could write an other SUB to store the user Pattern
8     ! block data to disc & recover it.
9     !
10    ! This program is equally suitable for for use with the HP 70842A
11    ! Error Detector module simply comment out the appropriate line.
12    !
13    !-------------------------------------------------
14    !REV 2 17 DEC 91 15:00
15    ! CHANGE IF/THEN CONDITION ON LINE 638 TO 255 BITS
16    ! AND CLARIFY MESSAGE WITH LINE 641
17    !-------------------------------------------------
18    !
19    !-------------------
20    !REV 3 19 FEB 92
21    !Reduced line lengths
22    !Added extra comments
23    !tested on B modules
24    !--------------------
25    !
26    Module_addss=718               ! HP 70841A Pattern Generator addresss
27  ! Module_addss=717               ! HP 70842A Error Detector address
28    ALLOCATE Block$[9000]
29    Block$=" "
30    Print_addss=1
31    !
32    ! SUITABLE CALL COMBINATIONS ARE 4,5,6,7...OR 1,4,7,8.....OR
33    ! 2,4,7,8.....OR 3......OR SAME AS ABOVE BUT WITH 7 COMMENTED OUT.
34    !
35    CALL Build_test_patt(Block$,Patt_length)                    ! 1
36  ! CALL Build_rpt_patt(Block$,Patt_length)                     ! 2
37  ! CALL Testpatt                                               ! 3
38    CALL Clear_instr(Module_addss)                              ! 4
39  ! CALL Ck_patt_length(Module_addss,Patt_length)               ! 5
40  ! CALL Store_user_patt(Module_addss,Block$)                   ! 6
41    CALL Print_user_patt(Print_addss,Block$,Patt_length)        ! 7
42    CALL Set_user_patt(Module_addss,Block$,Patt_length)         ! 8
43    !
44    !
45    END
46    !
47    SUB Clear_instr(Module)
48    !
49    ! This is the recommended sequence to initialize devices which conform to
50    ! IEEE 488-2.
51    !
52    ! CLEAR Module
53      OUTPUT Module;"*CLS; *SRE 0;  *ESE 0"
54      OUTPUT Module;"STATUS:PRESET"
55    SUBEND
```

```
!
!
SUB Ck_patt_length(Module,Patt_length)
!*************************************************************************
! NOTE THE NUMBERS FOR UPATTERN MUST MATCH IN THE LINES CONTAINING
! LENGTH? AND DATA?
!*************************************************************************
  OUTPUT Module;"PATTERN:UPATTERN4:LENGTH?"    ! Select UPATT 1,2,3,or 4
  ENTER Module;Patt_length
SUBEND
!
!
!
SUB Store_user_patt(Module,Block$)
  OUTPUT Module;"PATTERN:UPATTERN4:DATA?"    ! select UPATT 1,2,3,or 4
  ENTER Module;Block$
SUBEND
!
SUB Print_user_patt(Print_addss,Block$,Patt_length)
  DIM Dispblock$[9000]                       ! Sets up another array.
  Length_of_block=LEN(Block$)                ! Find length of array
  Start_number=VAL(Block$[2,2])+3            ! Miss out header
!
  FOR N=Start_number TO Length_of_block
    Pointer=Pointer+1                        ! Initialise new pointer
    Bit_value=NUM(Block$[N,N])               ! Assign 1 or 0 to Bit_value
    Dispblock$[Pointer,Pointer]=VAL$(Bit_value)   ! Fill new $ Dispblock$
  NEXT N
!
!
! THIS PRINT ROUTINE GIVES A CLOSE APPROXIMATION TO THE DISPLAY PICTURE.
!
  PRINT                                       ! throw a line space
  PRINT "Pattern Length:           ";Patt_length
  PRINT
  PRINT "[   0]";":";"  ";
  FOR M=1 TO Pointer
    PRINT Dispblock$[M,M];
    IF M MOD 4=0 THEN PRINT "  ";
    IF M MOD 32=0 THEN
      PRINT
      PRINT "[";M;"]";":";"  ";
    END IF
  NEXT M
SUBEND
!
SUB Set_user_patt(Module,Block$,Patt_length)
!  OUTPUT Module USING "#,K";":PATTERN:UPATTERN3:DATA ";Block$
  OUTPUT Module;":PATTERN:UPATTERN4:LENGTH ";Patt_length
  OUTPUT Module;":PATTERN:UPATTERN4:DATA ";TRIM$(Block$)
! To edit Block$ :
! Block$[7,7]=CHR$(0) ......[7,7] is the first data point, the header is
! before this, SOMETINMES IT BE BE <7,7 !!!!!
  LOCAL 717
SUBEND
```

```
111   SUB Build_test_patt(Block$,Patt_length)
112   !
113   ! This SUB allows you to put your own unique pattern in a DATA array
114   !
115     ALLOCATE Build_patt$(9000)[10]
116   !
117   ! The DATA blocks below were constructed using COPYLINES !!!
118   !
119     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
120     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
121     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
122     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
123     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
124     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
125     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
126     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
127     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
128     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
129     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
130     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
131     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
132     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
133     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
134     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
135     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
136     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
137     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
138     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
139     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
140     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
141     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
142     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
143     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
144     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
145     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
146     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
147     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
148     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
149     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
150     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
151     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
152     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
153     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
154     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
155     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
156     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
157     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
158     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
159     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
160     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
161     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
162     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
163     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
164     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
165     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
```

```
166     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
167     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
168     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
169     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
170     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
171     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
172     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
173     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
174     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
175     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
176     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
177     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
178     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
179     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
180     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
181     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
182     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
183     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
184     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
185     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
186     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
187     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
188     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
189     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
190     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
191     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
192     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
193     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
194     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
195     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
196     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
197     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
198     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
199     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
200     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,1111,0000
201     DATA 1111, 0000,1111,0000,1111,0000,1111,1111,0000,1111,0000,1111,FIN
202     !
203     !
204     REPEAT
205       READ Build_patt$(N)
206       N=N+1
207     UNTIL Build_patt$(N-1)="FIN"
208     !
209     !
210     N=0
211     Strbit=1
212     !
213     !
214     LOOP
215     EXIT IF Build_patt$(N)="FIN"
216       FOR Bit=1 TO 4
217         Block=VAL(Build_patt$(N)[Bit,Bit])
218         Block$[Strbit,Strbit]=CHR$(Block)
219         Strbit=Strbit+1
220       NEXT Bit
```

```
221       N=N+1
222     END LOOP
223     !
224     !  BUILD BLOCK
225     Patt_length=LEN(Block$)
226     IF Patt_length>255 THEN
227       Remainder=Patt_length MOD 32
228       IF Remainder>0 THEN
229         PRINT "IF YOUR PATTERN LENGTH IS 256 BITS OR LONGER"
230         PRINT "YOU MUST RESTRUCTURE DATA IN MULTIPLES OF 32 BITS."
231         PRINT "RESET THE COMPUTER AND ALTER THE PROGRAM BITS"
232         PRINT "IN ONE OFTHE DATA LINES..........."
233         PRINT "YOU NEED ";Remainder/4;"FEWER WORDS OR ";
234         PRINT (32-Remainder)/4;" MORE WORDS"
235         PAUSE
236       END IF
237     END IF
238     Patt_length$=VAL$(Patt_length)
239     Number_digits=LEN(Patt_length$)
240     Number_digits$=VAL$(Number_digits)
241     Header$="#"&Number_digits$&Patt_length$
242     Block$=Header$&Block$
243    !
244   SUBEND
245   SUB Build_rpt_patt(Block$,Patt_length)
246   !
247   !
248   ! This SUB builds a repeating 4 bit word pattern. You could easily
249   ! change this to an 8, 16, or 32 bit word. If you do this you will
250   ! need to change the TRAP.
251     No_4bit_words=99  !Pattern length must go up in multiples of 32
252                       !above lengths of 256 (64 4bit_words =256 length)
253   !
254   !***********TRAP TO SET PATTERN LENGTH FOR 4 BIT WORDS***********
255     IF No_4bit_words>63 THEN
256       Remainder=No_4bit_words*4 MOD 32
257       IF Remainder>0 THEN
258         No_4bit_words=((32-Remainder)+No_4bit_words*4)/4
259         PRINT "Pattern Length increased to ";No_4bit_words*4
260       END IF
261     END IF
262   !*****************************************************************
263     ALLOCATE Build_patt$(9000)[10]
264     Word$=VAL$(1)&VAL$(1)&VAL$(1)&VAL$(0)      ! 4 BIT WORD
265     !
266     REPEAT
267       Build_patt$(N)=Word$
268       N=N+1
269     UNTIL N=No_4bit_words
270     !
271     N=0
272     Strbit=1
273     !
274     LOOP
275     EXIT IF N=No_4bit_words
```

```
276       FOR Bit=1 TO 4
277         Block=VAL(Build_patt$(N)[Bit,Bit])
278         Block$[Strbit,Strbit]=CHR$(Block)
279         Strbit=Strbit+1
280       NEXT Bit
281       N=N+1
282     END LOOP
283     !
284     !
285     ! BUILD UP BLOCK$
286     Patt_length=LEN(Block$)
287     Patt_length$=VAL$(Patt_length)
288     Number_digits=LEN(Patt_length$)
289     Number_digits$=VAL$(Number_digits)
290     Header$="#"&Number_digits$&Patt_length$
291     Block$=Header$&Block$
292     !
293     !
294   SUBEND
295   SUB Testpatt
296   ! ---------------------------------------------------------------------
297   ! This SUB uses the CTRL key on the computer
298   ! CTRL  SPACE BAR =0,  CTRL A = 1.
299   ! Work out the header for yourself !
300   ! The 10 represents the numberof bits to follow....
301   ! while the 2 preceding the 10 tells you the number of
302   ! digits there are in 10.
303   !---------------------------------------------------------------------
304   !
305   !
306   !*********************************************************************
307   ! If this program is transferred as an ASCII file or printed out the
308   ! CTRL characters become invisible .....there are 10 CTRL characters in
309   ! the OUTPUT line immediately following #210 .
310   !*********************************************************************
311   !
312     OUTPUT 717;":PATT:UPAT3:DATA #210
313   !
314   SUBEND
```

6

TMSL Command Definition Quick Reference

TMSL Command Definition Quick Reference Guide

Introduction

The following pages list the TMSL commands for the HP 71600B Pattern Generator (HP 70841B) and Error Detector (HP 70842B) modules.

The Pattern Generator

The SOURce subsystem

SOURce1: The Data Source

KEYWORD	PARAMETER FORM	COMMENTS
[SOURce[1]:]		
PATTern		
[:SELect]	PRBS<n>|ZSUBstitut<n>|MDENsity<n>|UPATtern<n>|AWORd	
[:SELect]?	PRBS<n>|ZSUB<n>|MDEN<n>|UPAT|AWORd	
:ZSUBstitut		
[:ZRUN]	<numeric value>	
[:ZRUN]?	<NR1>	
:MDENsity		
[:DENSity]	<numeric value>	
[:DENSity]?	<NR3>	
:UPATtern<n>		
[:LENGth]	<numeric value>	
[:LENGth]?	<NR1>	
:LABel	<string>	
:LABel?	<string>	
:USE	STRaight|APATtern	
:USE?	STR|APAT	
:DATA	[A|B,]<block data>	
:DATA?	[A|B,]<block data>	
:IDATa	[A|B,]<start bit>,<length in bits>,<block data>	
:IDATa?	[A|B,]<start bit>,<length in bits>	
:FORMat		

```
      [:DATA]          PACKed,<numeric value>
      [:DATA]?         PACK,<NR1>
   :AWORd
      :DATA<n>         <NRf>{,<NRf>}
      :DATA<n>?        <NR1>{,<NR1>}
   :APCHange
      :SOURce          EXTernal|INTernal
      :SOURce?         EXT|INT
      :MODE            ALTernate|ONEShot
      :MODE?           ALT|ONES
      :SELect          AHALf|BHALf                  only if INT & ALT
      :SELect?         AHAL|BHAL                    only if INT & ALT
      :IBHalf          ONCE                         only if INT & ONES
                                                    event; no query
   :EADDition          ONCE|<boolean>
      :SOURce          EXTernal|FIXed
      :SOURce?         EXT|FIX
      :RATE            <numeric value>
      :RATE?           <NR3>
VOLTage
   [:LEVel]
      [:IMMediate]
         [:AMPLitude]   <numeric value>
         [:AMPLitude]?  <NR3>
         :HIGH          <numeric value>
         :HIGH?         <NR3>
   :ATTenuation         <numeric value>
   :ATTenuation?        <NR3>
   :ECL                   ---                        event; no query
```

SOURce2: The Clock Source

KEYWORD	PARAMETER FORM	COMMENTS
SOURce2		
:FREQuency		
[:CW|:FIXed]?	<NR3>	query only
:VOLTage		
[:LEVel]		
[:IMMediate]		
[:AMPLitude]	<numeric value>	
[:AMPLitude]?	<NR3>	
:HIGH	<numeric value>	
:HIGH?	<NR3>	
:ATTenuation	<numeric value>	
:ATTenuation?	<NR3>	
:ECL	---	event; no query

SOURce3: The Trigger Source

KEYWORD	PARAMETER FORM	COMMENTS
SOURce3		
:TRIGger		
[:MODE]	PATTern|DCLock	
[:MODE]?	PATT|DCL	
:CTDRatio?	<NR3>	query only
:PRBS<n>	<NRf>{,<NRf>}	
:PRBS<n>?	<NR1>{,<NR1>}	
:ZSUBstitut<n>	<numeric value>	
:ZSUBstitut<n>?	<NR1>	
:MDENsity<n>	<numeric value>	
:MDENsity<n>?	<NR1>	
:UPATtern<n>	<numeric value>	
:UPATtern<n>?	<NR1>	
:APATtern<n>	ABCHange|SOPattern	
:APATtern<n>?	ABCH|SOP	

The OUTPut subsystem

OUTPut1: The Data Output

KEYWORD	PARAMETER	COMMENTS
OUTPut[1]		
[:STATe]	<boolean>	
[:STATe]?	<boolean>	
:POLarity	NORMal|INVerted	
:POLarity?	NORM|INV	
:DELay	<numeric value>	
:DELay?	<NR3>	
:TERMination	<numeric value>	
:TERMination?	<NR1>	
:OPTimize	DATA|DADBar	
:OPTimize?	DATA|DADB	

OUTPut2: The Clock Output

KEYWORD	PARAMETER FORM	COMMENTS
OUTPut2		
:TERMination	<numeric value>	
:TERMination?	<NR1>	

The MMEMory subsystem

KEYWORD	PARAMETER FORM	COMMENTS
MMEMory		
:INITialize		
:DELete	<file name>	event; no query
:CATalog?	<NR3>,<NR3>{,<file entry>}	query only
:MPResent?	<boolean>	query only
:CPDisc	<NR1>	event; no query
:ICPDisc	<NR1>,<NR1>,<NR1>,<NR1>	event; no query

The SYSTem subsystem

KEYWORD	PARAMETER FORM	COMMENTS
SYSTem		
:BEEPer		
[:IMMediate]	[<freq>[,<time>[,<vol>]]]	parms no effect
:ERRor?	<NR1>,<string>	query only
:KLOCk	<boolean>	
:KLOCk?	<boolean>	
:PRESet|:PRESet<n>	---	event; no query
:PTHRough		
[:STRing]	<string>	
[:STRing]?	<string>	
:VERSion?	<NR2>	query only

The STATus subsystem

KEYWORD	PARAMETER FORM	COMMENTS
STATus		
:QUEStionable		
[:EVENt]?	<NR1>	query only
:CONDition?	<NR1>	query only
:ENABle	<NRf>	
:ENABle?	<NR1>	
:PTRansition	<NRf>	
:PTRansition?	<NR1>	
:NTRansition	<NRf>	
:NTRansition?	<NR1>	
:PRESet	---	event; no query
:FAILure		
[:EVENt]?	<NR1>	query only
:SSERvice		
[:EVENt]?	<NR1>	query only
:ENABle	<NRf>	
:ENABle?	<NR1>	

IEEE Common Commands

Mandatory Commands

KEYWORD	PARAMETER FORM	COMMENTS
*CLS	---	event; no query
*ESE	<NRf>	
*ESE?	<NR1>	
*ESR?	<NR1>	query only
*IDN?	<string>	query only
*OPC	---	
*OPC?	<NR1>	
*RST	---	event; no query
*SRE	<NRf>	
*SRE?	<NR1>	
*STB?	<NR1>	query only
*TST?	<NR1>	query only
*WAI	---	

Optional Commands

KEYWORD	PARAMETER FORM	COMMENTS
*OPT?	<NR1>	query only
*PSC	<NRf>	
*PSC?	<NR1>	
*RCL	<NRf>	event; no query
*SAV	<NRf>	event; no query

PART 2: Error Detector

The SENSe subsystem

SENSe1: The Data Sense

KEYWORD	PARAMETER FORM	COMMENTS
[SENSe[1]:]		
PATTern		
[:SELect]	PRBS<n>|ZSUBstitut<n>| MDENsity<n>|UPATtern<n>	
[:SELect]?	PRBS<n>|ZSUB<n>| MDEN<n>|UPAT	
:ZSUBstitut		
[:ZRUN]	<numeric value>	
[:ZRUN]?	<NR1>	
:MDENsity		
[:DENSity]	<numeric value>	
[:DENSity]?	<NR3>	
:UPATtern<n>		
[:LENGth]	<numeric value>	
[:LENGth]?	<NR1>	
:LABel	<string>	
:LABel?	<string>	
:USE	STRaight|APATtern	
:USE?	STR|APAT	
:DATA	[A|B,]<block data>	
:DATA?	[A|B,]<block data>	
:IDATa	[A|B,]<start bit>,<length in bits>,<block data>	
:IDATa?	[A|B,]<start bit>,<length in bits>	
:FORMat		
[:DATA]	PACKed,<numeric value>	
[:DATA]?	PACK,<NR1>	
VOLTage		
:ZOTHreshold	<numeric value>	
:ZOTHreshold?	<NR3>	
:AUTO	<boolean>	
:AUTO?	<boolean>	
GATE		
[:STATe]	<boolean>	
[:STATe]?	<boolean>	
:MODE	MANual|SINGle|REPetitive	
:MODE?	MAN|SING|REP	
:MANNer	TIME|ERRors|BITS	
:MANNer?	TIME|ERR|BITS	
:PERiod		
[:TIME]	<numeric value>	
[:TIME]?	<NR1>	

```
        :ERRors             <numeric value>
        :ERRors?            <NR1>
        :BITS               <numeric value>
        :BITS?              <NR3>
    SYNChronisat            ONCE|<boolean>
    SYNChronisat?           <boolean
       :THReshold           <numeric value>
       :THReshold?          <NR3>
    LOGGing                 ONCE|<boolean>
    LOGGing?                <boolean>
       :SQUelch             <boolean>
       :SQUelch?            <boolean>
       :ALARms              <boolean>
       :ALARms?             <boolean>
       :THReshold           <numeric parm>
       :THReshold?          <NR3>
       :DURing
          [:EVENt]          NEVer|ESECond|ERGThrshld
          [:EVENt]?         NEV|ESEC|ERGT
       :END
          [:EVENt]          NEVer|ALWays|NZECount|TERGthrshld
          [:EVENt]?         NEV|ALW|NZEC|TERG
          :REPort           FULL|UREP
          :REPort?          FULL|UREP
    EYE
       :TCENter|:TCENtre    ONCE|<boolean>
       :TCENter?:|TCENtre?  <boolean>
       :ACENter|:ACENtre    ONCE|<boolean>
       :ACENter?:|ACENtre?  <boolean>
       :WIDTh?              <NR3>                    query only
       :HEIGht?             <NR3>                    query only
       :THReshold           <numeric value>
       :THReshold?          <NR3>
```

SENSe2: The Clock Sense

```
    KEYWORD                 PARAMETER FORM                COMMENTS


  SENSe2
     :VOLTage
        :EDGE               POSitive|NEGative
        :EDGE?              POS|NEG
```

The INPut subsystem

INPut1: The Data Input

KEYWORD	PARAMETER FORM	COMMENTS
INPut[1]		
:POLarity	NORMal|INVerted	
:POLarity	NORM|INV	
:DELay	<numeric value>	
:DELay?	<NR3>	
:TERMination	<numeric value>	
:TERMination?	<NR1>	

INPut2: The Clock Input

KEYWORD	PARAMETER FORM	COMMENTS
INPut2		
:TERMination	<numeric value>	
:TERMination?	<NR1>	

The Measurement subsystem

KEYWORD	PARAMETER FORM	COMMENTS
FETCh|PFETch		
[:SENSe[1]]		
:ECOunt		
[:ALL]		
[:TOTal]?	<NR3>	query only
:DELTa?	<NR3>	query only
:ZASone		
[:TOTal]?	<NR3>	query only
:OASZero		
[:TOTal]?	<NR3>	query only
:ERATio		
[:ALL]		
[:TOTal]?	<NR3>	query only
:DELTa?	<NR3>	query only
:ZASone		
[:TOTal]?	<NR3>	query only
:OASZero		
[:TOTal]?	<NR3>	query only
:EINTerval		
:SEConds?	<NR3>	query only
:DSEConds?	<NR3>	query only
:CSEConds?	<NR3>	query only
:MSEConds?	<NR3>	query only

```
      :EFINterval
         :SEConds?          <NR3>                    query only
         :DSEConds?         <NR3>                    query only
         :CSEConds?         <NR3>                    query only
         :MSEConds?         <NR3>                    query only
      :LOSS
         :POWer?            <NR3>                    query only
         :SYNChronisat?      <NR3>                   query only
      :G821
         :AVAilability?      <NR3>                   query only
         :UNAVailabili?      <NR3>                   query only
         :SESeconds?         <NR3>                   query only
         :DMINutes?          <NR3>                   query only
         :ESEConds?          <NR3>                   query only
      :GATE
         :ELAPsed?           <NR3>                   query only
      :LTEXt?                <string>                query only
   :SENSe2
      :FREQuency?            <NR3>                   query only
```

DISPlay subsystem

```
   KEYWORD                   PARAMETER FORM            COMMENTS


DISPlay
   :SHOW                     PGENerator|EDETector|BOTH
   :SHOW?                    PGEN|EDET|BOTH
   :PAGE                     USER|ISTatus|MSTatus|LSTatus|
                             MRESults|IRESults|G821|ZOOZ results
   :PAGE?                    USER|IST|MST|LST|MRES|IRES|G821|ZOOZ
   :REPort                   PREVious|CURRent
   :REPort?                  PREV|CURR
   :UPAGe
      [:DEFine]              ECOunt|
                             DCOunt|
                             ERATio|
                             DRATio|
                             GELapsed|
                             BECount|
                             BEDCount|
                             BERatio|
                             BEDRatio|
                             BGELapsed|
                             OECount|
                             ZECount|
                             OERatio|
                             ZERatio|
                             DTIMe|
                             PIDentity|
```

```
                                ERSeconds|
                                EDSeconds|
                                ECSeconds|
                                EMSeconds|
                                CFRequency|
                                EFSeconds|
                                EFDSeconds|
                                EFCSeconds|
                                EFMSeconds|
                                BLANk|
                                AVAilability|
                                UNAVailabili|
                                SESeconds|
                                ERDSeconds|
                                DMINutes|
                                SMODe|
                                ZOTMode|
                                DTERmination|       for backwards compatibility
                                TERMination|
                                DPOLarity|
                                DIDelay|
                                CEDGe|
                                GMODe|
                                GREPort|
                                GPERiod|
                                PLSeconds|
                                SLSeconds|
                                LGSTatus|
                                ALOGging|
                                LDTRigger|
                                LETRigger|
                                LEReport|
                                LTHReshold|
                                SSTatus|
                                HCONtroller|
                                EETHreshold|
                                EWIDth|
                                EHEight
      [:DEFine]?                <boolean>
      :CLEar                      ---                     event; no query
```

The SYSTem subsystem

```
    KEYWORD                   PARAMETER FORM                  COMMENTS

SYSTem
   :BEEPer
      [:IMMediate]          [<freq>[,<time>[,<vol>]]]   parms no effect
      :STATe                 <boolean>
      :STATe?                <boolean>
```

:ERRor?	<NR1>,<string>	query only
:KLOCk	<boolean>	
:KLOCk?	<boolean>	
:PRESet|:PRESet<n>	---	event; no query
:PTHRough		
[:STRing]	<string>	
[:STRing]?	<string>	
:VERSion?	<NR2>	query only
:DATE	<year>,<month>,<day>	
:DATE?	<year>,<month>,<day>	
:TIME	<hour>,<minute>,<second>	
:TIME?	<hour>,<minute>,<second>	
:FREVision		
:MPRocessor	<string>	query only

The STATus subsystem

KEYWORD	PARAMETER FORM	COMMENTS
STATus		
:OPERtionable		
[:EVENt]?	<NR1>	query only
:CONDition?	<NR1>	query only
:ENABle	<NRf>	
:ENABle?	<NR1>	
:PTRansition	<NRf>	
:PTRansition?	<NR1>	
:NTRansition	<NRf>	
:NTRansition?	<NR1>	
:QUEStionable		
[:EVENt]?	<NR1>	query only
:CONDition?	<NR1>	query only
:ENABle	<NRf>	
:ENABle?	<NR1>	
:PTRansition	<NRf>	
:PTRansition?	<NR1>	
:NTRansition	<NRf>	
:NTRansition?	<NR1>	
:PRESet	---	event; no query
:FAILure		
[:EVENt]?	<NR1>	query only
:SSERvice		
[:EVENt]?	<NR1>	query only
:ENABle	<NRf>	
:ENABle?	<NR1>	

IEEE Common Commands

Mandatory Commands

KEYWORD	PARAMETER FORM	COMMENTS
*CLS	---	event; no query
*ESE	<NRf>	
*ESE?	<NR1>	
*ESR?	<NR1>	query only
*IDN?	<string>	query only
*OPC	---	
*OPC?	<NR1>	
*RST	---	event; no query
*SRE	<NRf>	
*SRE?	<NR1>	
*STB?	<NR1>	query only
*TST?	<NR1>	query only
*WAI	---	

Optional Commands

KEYWORD	PARAMETER FORM	COMMENTS
*OPT?	<NR1>	query only
*PSC	<NRf>	
*PSC?	<NR1>	
*RCL	<NRf>	event; no query
*SAV	<NRf>	event; no query

SCPI Conformance Information

SCPI Conformance Information

This section details how the Hewlett-Packard 71600B Series of Gbit/s Testers conform to *Standard Commands for Programmable Instruments (SCPI)*. It lists separately:

- The SCPI version to which the instruments comply.
- The commands confirmed by SCPI.
- The commands approved by SCPI.
- The commands which are not yet part of the SCPI definition.

SCPI Version

The HP 71600B Series of Gbit/s Testers complies with SCPI-1990.0

SCPI Confirmed Commands

The following commands are confirmed by SCPI.

```
[SOURce[1]:]
    VOLTage
        [:LEVel]
            [:IMMediate]
                [:AMPLitude]
                [:AMPLitude]?
                :HIGH
                :HIGH?
        :ATTenuation
        :ATTenuation?
     PATTern
        :FORMAT
                [:DATA]
                [:DATA]?

SOURce2
    :FREQuency
        [:CW|:FIXed]?
    :VOLTage
        [:LEVel]
```

```
                [:IMMediate]
                    [:AMPLitude]
                    [:AMPLitude]?
                    [:HIGH]
                    [:HIGH]?
            :ATTenuation
            :ATTenuation?

    OUTPut[1]
            [:STATE]
            [:STATE]?

    MMEMory
            :INITialize
            :DELete
            :CATalogue?

    [SENSe[1]:]
        PATTern
             :FORMat
                    [:DATA]
                    [:DATA]?

    SYSTem
        :BEEPer
            [:IMMediate]
        :ERRor?
        :KLOCk
        :KLOCk?
        :PRESet
        :VERSion?
        :DATE
        :DATE?
        :TIME
        :TIME?

    STATus
        :QUEStionable
            [:EVENt]?
            :CONDition
            :CONDition?
            :ENABle
            :ENABle?
            :PTRansition
            :PTRansition?
            :NTRansition
            :NTRansition?
            :PRESet
```

```
    :OPERation
        [:EVENt]?
        :CONDition
        :CONDition?
        :ENABle
        :ENABle?
        :PTRansition
        :PTRansition?
        :NTRansition
        :NTRansition?
        :PRESet
*CLS
*ESE
*ESE?
*ESR?
*IDN?
*OPC
*OPC?
*RST
*SRE
*SRE?
*STB?
*TST?
*WAI
*OPT?
*PSC
*PSC?
*RCL
*SAV
```

SCPI Approved Commands

There are no commands in this category.

Non SCPI Commands

The following commands are not yet part of the SCPI standard.

```
[SOURce[1]:]
    PATTern
        [:SELect]
        [:SELect]?
        :ZSUBstitut
            [:ZRUN]
            [:ZRUN]?
        :MDENsity
            [:DENSity]
            [:DENSity]?
        :UPATtern<n>
            [:LENGth]
            [:LENGth]?
            :LABel
            :LABel?
            :USE
            :USE?
            :DATA
            :DATA?
            :IDATa
            :IDATa?
        :AWORd
            :DATA<n>
            :DATA<n>?
         :APCHange
            :SOURce
            :SOURce?
            :MODE
            :MODE?
            :SELect
            :SELect?
            :IBHalf
        :EADDition
            :SOURce
            :SOURce?
            :RATE
            :RATE?
    VOLTage
        :ECL

SOURce2:]
    VOLTage
        :ECL

SOURce3:
    TRIGger
        [:MODE]
```

```
        [:MODE]?
        :CTDRatio?
        :PRBS<n>
        :PRBS<n>?
        :ZSUBstitut<n>
        :ZSUBstitut<n>?
        :MDENsity<n>
        :MDENsity<n>?
        :UPATtern<n>
        :UPATtern<n>?
        :APATtern<n>
        :APATern<n>?

OUTPut[1]
    :POLarity
    :POLarity?
    :DELay
    :DELay?
    :TERMination
    :TERMination?
    :OPTimize
    :OPTimize?

OUTPut2
     :TERMination
     :TERMination?

SYSTem
    :PTHRough
        [:STRing]
        [:STRing]?
    :FREVision
        :MPRocessor?
    MMEMORY
        :MPResent?
        :CPDisc<n>
        :ICPDisc<n>,<n>,<n>,<n>
STATus
    :FAILure
        [:EVENt]
        [:EVENt]?
     :SSERvice
        [:EVENt]
        [:EVENt]?
         :ENABLE
         :ENABLE?

[SENSe[1]:]
    VOLTage
        :ZOTHreshold
        :ZOTHreshold?
```

```
                :AUTO
                :AUTO?
        GATE
            [:STATe]
            [:STATe]?
            :MODE
            :MODE?
            :MANNER
            :MANNER?
            :PERiod
                [:TIME]
                [:TIME]?
                 :ERRors
                 :ERRors?
                 :BITS
                 :BITS?

        SYNChronisat
        SYNChronisat?
            :THReshold
            :THReshold?
        LOGGing
        LOGGing?
            :SQUelch
            :SQUelch?
            :ALARms
            :ALARms?
            :THReshold
            :THReshold?
            :DURing
                [:EVENt]
                [:EVENt]?
            :END
                [:EVENt]
                [:EVENt]?
                :REPort
                :REPort?

        EYE
            :TCENter
            :TCENter?
            :ACENter
            :ACENter?
            :WIDTh?
            :HEIGht?
            :THReshold
            :THReshold?
    SENSe2
        :VOLTage
            :EDGE
            :EDGE?
```

```
INPut[1]
    :POLarity
    :POLarity?
    :DELay
    :DELay?
    :TERMination
    :TERMination?

 INPut2
    :TERMination
    :TERMination?

FETCh|PFETch
    [:SENSe[1]]
        :ECOunt
            [:ALL]
               [:TOTal]?
               :DELTa?
            :ZASone
               [:TOTAL]?
            :OASZero
               [:TOTal]?
        :ERATio
            [:ALL]
               [:TOTal]?
                :DELTa?
             :ZASone
               [:TOTal]?
             :OASZero
               [:TOTal]?

        :EINTerval
            :SEConds?
            :DSEConds?
            :CSEConds?
            :MSEConds?
        :EFINterval
            :SEConds?
            :DSEConds?
            :CSEConds?
            :MSEConds?
        :LOSS
            :POWer?
            :SYNChronisat?
        :G821
            :AVAilability?
            :UNAVailabili?
            :SESeconds?
            :DMINutes?
            :ESEConds?
        :GATE
```

```
            :ELAPsed?
        :LTEXt?
    :SENSe2
        :FREQuency?
DISPlay
    :SHOW
    :SHOW?
    :PAGE
    :PAGE?
    :REPort
    :REPort?
    :UPAGe
        [:DEFine]
        [:DEFine]?
        :CLEar
```

SCPI 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.

0 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 unimplemented *IEEE 488.2* common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of 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, *EMC 1 :CH1:VOLTS 5.

−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 *EMC common command only accepts one parameter, so receiving *EMC 0,1 is not allowed.

−109 Missing parameter

Fewer parameters were received than required for the header; for example, the *EMC common command requires one parameter, so receiving *EMC 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 *GMC"MACRO" 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 nonheader 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 nondecimal 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 of 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 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 [−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.)