Programmer Manual

SONY Tektronix

DG2020A
Data Generator

071-0054-01

Printed in Japan.

Sony/Tektronix Corporation, P.O.Box 5209, Tokyo Int'l, Tokyo 100–31 Japan

Tektronix, Inc., P.O. Box 1000, Wilsonville, OR 97070–1000

Table of Contents

Appendices

Glossary & Index

List of Figures

List of Tables

Preface

This is the Programmer Manual for the DG2020A Data Generator and Pods. This manual provides information on operating these instruments using General Purpose Interface Bus (GPIB) interface and RS-232-C interface.

This manual provides the following information:

- *Getting Started* describes how to connect and set up for remote operation.
- *Syntax and Commands* defines the command syntax and processing conventions and describes each command in the data generator command set.
- *Status and Events* explains the status information and event messages reported by the data generator.
- *Appendices* contains various topics of use to the programmer.
- *Glossary and Index* contains a glossary of common terms and an index to this manual.

Related Manuals

Other documentation for the data generator includes:

- The *User Manual* that describes the operation of the Data Generator that was supplied as a standard accessory with the instrument.
- The *Service Manual* (optional accessory) provides information for maintaining and servicing the Data Generator.

Getting Started

Getting Started

Overview

The Data Generator has two interfaces for remote operation — the GPIB interface and the RS-232-C interface. All menu controlled and front-panel controlled functions, except the ON/STBY function, the edit function, and the GPIB and RS-232-C parameter setup functions, can be controlled through the GPIB or the RS-232-C interface using the programming command set (see Section 2).

The GPIB interface conforms to ANSI/IEEE Std 488.1-1987, which specifies the hardware interface, its basic functional protocol, and a set of interface messages (codes) that control the interface functions. This instrument also conforms to ANSI/IEEE Std 488.2-1987 which specifies Codes, Formats, Protocols, and Common Commands to support the system application. The functional layers of the GPIB system are shown in Figure 1-1.

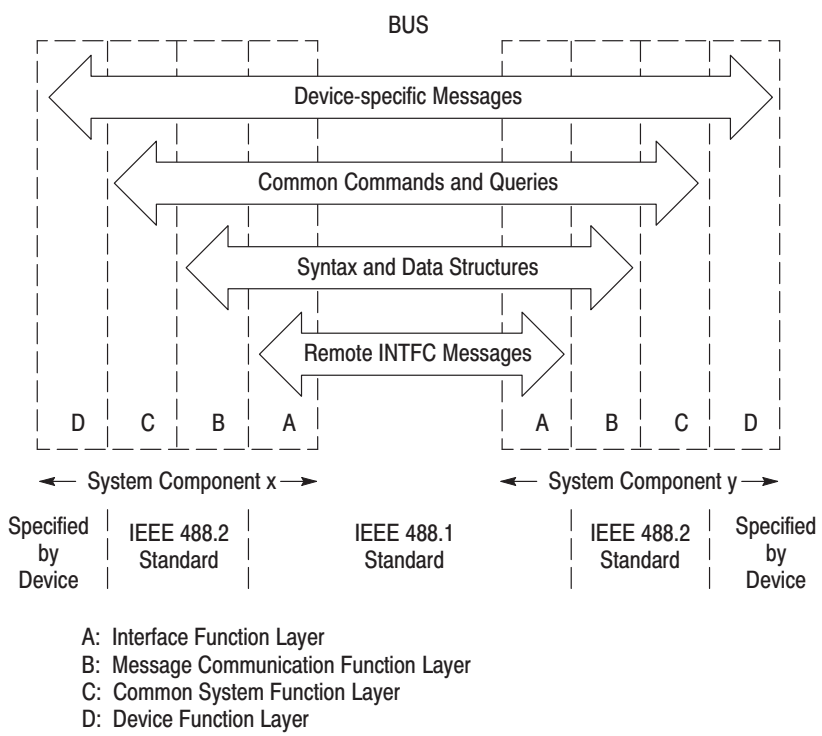


Figure 1-1: Functional layers in gpib system

The RS-232-C interface, which was established by the Electronic Industries Association (EIA), provides a common basis of communication between devices that exchange data. This interface has long been used on terminals, modems, printers, and other devices. The RS-232-C interface that the data generator provides also uses most of the same Codes, Formats, Protocols, and Common Commands as are used with the GPIB interface (ANSI/IEEE Std 488.2-1987).

Choosing an Interface

Your system hardware may let you choose which interface to use with your system; if so, you should consider the comparative advantages and disadvantages of each interface. For example, the GPIB interface is an eight-bit parallel bus and therefore it offers high-speed data transfers and multiple instrument control. In contrast, the RS-232-C interface is a slower serial data bus for single instrument control, but it is easy to connect to and can be used with a low-cost controller. Table 1-1 compares the GPIB and RS-232-C interface.

Table 1-1: GPIB and RS-232-C comparison

Operating attribute	GPIB	RS-232-C
Cable	ANSI/IEEE Std 488	9-wire (DCE)
Data flow control	Hardware, 3-wire handshake	Flagging: soft (XON/XOFF), hard (DTR/CTS)
Data format	8-bit parallel	8-bit serial
Interface control	Operator low-level control message	None
Interface messages	Most ANSI/IEEE Std 488	Device clear via ASCII break signal
Interrupts reported	Service requests status and event code	Status and event code (no service requests)
Message termination (Receive)	Hardware EOI, software LF, or both	Software CR, LF, or CR and LF
Message termination (Transmit)	Hardware EOI, and software LF	Software LF
Timing	Asynchronous	Asynchronous
Transmission path length	≤2 meters between devices; ≤20 meters total cabling for GPIB system	≤15 meters
Speed	200 Kbytes/sec	19,200 bits/sec
System environment	Multiple devices (≤15)	Single terminal (point to point connection)

Installing for GPIB Communication

With the power off, connect a GPIB cable from the GPIB controller to the ANSI/IEEE Std 488 port (GPIB) connector on the rear panel of the data generator (see Figure 1-2). For example, when using an MS-DOS compatible controller, connect the GPIB cable between the National Instrument PC2A GPIB board and the data generator GPIB connector.

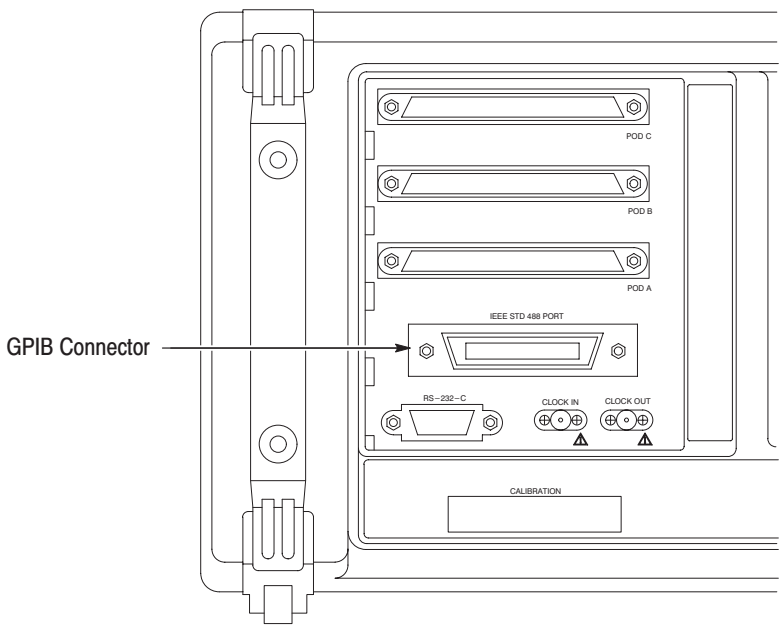


Figure 1-2: GPIB connector

Instruments can be connected to the GPIB in linear or star configurations or in a combination of both configurations. A linear hookup is one where a GPIB cable is used to string one device to a second, and then another GPIB cable is used to string from a second to a third, and so on until all devices in the system are connected. A star setup is one where one end of all the GPIB cables in the system are attached to one device. Refer to Figure 1-3 for these GPIB system configurations.

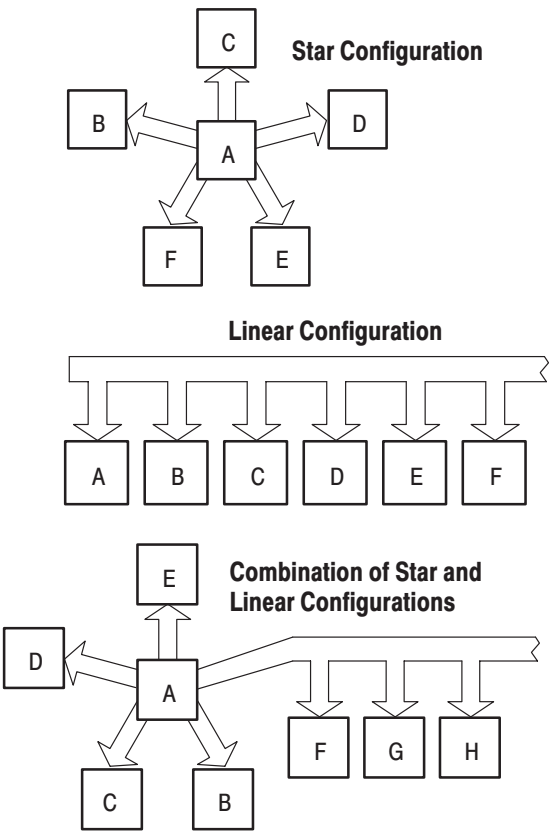


Figure 1-3: GPIB system configurations

Restrictions

Consider the following rules when distributing instruments on the GPIB:

1. No more than 15 total devices (including the controller) can be included on a signal bus.
2. In order to maintain the electrical characteristics of the bus, one device load must be connected for every two meters of cable (most often, each device represents one device load to the bus).
3. The total cable length (cumulative) must not exceed 20 meters.
4. At least two-thirds of the device loads must be powered on.

Setting the GPIB Parameters

To set the GPIB parameters, proceed as follows:

1. Press the UTILITY button in the MENU column to the right of the screen. The UTILITY menu appears above the bottom menu buttons.
2. Press the System bottom menu button to display the System menu (See figure 1-4).
3. Select the Configure item from the GPIB menu using the up and down arrow buttons. Set the GPIB operating mode using the left and right arrow buttons.
   - Talk/Listen. Sets the communications mode to talk/listen.
   - Talk Only. Sets the communications mode to talk only, which is used for hardcopy output.
   - Off Bus. Logically disconnect the data generator from GPIB system.

***NOTE**. The data generator accepts as a terminator either the software LF (Line Feed), sent as the last data byte, or the hardware EOI, with the EOI line asserted concurrently with the last data byte sent.*

4. Select the Address item from the GPIB menu using the up and down arrow buttons. Then use the rotary knob to set the primary address to a value in the range 0 to 30.
5. Select the Remote Port item using the up and down arrow buttons, and additionally, highlight ”GPIB” using the left and right arrow buttons. This selects the GPIB as the remote interface.

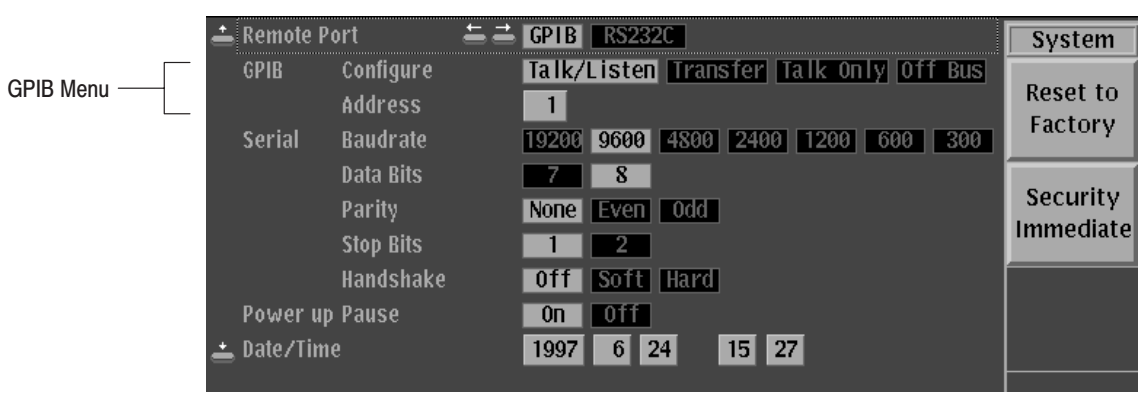


Figure 1-4: GPIB parameter settings

Installing for RS-232-C Communication

Connect an RS-232-C cable from the computer terminal to the RS-232-C connector on the rear panel of the data generator. Use a configuration based on the settings for the data flow control (flagging).

The RS-232-C provides a point-to-point connected communication interface between devices (see Figure 1-5). The data generator can transmit and receive the same message serially over the RS-232-C interface as it can in parallel over the GPIB interface.

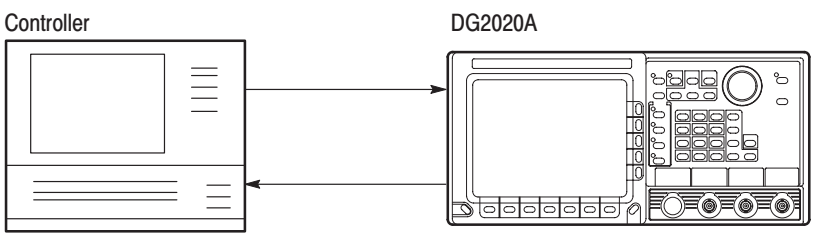


Figure 1-5: RS-232-C point-to-point connection

Several connectors are used with the RS-232-C interface: a DTE device uses a standard 25-pin male D-type shell connector; a DCE device uses a standard 25-pin female D-type shell connector. Some recent computers implement the RS-232-C interface using 9-pin D-type connector.

This data generator uses a standard 9-pin D-type shell connector, provided on the rear panel (see Figure 1-6), along with a 9-pin male to 25-pin male conversion cable. Figure 1-7 on page 1-8 shows both 9-pin and 25 pin connectors with their pin number assignments.

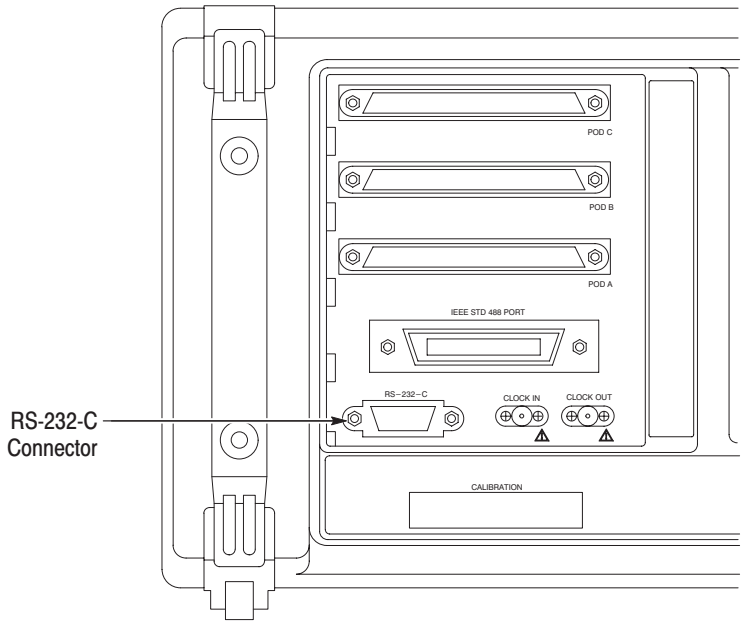
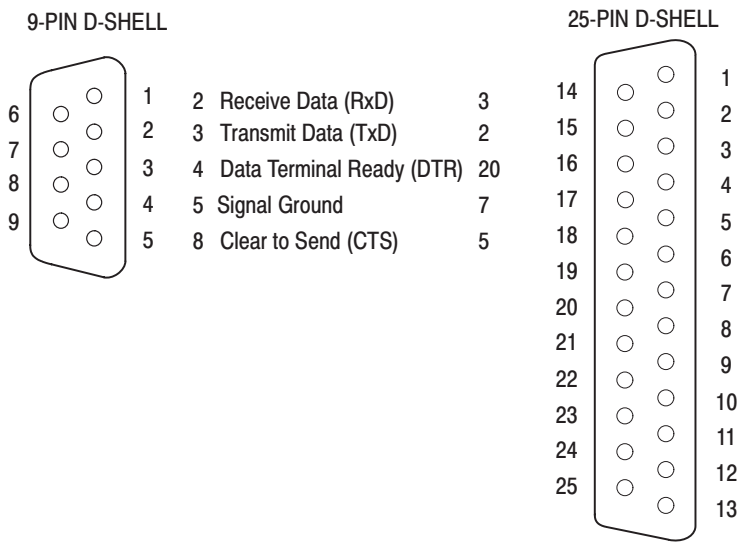


Figure 1-6: RS-232-C port

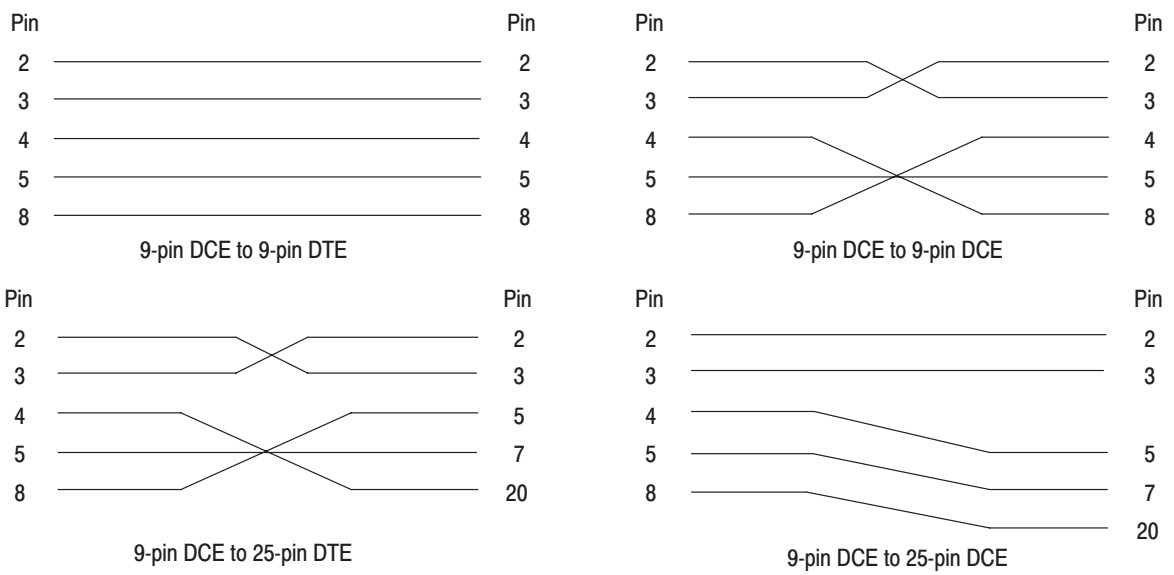
This data generator is designed as DCE device. You may connect it up to 15 meters (50 feet) from a DTE device using a straight-through male-to-female cable. However, if the other device is instead configured as a DCE device, you will need a special adapter or null-modem cable for local DCE-to-DCE communications. Refer to the wiring examples in the Figure 1-8 for the proper signal connections between devices.

***NOTE**. In this data generator, only TxD, RxD, DTR, CTS pins and Signal Ground are available.*



NOTE: *TxD, RxD, DTR, CTS and Ground lines are only available in the data generator.*

Figure 1-7: Pin assignments of 9-pin and 25-pin D-type shell connector



NOTE: *When using software flow control, the CTS-DTR lines do not need to be connected.*

Figure 1-8: Typical RS-232-C cable wiring requirements

Setting the RS-232 Parameters

To set the RS-232-C parameters, perform the following steps:

1. Press the UTILITY button in the MENU column to the right of the screen. The UTILITY menu appears above the bottom menu buttons.
2. Press the System bottom menu button to display the System menu (See figure 1-9).
3. Select the Baudrate item from the Serial menu using the up and down arrow buttons. Here select the data transfer rate using the left and right arrow buttons. The rate can be set to 300, 600, 1200, 2400, 4800, 9600, or 19200 baud.
4. Select the Data Bits item from the Serial menu using the up and down arrow buttons. Then use the left and right arrow buttons to select the data bit length for each character. The bit length can be set to either 7 or 8 bits.
5. Select the Parity item from the Serial menu using the up and down arrow buttons. Then use the left and right arrow buttons to set the error check bit for each character. The error bit can be set to None, Even, or Odd parity.
6. Select the Stop Bits item from the Serial menu using the up and down arrow buttons. Then use the left and right arrow buttons to select the number of stop bits sent after each character. The number of stop bits can be set to either 1 or 2.
7. Select the Handshake item from the Serial menu using the up and down arrow buttons. Then use the left and right arrow buttons to select the method of controlling the flow of data between devices. The data flow method can be set to Hard (DTR/CTS), Soft (XON/XOFF), and Off (no flow control).
8. Select the Remote Port item using the up and down arrow buttons, and additionally, highlight ”RS232C” using the left and right arrow buttons. This selects the RS-232-C interface as the remote interface.

Figure 1-9: RS-232-C parameter settings

Command Syntax

Command Syntax

A large set of commands can be used to control the operations and functions of the data generator from an external controller. This section describes the syntax and communication rules for using these commands to operate the data generator.

Command Notation

The command syntax is in extended BNF (Backus-Naur Form) notation. The extended BNF symbols used in the command set are shown in the following table.

Table 2-1: BNF symbols and meanings

Symbol	Meaning
< >	Indicates a defined element
|	Delimits Exclusive OR elements
{ }	Delimits a group of elements one of which the programmer must select
[ ]	Delimits an optional element that the programmer may omit
[ ]...	Delimits an optional element that the programmer may omit or may repeat one or more times
::=	Indicates that the left member is defined as shown by the the right member

Program and Response Messages

Programs created or placed in an external controller are transferred to the data generator as a program message. A program message is a sequence of zero or more program message units delimited by the program message unit delimiter, the semicolon (;).

A program message unit is a set command or query command. The data generator performs a function or changes a setting or mode when it receives a set command; when it receives a query command, it returns measurement data, settings, status codes and/or status messages. The data generator transfers these response messages to the external controller.

Command and Query Structure

Commands are either set commands or query commands (usually just called commands and queries in this manual). Most commands have both a set form and query form. The query form of a command is the same as the set form, except that the query form ends with a question mark.

Figure 2-1 shows a flowchart of the structure of the commands and queries. The structure of the header is described in detail in *Header* on page 2-6.

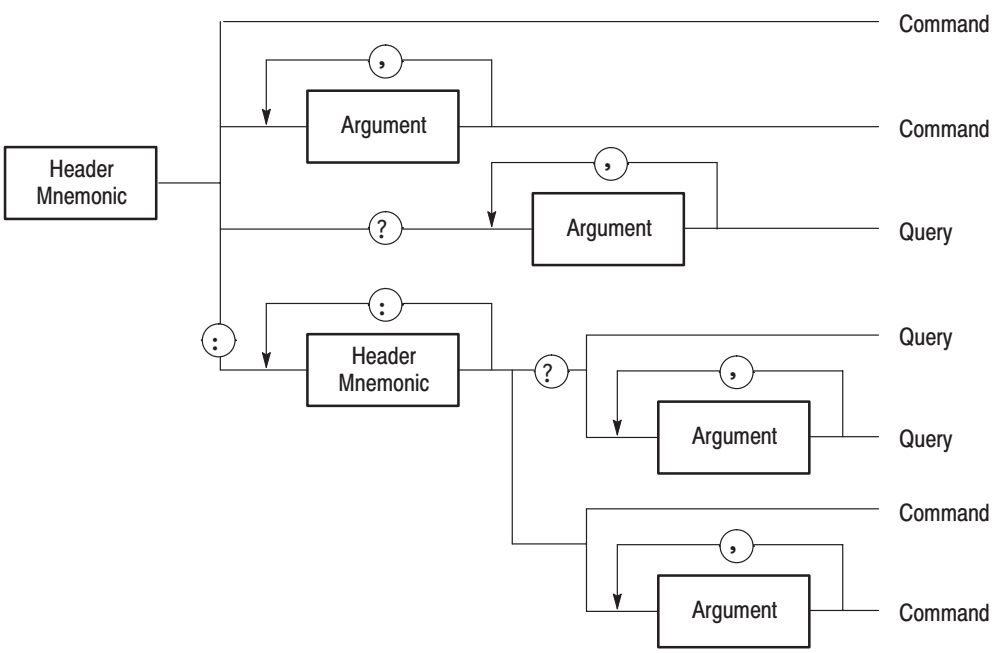


Figure 2-1: Command and query structure flowchart

Character Encoding

The program can be described using the American Standard Code for Information Interchange (ASCII) character encoding.

This seven-bit ASCII code is used for the majority of syntactic elements and semantic definitions. In special cases, an eight-bit ASCII Code is allowed in the arbitrary block arguments described on page 2-5. The ASCII code character set table is found in Appendix A.

Syntactic Delimiters

Syntactic elements in a program message unit are delimited (differentiated) with colons, white space, commas, or semicolons.

Colon (:). Typically delimits the compound command header.

```
MMEMORY:DELETE:ALL, SOURCE:OSCILLATOR:SOURCE
```

White Space. Typically delimits command/query headers from the argument.

```
DIAGNOSTIC:SELECT ALL
SYSTEM:DATE 1995,3,4
```

`DIAGNOSTIC:SELECT` and `SYSTEM:DATE` are the command headers, and `ALL` and `1995,3,4` are the arguments.

Comma (,). Typically delimits between multiple arguments. In the above example, a comma delimits the multiple arguments `1995`, `3` and `4`.

Semicolon (;). Typically delimits between multiple commands (or multiple program message units). For more information about using the semicolon, refer to *Concatenating Commands* on page 2-8.

White Space

White space, which is used to delimit certain syntactic elements in a command, is defined in the data generator as a single ASCII-encoded byte in the range ASCII 0-32 (decimal). This range consists of the standard ASCII characters exclusively except for ASCII 10, which is the Line Feed (LF) or New Line (NL) character.

Special Characters

The Line Feed (LF) character or the New Line (NL) character (ASCII 10) and all characters in the range of ASCII 127-255 are defined as special characters. These characters are used in arbitrary block arguments only; using these characters in other parts of any command yields unpredictable results.

Arguments

In a command or query, one or more arguments follow the command header. The argument, sometimes called program data, is a quantity, quality, restriction, or limit associated with the command or query header. Depending on the command or query header given, the argument is one of the following types:

- Decimal Numeric
- String
- Arbitrary Block

Decimal Numeric

The data generator defines a decimal numeric argument as one expressed in one of three numeric representations — NR1, NR2, or NR3. This definition complies with that found in ANSI/IEEE Std 488.2-1987. Any commands that use arguments in any of the the first three notations can use a fourth notation NRf (for Numerical Representation flexible). The four formats are shown in Table 2-2.

Table 2-2: Decimal numeric notation

Type	Format	Examples
NR1	implicit-point (integer)	1, +3, –2, +10, –20
NR2	explicit-point unscaled (fixed point)	1, 2, +23.5, –0.15
NR3	explicit-point scaled (floating point)	1E+2, +3.36E–2, –1.02E+3
NRf	numeric representation-flexible; any of NR1, NR2, and NR3 may be used	1, +23.5, –1.02E+3

As just implied, you can use NRf notation for arguments in your programs for any commands that this manual lists as using any of NR1, NR2, or NR3 notation in its arguments. Be aware, however, that query response will still be in the format specified in the command. For example, if the command description is `:DESE <NR1>`, you can substitute NR2 or NR3 when using the command in a program. However, if you use the query `:DESE?`, the data generator will respond in the format <NR1> to match the command description in this manual.

Unit and SI Prefix

If the decimal numeric argument refers to a voltage or frequency, you can express it using SI units instead of in the scaled explicit point input value format <NR3>. (SI units are units that conform to the Systeme International d'Unites standard.) For example, you can use the input format 200mV or 1.0MHz instead of 200.0E–3 or 1.0E+6, respectively, to specify voltage or frequency.

You can omit the unit, but you must include the SI unit prefix. You can use either upper or lowercase units.

V or v for voltage

Hz, HZ, or hz for frequency

The SI prefixes, which must be included, are shown below. Note that either lower or upper case prefixes can be used.

SI prefix[1]	m/M	k/K	m/M
Corresponding Power	10^{-3}	10^{3}	10^{6}

1 Note that the prefix m/M indicates 10^{-3} when the decimal numeric argument denotes voltage, but 10^{6} when it denotes frequency.

String

String, sometimes referred to as a string literal, a literal, or just a string, is defined as a series of characters enclosed by double quotation marks (") as in:

```
"This is a string constant"    or    "0 .. 127"
```

To include a double quoted character in the string, insert an additional double quote character ahead of the double quote character in the string. For example, the string:

```
serial number "B010000"
```

would be defined as:

```
"serial number ""B010000"""
```

Single quotation marks (') can also be used instead of double quotation marks. For instance:

```
'serial number ''B010000'''
```

String constants may be of any length up to the memory limits of the instrument in which the message is parsed.

Arbitrary Block

An arbitrary block argument is defined as:

```
#<byte count digit><byte count>[<contiguous eight-bit data
byte>]...
```

or:

```
#<contiguous eight-bit data byte]... <terminator>
```

where:

<byte count digit>::= a nonzero digit in the range ASCII 1–9 that defines the number of digits (bytes) in the <byte count> field.

<byte count>::= any number of digits in the range ASCII 0–9 that define how many bytes are in the <contiguous 8-bit data byte> field.

<contiguous 8-bit data byte>::= a <byte count> number of 8-bit bytes in the range ASCII 0–255 that define the message. Each byte defines one character.

<terminator>::= a software LF followed by a hardware EOI. For example,

```
#16AB4ZLT<LF><&EOI>
```

Header

Header Mnemonic

The header mnemonic represents a header node or a header subfunction. The command or query header comprises one or more header mnemonics that are delimited with the colon (:).

Pod and Channel Representation

The pod and channel can be specified by using the OUTPut:POD<s>:CH<n> header mnemonic in commands and query commands. The term <s> is either A, B, or C, and expresses the connected pattern data output connector for the pod being specified. The term <n> is a number between 0 and 11 that expresses the specified channel.

Header Structure

Commands and queries can be structured into six basic forms.

- Simple command header
- Simple query header
- Compound command header
- Compound query header
- Common command header
- Common query header

Figure 2-1 on page 2-2 shows the syntax for all possible structures, and each of the six basic forms are explained below.

Simple Command Header. A command that contains only one header mnemonic. It may also contain one or more arguments. Its message format is:

```
[:]<Header Mnemonic> [<Argument>[,<Argument>]...]
```

such as:

```
START
```

or

```
STOP
```

Simple Query Header. A command that contains only one header mnemonic followed by a question mark (?). Its message format is:

```
[:]<Header Mnemonic>? [<Argument>[,<Argument>]...]
```

such as:

```
HCOPY?
```

or

```
TRIGGER?
```

Compound Command Header. A command that contains multiple header mnemonics plus argument(s). Its message format is:

```
[:]<Header Mnemonic>[:<Header Mnemonic>]...
[<Argument>[,<Argument>]...]
```

such as:

```
MMEMORY:INITIALIZE HD1
```

or

```
SYSTEM:SECURITY:STATE ON
```

Compound Query Header. A command that contains multiple header mnemonics followed by a question mark (?). Its message format is:

```
[:]<Header Mnemonic>[:<Header Mnemonic>]...?
[<Argument>[,<Argument>]...]
```

such as:

```
DIAGNOSTIC:RESULT?
```

or

```
DATA:BLOCK:SIZE? "BLOCK1"
```

Common Command Header. A command that precedes its header mnemonic with an asterisk (*). Its message format is:

```
<Header Mnemonic> [<Argument>[,<Argument>]...]
```

such as:

```
*RST
```

The common commands are defined by IEEE Std 488.2 and are common to all devices which support IEEE Std 488.2 on the GPIB bus.

Common Query Header. A command that precedes its header mnemonic with an asterisk (*) and follows it with a question mark (?). Its message format is:

```
<Header Mnemonic>? [<Argument>[,<Argument>]...]
```

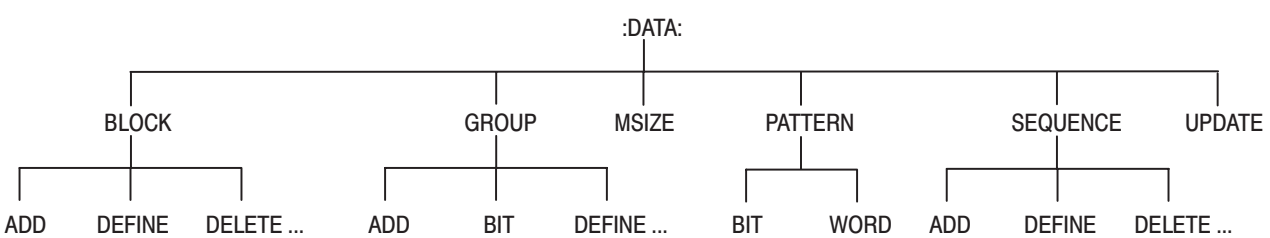
such as:

```
*IDN?
```

The common commands are defined by IEEE Std 488.2 and are common to all devices which support the IEEE Std 488.2 on the GPIB bus.

Concatenating Commands

Most of the compound command headers are in a tree structure. The tree structure of an example command is diagrammed below. Note that the top of the structure always begins with a colon (:).



The following example of a compound command combines three headers delimited by semicolons:

```
:DATA:BLOCK:ADD 512,"BLOCK3"; :DATA:BLOCK:DELETE "BLOCK2";
:DATA:BLOCK:SIZE "BLOCK1",512
```

You must include the complete path in each header when there is no common complete path to the start of the tree structure (the colon). However, note that part of each header in the above example has a common path `:DATA:BLOCK`. You

may shorten compound command structures with such headers. For example, the command above may be rewritten as follows.

```
:DATA:BLOCK:ADD 512,"BLOCK3"; DELETE "BLOCK2"; SIZE
"BLOCK1",512
```

Note that the mnemonics :DATA and :BLOCK are assumed from the first header by the headers that follow. The following command descriptions are valid examples of commands shortened using the principle just described. (Note that the insertion of common command (*SRE) between headers does not prevent the headers that follow from assuming the earlier header mnemonics.)

```
:DATA:BLOCK:ADD 512,"BLOCK3"; DELETE "BLOCK2";
:DATA:GROUP:DELETE "GROUP4"
```

```
:DATA:MSIZE 16384; BLOCK:ADD 512,"BLOCK3"; DELETE "BLOCK2"
```

```
:DATA:BLOCK:ADD 512,"BLOCK3"; *SRE?; DELETE "BLOCK2"; SIZE
"BLOCK1",512
```

The following examples have been shortened incorrectly and cause errors.

```
:DATA:BLOCK:DELETE "BLOCK2"; DATA:GROUP:DELETE "GROUP4"
```

```
:DATA:BLOCK:ADD 512,"BLOCK3"; GROUP:DELETE "GROUP4"
```

```
:DATA:BLOCK:DELETE "BLOCK2"; MSIZE 16384
```

Query Responses

The query causes the data generator to return information about its status or settings. A few queries also initiate an operation action before returning information; for instance, the *TST? query performs the self test.

If the programmer has enabled headers to be returned with query responses, the data generator formats a query response like the equivalent set-command header followed by its argument(s). When headers are turned off for query responses, only the values are returned. Table 2-3 shows the difference in query responses.

Table 2-3: Header in query responses

Query	Header on	Header off
DATA:MSIZE?	:DATA:MSIZE 16384	16384
DIAGNOSTIC:SELECT?	:DIAGNOSTIC:SELECT PMEMORY	PMEMORY

Use the command HEADER ON when you want the header returned along with the information. You can save such a response and send it back as a set-command later. Use HEADER OFF when you want only the information back.

Other General Command Conventions

Upper and Lower Case

The instrument accepts upper, lower, or mixed case alphabetic messages. The following three commands are recognized as identical.

```
HEADER ON
```
or
```
header on
```
or
```
header On
```

Abbreviation

Any header, argument, or reserved word that is sent to the data generator can be abbreviated. The minimum required spelling is shown in upper case throughout the subsection *Command Groups* beginning on page 2-11. The command TRIGger:SLOPe POSitive can be rewritten in either of the following forms.

```
TRIGGER:SLOPE POSITIVE
```
or
```
TRIG:SLOP POS
```

Command Groups

This subsection describes the organization of the DG2020A Data Generator command as a number of functional groups. (See subsection *Command Descriptions* on page 2-19 for a complete description of each command in alphabetical order.)

Throughout this section, the parenthesized question symbol (?) follows the command header to indicate that both a command and query form of the command can be used.

Command Summaries

Tables 2-4 through 2-14 describe each command in each of the 11 functional groups.

DATA Commands

The DATA commands are used to define blocks, groups, and sequences, to set up pattern data, and to set which sequence controls become valid when the run mode is set to Enhanced.

Table 2-4: DATA commands

Header	Description
DATA?	Query the settings related to pattern data
DATA:BLOCk:ADD	Add a block definition
DATA:BLOCk:DEFine(?)	Set the block definitions
DATA:BLOCk:DELete	Delete a block definition
DATA:BLOCk:DELete:ALL	Delete all block definitions
DATA:BLOCk:REName	Change a block name
DATA:BLOCk:SIZe(?)	Change the size of a block
DATA:GROUp:ADD	Add a group definition
DATA:GROUp:BIT(?)	Change a groups bit structure
DATA:GROUp:DEFine(?)	Sets the group definitions
DATA:GROUp:DELete	Delete a group definition
DATA:GROUp:DELete:ALL	Delete all group definitions
DATA:GROUp:NAME?	Query the name of a group
DATA:GROUp:REName	Change a group name
DATA:MSIZe(?)	Set the pattern data memory size

Table 2-4: DATA commands (Cont.)

Header	Description
DATA:PATTern:BIT(?)	Set individual pattern data bits
DATA:PATTern[:WORD](?)	Set pattern data in word units
DATA:SEQuence:ADD	Add a sequence step
DATA:SEQuence:DEFine(?)	Set the sequence definitions
DATA:SEQuence:DELete	Delete a sequence step
DATA:SEQuence:DELete:ALL	Delete all sequence definitions
DATA:SEQuence:EVJ(?)	Set the event jump on/off state
DATA:SEQuence:EVJTO(?)	Set the event jump destination
DATA:SEQuence:LOOP(?)	Set the infinite loop on/off state
DATA:SEQuence:REPeat(?)	Set the repeat count
DATA:SEQuence:TWAIT(?)	Set the trigger wait on/off state
DATA:SUBSequence:ADD	Add a sub sequence step
DATA:SEBSequence:CLEAr	Delete all sub sequence definitions
DATA:SUBSequence:DEFine(?)	Set or query the sub sequence definitions
DATA:SUBSequence:DELete	Delete a sub sequence step
DATA:SUBSequence:DELete:ALL	Delete a sub sequence definition
DATA:SUBSequence:REPeat(?)	Set or query the repeat count of a sub sequence step
DATA:UPDate	Forcibly update the pattern and other data

DIAGNOSTIC Commands

The DIAGNOSTIC commands select and execute the self–test routines, which are classified by function.

Table 2-5: DIAGNOSTIC commands

Header	Description
DIAGnostic?	Query all current settings related to self test
DIAGnostic:RESUlt?	Query self-test result
DIAGnostic:SELect(?)	Select self-test routine
DIAGnostic:STATe	Perform self test
*TST?	Perform self test

DISPLAY Commands

The DISPLAY commands execute functions associated with front panel keys, buttons, and knobs, adjust the screen brightness, and perform other display related functions.

Table 2-6: DISPLAY commands

Header	Description
ABSTouch	Perform the function corresponding to the front–panel control selected
DISPlay?	Query settings made with display group commands
DISPlay:BRIGhtness(?)	Set brightness of screen
DISPlay:CLOCk(?)	Set the date and time display state
DISPlay:DIMmer(?)	Set the state of the display dimmer function
DISPlay:ENABle(?)	Set the display on/off state
DISPlay:MENU?	Query all menu display related information
DISPlay:MENU[:NAME]	Set the menu selection state
DISPlay:MENU:NAME?	Query the menu selection state
DISPlay:MENU:STATe(?)	Set the menu display on/off state
DISPlay[:WINDow]:TEXT:CLEar	Erase the message display area
DISPlay[:WINDow]: TEXT[:DATA](?)	Set the contents of the message display area

HARDCOPY Commands

The HARDCOPY commands start and stop for hardcopy operation, and select port and its outputting format.

Table 2-7: HARDCOPY commands

Header	Description
HCOPy?	Query all hardcopy related information
HCOPy:ABORt	Stop the current hardcopy operation
HCOPy:DATA?	Create and send hardcopy data
HCOPy:FORMat(?)	Select output format of hardcopy
HCOPy:PORT(?)	Select output port of hardcopy
HCOPy:STARt	Start a hardcopy operation

MEMORY Commands

The MEMORY commands control all floppy disk and file operations.

Table 2-8: MEMORY commands

Header	Description
MMEMory:CATalog[:ALL]?	Query disk file and directory information
MMEMory:CATalog:ORDer(?)	Set the display order for disk file and directory information
MMEMory:CDIRectory(?)	Set the current working directory
MMEMory:COPY	Copy a disk file
MMEMory:DELete:ALL	Delete all files and directories on a disk
MMEMory:DELete[:NAME]	Delete the specified file or directory on a disk
MMEMory:FREE?	Query disk memory usage
MMEMory:INITialize	Format a disk
MMEMory:LOAD	Read data from a disk file
MMEMory:LOCK(?)	Set the disk file lock state
MMEMory:MDIRectory	Create a new directory on the disk
MMEMory:RDIRectory	Remove a directory on the disk
MMEMory:RENname	Change the name of a disk file or directory
MMEMory:SAVE	Write data to a disk file

MODE Commands

The MODE commands are used to set the run and update modes, to start or stop pattern data or sequence output, and to set the trigger conditions for the external trigger source.

Table 2-9: MODE commands

Header	Description
MODE?	Query the states related to pattern generation
MODE:STATe(?)	Set the run mode for pattern generation
MODE:UPDate(?)	Set the data update mode
RUNNing?	Query whether the instrument is currently outputting a pattern or sequence
STARt	Start pattern or sequence output
STOP	Stop pattern or sequence output
*TRG	Generate the triggering event
TRIGger?	Query all current trigger-related settings

Table 2-9: MODE commands (Cont.)

Header	Description
TRIGger:IMPedance(?)	Select the impedance presented to the the external trigger signal
TRIGger:LEVel(?)	Set the level of the external trigger signal that generates the triggering event
TRIGger:SLOPe(?)	Select the slope of the external signal that generates a triggering event

OUTPUT Commands

The OUTPUT commands set all the pod–related settings. The <s> and <n> terms in the header mnemonic are used to specify the pod and channel in these commands.

Table 2-10: OUTPUT commands

Header	Description
OUTPut?	Query the settings related to the output channels and clock
OUTPut:ELEVel(?)	Set the event input level
OUTPut:ILEVel(?)	Set the inhibit input level
OUTPut:POD<s>:CH<n>:ASSIGn(?)	Set the pod data bit assignments
OUTPut:POD<s>:CH<n>:DELAy(?)	Set the pod delay times
OUTPut:POD<s>:CH<n>:HIGH(?)	Set the pod high–level output voltage
OUTPut:POD<s>:CH<n>:INHibit(?)	Set the pod high–impedance control method
OUTPut:POD<s>:CH<n>:LOW(?)	Set the pod low–level output voltage
OUTPut:POD<s>:CH<n>:RELEase	Clear the pod data bit assignments
OUTPut:POD<s>:DEFine(?)	Set pod data bit assignments
OUTPut:POD<s>:TYPE?	Query the pod type

SOURCE Commands

The SOURCE commands are used to select the clock signal source, set the clock frequency, and enable or disable the event input of the pod.

Table 2-11: SOURCE commands

Header	Description
SOURce[:OSCillator]?	Query all clock signal settings
SOURce:OSCillator:EXTernal:FREQuency(?)	Input the external clock frequency
SOURce:OSCillator[:INTernal]:FREQuency(?)	Set the internal clock frequency
SOURce:OSCillator[:INTernal]:PLLlock(?)	Set the internal clock oscillator circuit PLL operating state
SOURce:OSCillator:SOURce(?)	Set the clock signal internal/external selection
SOURce:POD<s>:EVENT:ENABle(?)	Enable or disable the event input of the pod

STATUS & EVENT Commands

The STATUS & EVENT commands are used to set and query the registers and queues used by the status and event reporting system, to investigate the state of the instrument, and to control event generation. See section 3 for details on the status and event reporting system.

Table 2-12: STATUS & EVENT commands

Header	Description
ALLEv?	Dequeue all events from Event Queue
*CLS	Clear SESR, SBR and Event Queue
DESE(?)	Set and query DESER
*ESE(?)	Set and query ESER
*ESR?	Query SESR
EVENT?	Dequeue event from Event Queue
EVMsg?	Dequeue event from Event Queue
EVQty?	Query number of event on Event Queue
*PSC(?)	Set power-on status clear flag
*SRE(?)	Set and query SRER
*STB?	Query SBR

SYNCHRONIZATION Commands

The SYNCHRONIZATION commands monitor for the completion of all pending operations.

Table 2-13: SYNCHRONIZATION commands

Header	Description
*OPC(?)	Generate or return the operation complete message
*WAI	Hold off all commands until all pending operations complete

SYSTEM Commands

The SYSTEM commands are used (for example) to set the date and time, to lock out front panel control, to control the handling of headers in responses, and to query for ID and setting information. This group is a collection of commands that cannot be classified in any other group.

Table 2-14: SYSTEM commands

Header	Description
DEBug?	Query all settings for debugging
DEBug:SNOop?	Query all settings for debugging
DEBug:SNOop:DELAy?	Query delay time for debugging
DEBug:SNOop:DELAy:TIME(?)	Set delay time for debugging
DEBug:SNOop:STATe(?)	Turn on or off for debugging
FACTory	Reset all settings to defaults
HEADer(?)	Allow or suppress the return of the control header in response messages
ID?	Query ID information about the data generator
*IDN?	Query ID information about the data generator
LOCk(?)	Lock or unlock local control using the front-panel controls
*OPT?	Query which options are implemented for this data generator
*RST	Reset this data generator
SYSTem:DATE(?)	Set the clock date
SYSTem:PPAUse(?)	Set the setting for system operation when a self–diagnostics detects an error
SYSTem:SECurity:IMMediate	Delete all settings and data
SYSTem:SECurity:STATe(?)	Set the security on/off state
SYSTem:TIME(?)	Set the clock time

Table 2-14: SYSTEM commands (Cont.)

Header	Description
UNLock	Unlock (allow) local control using the front-panel controls
UPTime?	Query the elapsed time since power on
VERBose(?)	Select short or long response headers

Command Descriptions

This subsection lists each command and query in the command set alphabetically. Each command entry includes its command description and command group, its related commands (if any), its syntax, and its arguments. Each entry also includes one or more usage examples.

This subsection fully spells out headers, mnemonics, and arguments with the minimal spelling shown in upper case. For example, to use the abbreviated version of the DISPlay:BRIGhtness command, just type DISP:BRIG.

The symbol '(?)' follows the command header of those commands that can be used as either a command or a query. The symbol '?' follows those commands that can only be a query. If neither symbol follows the command, it can only be used as a command.

ABSTouch

The ABSTouch command performs the same action that actuating the corresponding front-panel key, button, or knob would do.

Group DISPLAY

Related Commands

Syntax

```
ABSTouch {BOTTOM1 | BOTTOM2 | BOTTOM3 | BOTTOM4 | BOTTOM5 |
BOTTOM6 | BOTTOM7 | SIDE1 | SIDE2 | SIDE3 | SIDE4 | SIDE5 |
CLEARMenu | SETUp | EDIT | APPLication | UTILity | CURSor |
EXECute | UParrow | DOWNarrow | LEFTarrow | RIGHtarrow | KNOBLeft
| KNOBRight | RUN | STEp | ZERo | ONE | TWO | THREe | FOUR | FIVe
| SIX | SEVen | EIGHt | NINe | POINt | A | MINUs | B | HZ | S | V
| C | KHZ | MS | MV | D | MHZ | US | E | NS | F | DELete | ENTer
| HARDcopy | MANual}
```

Arguments Sending any of the arguments that are shown in Figure 2-2 is the equivalent of operating a front panel control. Which argument corresponds to which control is shown by in Figure 2-2. Sending an argument corresponding to a front-panel button is the same as pressing that button once; if the argument sent corresponds to a knob, it is the same as rotating the knob clockwise or counterclockwise by $\frac{1}{25}$ of a turn.

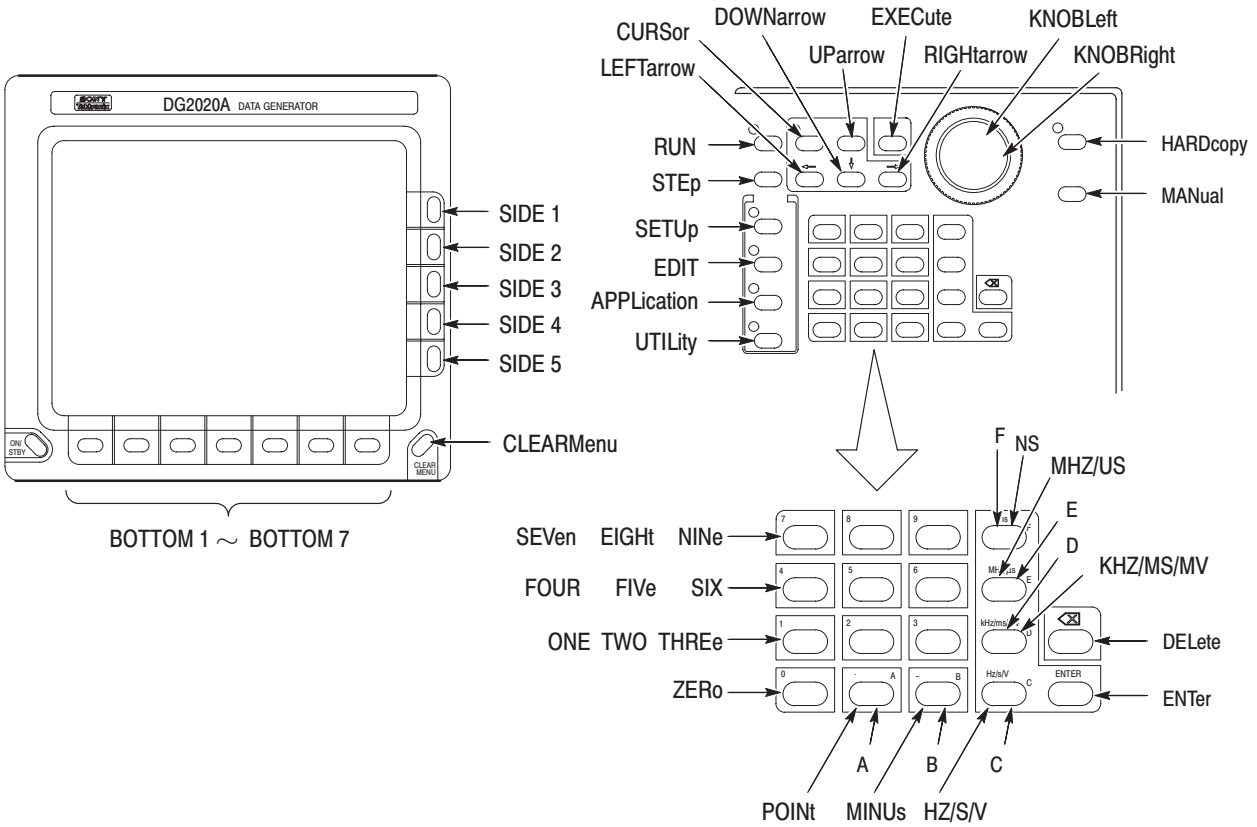


Figure 2-2: ABSTouch arguments and associated controls

Examples

```
ABSTOUCH SETUP
```

displays the same setup menu that is displayed by pressing the front-panel button SETUP in the MENU column on the front panel.

ALLEv?

The ALLEv? query dequeues all event codes and their corresponding event messages. Use the *ESR? query to make events available for dequeuing using ALLEv? query.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR?, EVENT?, EVMsg?, EVQty?, *SRE, *STB?

Syntax

```
ALLEv?
```

Arguments None

Responses `[:ALLEV]<event code>,"<event message;second message>" [;<event code>,"<event message:second message>"]...`

Examples `ALLEV?`
might return the string
`:ALLEV 113,"Undefined header; unrecognized command - OUT:ELEV"; 420, "Query UNTERMINATED".`

*CLS

The *CLS common command clears SESR (Standard Event Status Register), the SBR (Status Byte Register) and the Event Queue, which are used in the data generator status and event reporting system. For more details, refer to Section 3 *Status and Events*.

Group STATUS & EVENT

Related Commands DESE, *ESE, *ESR?, *EVENT?, EVMsg?, EVQty?, *SRE, *STB?

Syntax *CLS

Examples *CLS
clears the SESR, the SBR, and the Event Queue.

DATA?

The DATA? query returns the setting states related to the pattern data.

Group DATA

Related Commands OUTPut?

Syntax DATA?

Examples

```
DATA?
```

might return

```
:DATA:MSIZE378;BLOCK:DEFINE #2440,BLOCK_1<LF>99,BLOCK_2<LF>189,
BLOCK_3<LF>288,BLOCK_4;:DATA:SUBSEQUENCE:DEFINE #217UNNAMED,1;
:DATA:SEQUENCE:DEFINE #271BLOCK_1,1,0,1,0,0<LF>BLOCK_2,1,0,0,1,0
<LF>BLOCK_3,1,0,0,0,0<LF>BLOCK_4,1,0,0,0;:DATA:GROUP:DEFINE #279
DATA7,7,7<LF>DATA6,6,6<LF>DATA5,5,5<LF>DATA4,4,4<LF>DATA3,3,3<LF>
DATA2,2,2<LF>DATA1,1,1<LF>DATA0,0,0
```

DATA:BLOCk:ADD

The DATA:BLOCk:ADD command adds a block. This results in one new block being defined in the block definition section.

Group DATA

Related Commands DATA:BLOCk:DEFine, DATA:BLOCk:DELete, DATA:BLOCk:DELete:ALL, DATA:BLOCk:REName, DATA:BLOCk:SIZe

Syntax

```
DATA:BLOCk:ADD <Position>,<Name>
```

Arguments

<Position>::=<NR1>
where <NR1> is the start position of the added block.

<Name>::=<string>
where <string> is the name of the added block.

Examples

```
:DATA:BLOCK:ADD 512,"BLOCK1"
```

adds a block starting at address 512 named BLOCK1.

DATA:BLOCk:DEFine (?)

The DATA:BLOCk:DEFine command sets up the information for the whole block definition section in ASCII. The DATA:BLOCk:DEFine? query returns the whole block definition section.

Group DATA

Related Commands DATA:BLOCk:ADD, DATA:BLOCk:DELete, DATA:BLOCk:DELete:ALL, DATA:BLOCk:REName, DATA:BLOCk:SIZe

Syntax

```
DATA:BLOCk:DEFine <Blockinfo>
DATA:BLOCk:DEFine?
```

Arguments

<Blockinfo>::=<blockheader><Blkdef>[<LF><Blkdef>][<LF><Blkdef>]...
Arbitrary block data for the block definition

where,

<blockheader>::=<byte count digit><byte count>

<Blkdef>::=<APosition>,<AName>

<Aposition> is the block starting position specified in ASCII (Note that the starting position of the first block must be zero), and <AName> is the block name specified in ASCII.

<LF>::=<ASCII line feed code (dec 10)>

Responses

[:DATA:BLOCk:DEFINE] <Blockinfo>
where <Blockinfo> is a data block in the same format as the argument.

Examples

:DATA:BLOCk:DEFine #2320,BLOCK0<LF>512,BLOCK1<LF>1024,BLOCK2
defines three blocks: BLOCK0, BLOCK1, and BLOCK2.

DATA:BLOCk:DELete

The DATA:BLOCk:DELete command deletes the specified block. Note that the first block cannot be deleted.

Group DATA

Related Commands DATA:BLOCk:ADD, DATA:BLOCk:DEFine, DATA:BLOCk:DELete:ALL, DATA:BLOCk:REName, DATA:BLOCk:SIZe

Syntax DATA:BLOCk:DELete <Name>

Arguments <Name>::=<string>
where <string> is the name of the block to be deleted.

Examples :DATA:BLOCK:DELETE "BLOCK2"
deletes the block with the name BLOCK2.

DATA:BLOCk:DELete:ALL

The DATA:BLOCk:DELete:ALL command deletes all blocks. After this command is executed, the whole memory area consists of one block with the name ”NO NAME”.

Group DATA

Related Commands DATA:BLOCk:ADD, DATA:BLOCk:DEFine, DATA:BLOCk:DELete, DATA:BLOCk:REName, DATA:BLOCk:SIZe

Syntax DATA:BLOCk:DELete:ALL

Arguments None

DATA:BLOCk:REName

The DATA:BLOCk:REName command changes the name of a data block.

Group DATA

Related Commands DATA:BLOCk:ADD, DATA:BLOCk:DEFine, DATA:BLOCk:DELete, DATA:BLOCk:DELete:ALL, DATA:BLOCk:SIZe

Syntax DATA:BLOCk:REName <From-blockname>,<To-blockname>

Arguments <From-blockname>::=<string>
where <string> is the name of the block before it is renamed.

<To-blockname>::=<string>
where <string> is the name of the block after it is renamed.

Examples :DATA:BLOCK:RENAME "BLOCK3","BLOCK4"
changes the name of BLOCK3 to BLOCK4.

DATA:BLOCk:SIZe (?)

The DATA:BLOCk:SIZe command changes the size of a data block. The DATA:BLOCk:SIZe? query returns the size of the specified block.

Group DATA

Related Commands DATA:BLOCk:ADD, DATA:BLOCk:DEFine, DATA:BLOCk:DELete, DATA:BLOCk:DELete:ALL, DATA:BLOCk:REName

Syntax DATA:BLOCk:SIZe <Name>,<Size>
DATA:BLOCk:SIZe? <Name>

Arguments <Name>::=<string>
where <string> is a block name.

<Size>::=<NR1>
where <NR1> is a new block size.

Responses [:DATA:BLOCK:SIZE] <Name>,<Size>

Examples

```
:DATA:BLOCK:SIZE "BLOCK1",512
```

changes the block size of the block BLOCK1 to 512.

DATA:GROUp:ADD

The DATA:GROUp:ADD command adds a group.

Group

DATA

Related Commands

DATA:GROUp:BIT, DATA:GROUp:DEFine, DATA:GROUp:DELete, DATA:GROUp:DELete:ALL, DATA:GROUp:NAME?, DATA:GROUp:REName

Syntax

```
DATA:GROUp:ADD <Name>,<MSB>,<LSB>
```

Arguments

<Name>::=<string>
where <string> is the name of the group to be added.

<MSB>::=<NR1>
where MSB is the Most Significant Bit
where <NR1>is the high order bit for the group.

<LSB>::=<NR1>
where LSB is the Least Significant bit
where <NR1>is the low order bit for the group.

Examples

```
:DATA:GROUP:ADD "GROUP01",3,0
```

adds a group that consists of 4 bits, DATA00 to DATA03, and has the name GROUP01.

DATA:GROUp:BIT (?)

The DATA:GROUp:BIT command changes the bit configuration of a group. The DATA:GROUp:BIT? query returns the set bit configuration.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:DEFine, DATA:GROUp:DELete, DATA:GROUp:DELete:ALL, DATA:GROUp:NAME?, DATA:GROUp:REName

Syntax

```
DATA:GROUp:BIT <Name>,<MSB>,<LSB>
DATA:GROUp:BIT? <Name>
```

Arguments

<Name>::=<string>
where the name of the group to be changed or queried.

<MSB>::=<NR1>
where <NR1>is the high order bit for the group.

<LSB>::=<NR1>
where <NR1>is the low order bit for the group.

Responses [:DATA:GROUP:BIT] <Name>,<MSB>,<LSB>

Examples

```
:DATA:GROUP:BIT "GROUP02",7,4
```

changes the bit configuration for the group named GROUP02 to be DATA04 to DATA07.

DATA:GROUp:DEFine (?)

The DATA:GROUp:DEFine command sets up the information for the whole group definition section in ASCII. The DATA:GROUp:DEFine? query returns the information for the whole group definition section.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:BIT, DATA:GROUp:DELete, DATA:GROUp:DELete:ALL, DATA:GROUp:NAME?, DATA:GROUp:REName

Syntax

```
DATA:GROUp:DEFine <Groupblock>
DATA:GROUp:DEFine?
```

Arguments

<Groupblock>::=<blockheader><Group>[<LF><Group>][<LF><Group>]...
Arbitrary block data for the group definition

where,

<blockheader>::=<byte count digit><byte count>

<Group>::=<AName>,<AMSB>,<ALSB>

The <AName>, <AMSB>, and <ALSB> fields are ASCII character strings that specify the following information.

<AName>	group name
<AMSB>	group's high order bit
<ALSB>	group's low order bit

<LF>::=<ASCII line feed code (10)>

Responses

[:DATA:GROUP:DEFINE] <Groupblock>
where <Groupblock> is a data block with the same format as the argument.

Examples

```
:DATA:GROUp:DEFine
#238GROUP01,7,0<LF>GROUP02,11,8<LF>GROUP03,15,12
```

defines the three groups GROUP01, GROUP02, and GROUP03.

DATA:GROUp:DELete

The DATA:GROUp:DELete command deletes the specified group.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:BIT, DATA:GROUp:DEFine, DATA:GROUp:DELete:ALL, DATA:GROUp:NAME?, DATA:GROUp:REName

Syntax DATA:GROUp:DELete <Name>

Arguments <Name>::=<string>
where <string> is the name of the group to delete.

Examples :DATA:GROUP:DELETE "GROUP02"
deletes the group with the name GROUP02.

DATA:GROUp:DELete:ALL

The DATA:GROUp:DELete:ALL command deletes all group definitions.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:BIT, DATA:GROUp:DEFine, DATA:GROUp:DELete, DATA:GROUp:NAME?, DATA:GROUp:REName

Syntax DATA:GROUp:DELete:ALL

Arguments None

DATA:GROUp:NAME?

The DATA:GROUp:NAME? query returns the name of the group that includes the specified bit.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:BIT, DATA:GROUp:DEFine, DATA:GROUp:DELete, DATA:GROUp:DELete:ALL, DATA:GROUp:REName

Syntax DATA:GROUp:NAME? <Bit>

Arguments <Bit>::=<NR1>
where <NR1> is the number of the bit to be queried (0 to 35).

Responses [:DATA:GROUP:NAME] <Bit>,<Name>
where

<Bit>::=<NR1>	a bit number (0 to 35)
<Name>::=<string>	the group name

Examples DATA:GROUp:NAME? <6>
might return :DATA:GROUP:NAME 6,"GROUP02", which indicates that the name of the group that includes the DATA06 bit is GROUP02.

DATA:GROUp:REName

The DATA:GROUp:REName command changes the name of a group.

Group DATA

Related Commands DATA:GROUp:ADD, DATA:GROUp:BIT, DATA:GROUp:DEFine, DATA:GROUp:DELete, DATA:GROUp:DELete:ALL, DATA:GROUp:NAME?

Syntax DATA:GROUp:REName <From-groupname>,<To-groupname>

Arguments <From-groupname>::=<string>
where <string> is the name of the group before it is renamed.

<To-groupname>::=<string>
where <string> is the name of the group after it is renamed.

Examples :DATA:GROUP:RENAME "GROUP03","GROUP04"
changes the name of the group GROUP03 to be GROUP04.

DATA:MSIZe (?)

The DATA:MSIZe command sets the bit pattern section memory area size. The DATA:MSIZe? query returns the bit pattern section memory area setting.

Group DATA

Related Commands

Syntax
```
DATA:MSIZe <Memory Size>
DATA:MSIZe?
```

Arguments
<Memory Size>::=<NR1>
where <NR1> is the number that expresses the memory size (in words).

Responses
```
[:DATA:MSIZE] <Memory Size>
```

DATA:PATTern:BIT (?)

The DATA:PATTern:BIT command sets the data memory bit pattern section. Data is given in bit units. The DATA:PATTern:BIT? query returns the contents of the data memory bit pattern section.

Group DATA

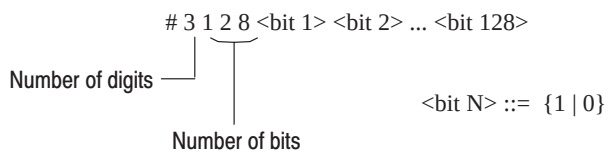
Related Commands
```
DATA:PATTern[:WORD]
```

Syntax
```
DATA:PATTern:BIT <Bit Position>,<Address>,<Length>,<Data>
DATA:PATTern:BIT? <Bit Position>,<Address>,<Length>
```

Arguments

`<Bit Position>::=<NR1>`	bit position (0 to 35)
`<Address>::=<NR1>`	start address (0 to 65535)
`<Length>::=<NR1>`	data length (1 to 65536)
`<Data>::=<block>`	arbitrary block data for the bit pattern section

Example where the data length is 128:



The value of the data bit at the specified address is specified with the ASCII character for 0 or 1. Data bits for the specified data length are stored in address order, with all bits expressed similarly in ASCII. The number of bytes in the block header will be equal to the length of the specified data.

Responses `[:DATA:PATTERN:BIT] <Bit Position>,<Address>,<Length>,<Data>`

DATA:PATTern[:WORD] (?)

The DATA:PATTern[:WORD] command sets the data memory bit pattern section. The data is given in word units. The DATA:PATTern:WORD? query returns the contents of the data memory bit pattern section.

Group DATA

Related Commands DATA:PATTern:BIT

Syntax

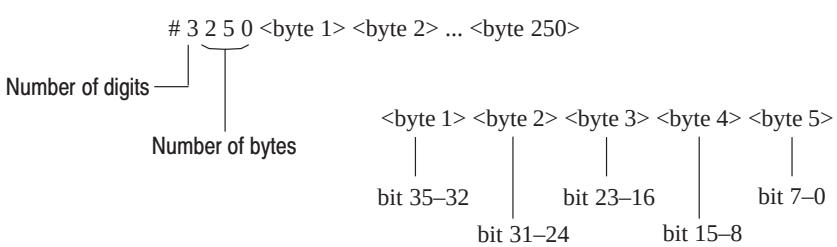
```
DATA:PATTern[:WORD] <Address>,<Length>,<Data>
DATA:PATTern:[WORD]? <Address>,<Length>
```

Arguments

<Address>::=<NR1>
where <NR1> is a start address (0 to 65535)

<Length>::=<NR1>	data length (1 to 65536)
<Data>::=<block>	arbitrary block data for the bit pattern section

Example where the data length is 50:



Each word (36 bits) of the bit pattern data is expressed as a group of 5 bytes starting with the first byte. When each byte group is seen as consisting of the bytes byte1 to byte5 as shown in the figure, the bits correspond to the bits in the bit pattern data starting with the MSB in order starting with byte1. Although all 8 bits in byte2 to byte5 are used, the high–order 4 bits in byte1 are unused. The data block is formed by iterating this packing method for each word in order starting with the start address. Thus the number of bytes in the data block (excluding the header) will be 5 times the number of words.

Responses [:DATA:PATTERN:WORD] <Address>,<Length>,<Data>

DATA:SEQuence:ADD

The DATA:SEQuence:ADD command adds a sequence step.

Group DATA

Related Commands DATA:SEQuence:DEFine, DATA:SEQuence:DELete, DATA:SEQuence:DELete:ALL

Syntax

```
DATA:SEQuence:ADD <LineN>,<Name>,<Repeat>,<To>,<WaitE>,<JumpE>,<LoopE>
```

Arguments

<LineN>::=<NR1>
where <NR1> is a sequence step number.

<Name>::=<string>
where <string> is a block name (surrounded in double (”) or single (’) quotes).

<Repeat>::=<NR1>
where <NR1> is a repeat count (1 to 65536).

<To>::=<NR1>
where <NR1> is a event jump destination line number.

<WaitE>::={ON | OFF | 1 | 0}	trigger wait on/off state
<JumpE>::={ON | OFF | 1 | 0}	event jump on/off state
<LoopE>::={ON | OFF | 1 | 0}	infinite loop on/off

Examples

```
:DATA:SEQUENCE:ADD 4,"BLOCK3",16,0,0,1,1
```

adds a sequence step that consists of the block named BLOCK3 at the sequence line number 4 position.

When this sequence is executed in enhanced mode, BLOCK3 will be executed repeatedly since the infinite loop setting is set on. However, since event jump is also set on, the instrument will jump to sequence line number 0 if an external event occurs. In run modes other than enhanced mode, BLOCK3 will be executed 16 times and then control will switch to the next line number.

DATA:SEQuence:DEFine (?)

The DATA:SEQuence:DEFine command sets up all of the sequence definition section information in ASCII. The DATA:SEQuence:DEFine? query returns all of the sequence definition section information.

Group DATA

Related Commands DATA:SEQuence:ADD, DATA:SEQuence:DELete, DATA:SEQuence:DELete:ALL

Syntax

```
DATA:SEQuence:DEFine <Sequence Block>
DATA:SEQuence:DEFine?
```

Arguments

<Sequence Block>::=<blockheader><Step>[<LF><Step>][<LF><Step>]...
Arbitrary block data for the sequence definition

where,

<blockheader>::=<byte count digit><byte count>

<Step>::=<AName>,<ARepeat>,<ATo>,<AWaitE>,<AJumpE>,<ALoopE>

The items in <Step> are ASCII character strings that express the following information.

<AName>	the block name (with no quotation marks)
<ARepeat>	repeat count (1 to 65536)
<ATo>	event jump destination line number
<AWaitE>	trigger wait on/off state ({ON | 1}:ON, {OFF | 0}:OFF)
<AJumpE>	event jump on/off state ({ON | 1}:ON, {OFF | 0}:OFF)
<ALoopE>	infinite loop on/off ({ON | 1}:ON, {OFF | 0}:OFF)

<LF>::=<ASCII line feed code (10)>

Responses

[:DATA:SEQUENCE:DEFINE] <Sequence Block>
where <Sequence Block> is a data block with the same format as the argument. However, note that rather than the ON and OFF keywords, only 0 and 1 are used for the <AWaitE>, <AJumpE>, and <ALoopE> items.

Examples

```
:DATA:SEQuence:DEFine #235BLOCK1,16,0,1,0,0<LF>BLOCK2,32,0,0,1,1
```

defines a two step sequence that consists of the two blocks BLOCK1 and BLOCK2.

DATA:SEQuence:DELete

The DATA:SEQuence:DELete command deletes the specified sequence step.

Group DATA

Related Commands DATA:SEQuence:ADD, DATA:SEQuence:DEFine, DATA:SEQuence:DELete:ALL

Syntax DATA:SEQuence:DELete <Line Number>

Arguments <Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be deleted.

Examples :DATA:SEQUENCE:DELETE 3
deletes the line 3 sequence step.

DATA:SEQuence:DELete:ALL

The DATA:SEQuence:DELete:ALL command deletes all sequence definitions.

Group DATA

Related Commands DATA:SEQuence:ADD, DATA:SEQuence:DEFine, DATA:SEQuence:DELete

Syntax DATA:SEQuence:DELete:ALL

Arguments None

DATA:SEQuence:EVJ (?)

The DATA:SEQuence:EVJ command sets the sequence step event jump to on or off. The DATA:SEQuence:EVJ? query returns the sequence step event jump on/off state.

Group DATA

Related Commands DATA:SEQuence:EVJTO, DATA:SEQuence:LOOP, DATA:SEQuence:REPeat, DATA:SEQuence:TWAIT

Syntax

```
DATA:SEQuence:EVJ <Line Number>,{ON | OFF | 1 | 0}
DATA:SEQuence:EVJ? <Line Number>
```

Arguments

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

ON or 1
sets the event jump to on.

OFF or 0
sets the event jump to off.

Responses

```
[:DATA:SEQUENCE:EVJ] <Line Number>,{1 | 0}
```

Examples

```
:DATA:SEQUENCE:EVJ 8,ON
```
sets the event jump state for the line 8 sequence step to on.

DATA:SEQuence:EVJTO (?)

The DATA:SEQuence:EVJTO command sets the sequence step event jump destination. The DATA:SEQuence:EVJTO? query returns the event jump destination set for the sequence step.

Group DATA

Related Commands DATA:SEQuence:EVJ, DATA:SEQuence:LOOP, DATA:SEQuence:REPeat, DATA:SEQuence:TWAIT

Syntax

```
DATA:SEQuence:EVJTO <Line Number>,<Target>
DATA:SEQuence:EVJTO? <Line Number>
```

Arguments

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

<Target>::=<NR1>
where <NR1> is the line number of the jump destination sequence step.

Responses

```
[:DATA:SEQUENCE:EVJTO] <Line Number>,<Target>
```

Examples

```
:DATA:SEQUENCE:EVJTO 5,0
```

sets the line 5 sequence step event jump destination to line 0.

DATA:SEQuence:LOOP (?)

The DATA:SEQuence:LOOP command sets the sequence step infinite loop state to on or off. The DATA:SEQuence:LOOP? query returns the sequence step infinite loop on/off state.

Group DATA

Related Commands DATA:SEQuence:EVJ, DATA:SEQuence:EVJTO, DATA:SEQuence:REPeat, DATA:SEQuence:TWAIT

Syntax

```
DATA:SEQuence:LOOP <Line Number>,{ON | OFF | 1 | 0}
DATA:SEQuence:LOOP? <Line Number>
```

Arguments

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

ON or 1
sets the infinite loop state to on.

OFF or 0
sets the infinite loop state to off.

Responses

```
[:DATA:SEQUENCE:LOOP] <Line Number>,{1 | 0}
```

Examples

```
:DATA:SEQUENCE:LOOP 9,OFF
```
sets the infinite loop state for the line 9 sequence step to off.

DATA:SEQuence:REPeat (?)

The DATA:SEQuence:REPeat command sets the sequence step repeat count. The DATA:SEQuence:REPeat? query returns the repeat count set for the sequence step.

Group DATA

Related Commands DATA:SEQuence:EVJ, DATA:SEQuence:EVJTO, DATA:SEQuence:LOOP, DATA:SEQuence:TWAIT

Syntax

```
DATA:SEQuence:REPeat <Line Number>,<Times>
DATA:SEQuence:REPeat? <Line Number>
```

Arguments

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

<Times>::=<NR1>
where <NR1> is a repeat count (1 to 65536).

Responses

```
[:DATA:SEQUENCE:REPEAT] <Line Number>,<Times>
```

Examples

```
:DATA:SEQUENCE:REPEAT 5,8
```
sets the line 5 sequence step repeat count to 8.

DATA:SEQuence:TWAIT (?)

The DATA:SEQuence:TWAIT command sets the sequence step trigger wait state to on or off. The DATA:SEQuence:TWAIT? query returns the sequence step trigger wait on/off state.

Group DATA

Related Commands DATA:SEQuence:EVJ, DATA:SEQuence:EVJTO, DATA:SEQuence:LOOP, DATA:SEQuence:REPeat

Syntax

```
DATA:SEQuence:TWAIT <Line Number>,{ON | OFF | 1 | 0}
DATA:SEQuence:TWAIT? <Line Number>
```

Arguments

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

ON or 1
sets the trigger wait state to on.

OFF or 0
sets the trigger wait state to off.

Responses

```
[:DATA:SEQUENCE:TWAIT] <Line Number>,{1 | 0}
```

Examples

```
:DATA:SEQUENCE:TWAIT 5,ON
```
sets the line 5 sequence step trigger wait state to on.

DATA:SUBSequence:ADD

The DATA:SUBSequence:ADD command adds a sub sequence step.

Group DATA

Related Commands DATA:SUBSequence:DEFine, DATA:SUBSequence:DELete, DATA:SUBSequence:DELete:ALL

Syntax DATA:SUBSequence:ADD <Sname>, <LineN>, <Name>, <Repeat>

Arguments <Sname>::=<String>
where <string> is a sub sequence name (surrounded in double (”) or single (’) quotes).

<LineN>::=<NR1>
where <NR1> is a sub sequence step number.

<Name>::=<String>
where <string> is a block name (surrounded in double (”) or single (’) quotes).

<Repeat>::=<NR1>
where <NR1> is a repeat count (1 to 65536).

Examples :DATA:SUBSEQUENCE:ADD "SUB1",2"BLOCK3",10
adds a sub sequence step that consists of the block named BLOCK3 at the sub sequence line number 2 position in the sub sequence named SUB1.

DATA:SUBSequence:CLEAr

The DATA:SUBSequence:CLEAr command clears all sub sequence definitions.

Group DATA

Related Commands DATA:SUBSequence:ADD, DATA:SUBSequence:DEFine, DATA:SUBSequence:DELete, DATA:SUBSequence:DELete:ALL

Syntax DATA:SUBSequence:DELete:CLEAr

Arguments None

DATA:SUBSequence:DEFine (?)

The DATA:SUBSequence:DEFine command sets up all of the sub sequence definition section information in ASCII. The DATA:SUBSquence:DEFine? query returns all of the sub sequence definition section information.

Group DATA

Related Commands DATA:SUBSequence:ADD, DATA:SUBSequence:CLEAr, DATA:SUBSequence:DELete, DATA:SUBSequence:DELete:ALL

Syntax

```
DATA:SUBSequence:DEFine <Subseq Block>
DATA:SUBSequence:DEFine?
```

Arguments

<Subseq Block>::=<blockhead-er><SName>,<Step>[,<Step>...][<LF><SName>,<Step>][,<Step>...]...]
Arbitrary block data for the sequence definition

where,

<Step>::=<AName>,<ARepeat>

The items in <Step> are ASCII character strings that express the following information.

<SName>	the sub sequence name (with no quotation marks)
<AName>	the block name (with no quotation marks)
<ARepeat>	repeat count (1 to 65536)

<LF>::=<ASCII line feed code (10)>

Responses [:DATA:SUBSEQUENCE:DEFINE] <Subseq Block>
where <Subseq Block> is a data block with the same format as the argument.

Examples :DATA:SUBSEQUENCE:DEFINE #233SUB1,B1,16,B2,32<LF>SUB2,B3,2,B4,3
defines a two step sub sequence that consists of the two sub sequences SUB1 and SUB2.

DATA:SUBSequence:DELete

The DATA:SUBSequence:DELete command deletes the specified sub sequence step.

Group DATA

Related Commands DATA:SUBSequence:ADD, DATA:SUBSequence:CLEAr, DATA:SUBSequence:DEFine, DATA:SUBSequence:DELete:ALL

Syntax DATA:SUBSequence:DELete <SName>,<Line Number>

Arguments <SName>::=<String>
where <String> is a sub sequence name (surrounded in double (”) or single (’) quotes).

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be deleted.

Examples :DATA:SUBSEQUENCE:DELETE "SUB2", 7
deletes the line 7 sub sequence step named SUB2.

DATA:SUBSequence:DELete:ALL

The DATA:SUBSequence:DELete:ALL command deletes the specified sub sequence definitions.

Group DATA

Related Commands DATA:SUBSequence:ADD, DATA:SUBSequence:CLEAr, DATA:SUBSequence:DEFine, DATA:SUBSequence:DELete

Syntax DATA:SUBSequence:DELete:ALL <SName>

Arguments <SName>::=<String>
where <String> is a sub sequence name (surrounded in double (”) or single (’) quotes).

Examples

```
:DATA:SUBSEQUENCE:DELETE:ALL "SUB1"
```

deletes the sub sequence definition named SUB1.

DATA:SUBSequence:REPeat (?)

The DATA:SUBSequence:REPeat command sets the sub sequence step repeat count. The DATA:SUBSequence:REPeat? query returns the repeat count set for the sub sequence step.

Group DATA

Related Commands None

Syntax

```
DATA:SUBSequence:REPeat <SName>,<Line Number>,<Times>
DATA:SUBSequence:REPeat? <SName>,<Line Number>
```

Arguments

<SName>::=<String>
where <String> is a sub sequence name (surrounded in double (”) or single (’) quotes).

<Line Number>::=<NR1>
where <NR1> is the line number of the sequence step to be set.

<Times>::=<NR1>
where <NR1> is a repeat count (1 to 65536).

Responses

```
[:DATA:SUBSEQUENCE:REPEAT] <SName>,<Line Number>,<Times>
```

Examples

```
:DATA:SUBSEQUENCE:REPEAT "SUB1",5,8
```

sets the line 5 sequence step repeat count in the sub sequence named SUB1 to 8.

DATA:UPDate

The DATA:UPDate command transfers the contents of data memory to pattern generation memory so that the output reflects the most recent data. This command is only valid when the mode is set to manual mode. The processing performed by this command is executed automatically if any data changes when the instrument is in automatic mode.

Group DATA

Related Commands

Syntax DATA:UPDate

Arguments None

DEBug?

The DEBug? query returns all current settings for the remote command debugging function.

This query is equivalent to the DEBug:SNOop? query.

Group SYSTEM

Related Commands DEBug:SNOop?, DEBug:SNOop:DELAy:TIME, DEBug:SNOop:STATe

Syntax DEBug?

Arguments None

Responses See Examples

Examples DEBUG?
might return :DEBUG:SNOOP:STATE 0; DELAY:TIME 0.2

DEBug:SNOop?

The DEBug:SNOop? query returns all current settings for the remote command debugging function.

This query is equivalent to the DEBug? query.

Group SYSTEM

Related Commands DEBug?, DEBug:SNOop:DELAy:TIME, DEBug:SNOop:STATe

Syntax DEBug:SNOop?

Arguments None

Responses See Examples

Examples DEBUG:SNOOP?
might return :DEBUG:SNOOP:STATE 0; DELAY:TIME 0.2

DEBug:SNOop:DELAy?

The DEBug:SNOop:DELAy? query returns the display time for commands in a sequence of commands connected by semicolons.

This query is equivalent to the DEBug:SNOop:DELAy:TIME? query.

Group SYSTEM

Related Commands DEBug?, DEBug:SNOop?, DEBug:SNOop:DELAy:TIME?, DEBug:SNOop:STATe

Syntax DEBug:SNOop:DELAy?

Arguments None

Responses [:DEBUG:SNOOP:DELAY]<Delay Time>
where <Delay Time>::=<NR2>

Examples DEBUG:SNOOP:DELAY?
might return :DEBUG:SNOOP:DELAY:TIME 0.2

DEBug:SNOop:DELAy:TIME (?)

The DEBug:SNOop:DELAy:TIME command sets the display time for commands in a sequence of commands that are connected by semicolons.

The DEBug:SNOop:DELAy:TIME? query returns the display time for commands in a sequence of commands connected by semicolons.

Group SYSTEM

Related Commands DEBug?, DEBug:SNOop?, DEBug:SNOop:DELAy?, DEBug:SNOop:STATe

Syntax

```
DEBug:SNOop:DELAy:TIME <Time>
DEBug:SNOop:DELAy:TIME?
```

Arguments

<Time>::=<NR2>[<unit>]
where <NR2> combined with [<unit>] specifies a time in the range 0.0 s to 10.0 s in steps of 0.1 s, and [<unit>]::={s|ms|µs}, for seconds, milliseconds, or microseconds.

Examples

```
:DEBUG:SNOOP:DELAY:TIME 0.5
```

sets the command display time to 0.5 seconds.

DEBug:SNOop:STATe (?)

The DEBug:SNOop:STATe command sets and clears the remote command debugging function.

The DEBug:SNOop:STATe? query returns the currently specified state of the remote command debugging function.

The debugging function displays messages input from the remote interface in the CRT screen message area. If commands are connected by semicolons, each message is displayed for the time specified with the DEBug:SNOop:DELAy:TIME command.

The display format is as follows.

Control codes — "<code decimal display>", e.g. LF is displayed as "<10>".

Alphanumerics and symbols — "<code ASCII display>", e.g., "A" is displayed as "A".

Message termination — "<PMT>"

Interface messages — "<DCL>" and "<GET>". Others are displayed as "<code decimal display>".

Block data — "#0"

Any data other than one of the above — "<code decimal display>", e.g. a code value of 80 (hexadecimal) would be displayed as <128>.

Group SYSTEM

Related Commands DEBug?, DEBug:SNOop?, DEBug:SNOop:DELAy?, DEBug:SNOop:TIME

Syntax

```
DEBug:SNOop:STATe {ON | OFF | <NR1>}
DEBug:SNOop:STATe?
```

Arguments

ON or nonzero value
enables the debugging function.

OFF or zero value
clears the debugging function.

Responses

1 the debugging function is currently set.
0 the debugging function is currently cleared.

Examples

`:DEBUG:SNOOP:STATE ON`
enables the debugging function.

DESE (?)

The DESE command sets the bits of the DESER (Device Event Status Enable Register) used in the status and event reporting system of the data generator. The DESE? query returns the contents of the DESER. Refer to Section 3 *Status and Events* for more information about DESE.

The power-on default for the DESER is to set all bits to 1 if the power-on status flag is TRUE. If this flag is set to FALSE, the DESER maintains its current value through a power cycle.

Group

STATUS & EVENT

Related Commands

*CLS, *ESE, *ESR?, EVENT?, EVMsg?, EVQty?, *SRE, *STB?

Syntax

```
DESE <Bit Value>
DESE?
```

Arguments

`<Bit Value>::=<NR1>`
where <NR1> is a decimal integer, which must range from 0 to 255, that sets the DESER bits to its binary equivalent.

Examples

`:DESE 177`
sets the DESER to 177 (binary 10110001), which sets the PON, CME, EXE and OPC bits.

`:DESE?`
might return `:DESE 176`, which indicates that the DESER contains the binary number 10110000.

DIAGnostic?

The DIAGnostic? query returns the selected self-test routine(s), runs the routine, and returns the results.

Group DIAGNOSTIC

Related Commands DIAGnostic:SELect, DIAGnostic:STATe, DIAGnostic:RESUlt?

Syntax DIAGnostic?

Arguments None

Responses [:DIAGNOSTIC:SELECT] <Self-test Routine>; [RESULT],<Result>[, <Result>]...
<Self-test Routine>::= <label>
where <label> is one of following routines:

ALL	all routines
CPU	CPU unit check routine
DISPlay	display unit check routine
FPANel	front panel control unit check routine
CLOCk	clock unit check routine
TRIGger	trigger unit test routine
PMEMory	pattern memory check routine
SMEMory	sequence memory check routine

and where <Result>::=<NR1> is one of following responses:

0	terminated without error
100	detected an error in the CPU unit
200	detected an error in the display unit
300	detected an error in the front panel unit
400	detected an error in the clock unit
500	detected an error in the trigger unit
600	detected an error in the sequence memory
700	detected an error in the pattern memory

***NOTE**. The does not respond to any commands or queries issued during Self Test.*

Examples

```
DIAGNOSTIC?
```
might return `:DIAGNOSTIC:SELECT ALL;RESULT 0.`

DIAGnostic:RESUlt?

The `DIAGnostic:RESUlt?` query returns the results of self-test execution.

Group DIAGNOSTIC

Related Commands `DIAGnostic:SELect`, `DIAGnostic:STATe`

Syntax

```
DIAGnostic:RESUlt?
```

Arguments None

Responses

```
:DIAGNOSTIC:RESULT<Result>[,<Result>]...
<Result>::=<NR1>
```
where `<NR1>` is one of following values:

0	terminated without error
100	detected an error in the CPU unit
200	detected an error in the display unit
300	detected an error in the front panel unit
400	detected an error in the clock unit
500	detected an error in the trigger unit
600	detected an error in the sequence memory
700	detected an error in the pattern memory

Examples

```
DIAGNOSTIC:RESULT?
```
might return `:DIAGNOSTIC:RESULT 200`

DIAGnostic:SELect (?)

The DIAGnostic:SELect command selects the self test routine. The DIAGnostic:SELect? query returns the currently selected routine. The DIAGnostic:STATe command executes the routine.

Group DIAGNOSTIC

Related Commands DIAGnostic:STATe, DIAGnostic:RESULt?

Syntax

```
DIAGnostic:SELect { ALL | CPU | DISPlay | FPANel | CLOCk |
TRIGger | SMEMory | PMEMory }

DIAGnostic:SELect?
```

Arguments

ALL	checks all routines that follow
CPU	checks the CPU unit
DISPlay	checks the display unit
FPANel	checks the front panel control unit
CLOCk	checks the clock unit
TRIGger	checks the trigger unit
SMEMory	checks the sequence memory
PMEMory	checks the pattern memory

Examples

```
:DIAGNOSTIC:SELECT CPU ; STATE EXECUTE
```

executes the CPU self-test routine.

DIAGnostic:STATe

The DIAGnostic:STATe command executes the self-test routine(s) selected with the DIAGnostic:SELect command. If an error is detected during execution, the routine that detected the error terminates. If all of the self-test routines are selected using the DIAGnostic:SELect command, self-testing continues with execution of the next self-test routine.

Group DIAGNOSTIC

Related Commands DIAGnostic:SELect, DIAGnostic:RESUlt?

Syntax

```
DIAGnostic:STATe EXECute
```

Arguments

EXECute
Performs the self-test using the selected routine.

Examples

:DIAGNOSTIC:SELECT ALL ; STATE EXECUTE ; RESULT?
executes all of the self-test routines. After all self-test routines finish, the results of the self tests are returned.

DISPlay?

The DISPlay? query returns all the settings set using the display commands.

Group

DISPLAY

Related Commands

None

Syntax

DISPlay?

Arguments

None

Responses

Returns the settings as a sequence of commands, suitable for sending as set commands later to restore a setup. See *Examples*.

Examples

DISPLAY?
might return :DISPLAY:BRIGHTNESS 0.7;CLOCK 0;DIMMER 1;ENABLE 1;MENU:NAME SETUP;STATE 1;:DISPLAY:WINDOW:TEXT:DATA " "

DISPlay:BRIGhtness (?)

The DISPlay:BRIGhtness command adjusts the brightness of the screen; the DISPlay:BRIGhtness? query returns the current brightness setting.

Group DISPLAY

Related Commands DISPlay?

Syntax
```
DISPlay:BRIGhtness <Value>
DISPlay:BRIGhtness?
```

Arguments <Value>::=<NRf>
where <NRf> is a real number ranging from 0 to 1.

Examples :DISPLAY:BRIGHTNESS 0.7
sets screen brightness to 70% of maximum intensity.

DISPlay:CLOCk (?)

The DISPlay:CLOCk command sets whether or not the data and time are displayed.

The DISPlay:CLOCk? query returns whether or not the data and time are displayed.

Group DISPLAY

Related Commands DISPlay?

Syntax
```
DISPlay:CLOCk {ON | OFF | 1 | 0}
DISPlay:CLOCk?
```

Arguments ON or 1
sets the data generator to display the date and time.

OFF or 0
sets the data generator to not display the date and time.

Responses

1 Date and time is currently displayed.
0 Date and time is currently not displayed.

Examples

```
:DISPLAY:CLOCK ON
```
sets the data generator to display the date and time.

DISPlay:DIMmer (?)

The `DISPlay:DIMmer` command sets whether or not the screen dimmer function operates. The `DISPlay:DIMmer?` query returns the on/off state of the screen dimmer function. When the dimmer function is on, if no front panel controls are used for about 10 minutes, the screen brightness is lowered automatically.

Group

DISPLAY

Related Commands

`DISPlay?`

Syntax

```
DISPlay:DIMmer {ON | OFF | 1 | 0}
DISPlay:DIMmer?
```

Arguments

`ON or 1`
enables the dimmer function.

`OFF or 0`
clears the dimmer function.

Responses

`[:DISPLAY:DIMMER] {1 | 0}`

Examples

```
:DISPLAY:DIMMER ON
```
turns the dimmer function on.

DISPlay:ENABle (?)

The DISPlay:ENABle command turns the display on or off. When security is turned on, once the display is set to off, it cannot be turned on again. The DISPlay:ENABle? query returns the on/off state of the display.

Group DISPLAY

Related Commands DISPlay?, DISPlay:MENU:STATe

Syntax
```
DISPlay:ENABle {ON | OFF | 1 | 0}
DISPlay:ENABle?
```

Arguments

ON or 1
turns the display on.

OFF or 0
turns the display off.

Responses [:DISPLAY:ENABLE] {1 | 0}

Examples
```
:DISPLAY:ENABLE OFF
```
turns the display off.

DISPlay:MENU?

The DISPlay:MENU? query returns the type and display state of the selected menu.

Group DISPLAY

Related Commands DISPlay?, DISPlay:MENU[:NAME], DISPlay:MENU:NAME?

Syntax DISPlay:MENU?

Arguments None

Responses [:DISPLAY:MENU:NAME] {SETUP | EDIT | APPLICATION | UTILITY};[STATE] {1 | 0}

Examples

```
:DISPlay:MENU?
```
might return `:DISPLAY:MENU:NAME SETUP;STATE 1`

DISPlay:MENU[:NAME]

The `DISPlay:MENU[:NAME]` command selects the menu to be displayed on the screen.

Group DISPLAY

Related Commands `DISPlay?`, `DISPlay:MENU?`, `DISPlay:MENU:NAME?`

Syntax

```
DISPlay:MENU[:NAME] {SETUp | EDIT | APPLication | UTILity}
```

Arguments

`SETUp`	displays the setup menu
`EDIT`	displays the edit menu
`APPLication`	displays the application menu
`UTILity`	displays the utility menu

Examples

```
:DISPLAY:MENU:NAME UTILITY
```
selects the UTILITY menu.

DISPlay:MENU:NAME?

The DISPlay:MENU:NAME? query returns the type of the selected menu.

Group DISPLAY

Related Commands DISPlay?, DISPlay:MENU?, DISPlay:MENU:[:NAME]

Syntax DISPlay:MENU:NAME?

Arguments None

Responses [:DISPLAY:MENU:NAME] {SETUP | EDIT | APPLICATION | UTILITY}

Examples DISPlay:MENU:NAME?
might return :DISPLAY:MENU:NAME EDIT

DISPlay:MENU:STATe (?)

The DISPlay:MENU:STATe command sets whether or not menus are displayed on the screen. The DISPlay:MENU:STATE? query returns whether or not menus are displayed on the screen. This command is equivalent to the DISPlay:ENABle command.

Group DISPLAY

Related Commands DISPlay?, DISPlay:ENABle, DISPlay:MENU?, DISPlay:MENU:[:NAME]

Syntax DISPlay:MENU:STATe {ON | OFF | 1 | 0}
DISPlay:MENU:STATe?

Arguments

ON or 1	Menus are displayed.
OFF or 0	Menus are not displayed.

Responses [:DISPLAY:MENU:STATE] {1 | 0}

Examples DISPLAY:MENU:STATE ON
sets the instrument to display menus on the screen.

DISPlay[:WINDow]:TEXT:CLEar

The DISPlay[:WINDow]:TEXT:CLEar command clears the message display area on the screen.

Group DISPLAY

Related Commands DISPlay?, DISPlay[:WINDow]:TEXT[:DATA]

Syntax DISPlay[:WINDow]:TEXT:CLEar

Arguments None

Examples :DISPLAY:WINDOW:TEXT:CLEAR
clears the message display area.

DISPlay[:WINDow]:TEXT[:DATA] (?)

The DISPlay[:WINDow]:TEXT[:DATA] command sends a message to be displayed in the screen message display area. The sent message is displayed immediately. The DISPlay[:WINDow]:TEXT[:DATA]? query returns the contents of the input screen message.

***NOTE**. The contents of the message display area scrolls automatically. To fully update the display contents, first clear the message display area using the DISPlay[:WINDow]:TEXT:CLEar command.*

Group DISPLAY

Related Commands DISPlay?, DISPlay[:WINDow]:TEXT:CLEar

Syntax
```
DISPlay[:WINDow]:TEXT[:DATA] <Message>
DISPlay[:WINDow]:TEXT[:DATA]?
```

Arguments
<Message>::=<string>
where <string> is a message character string.

Responses [:DISPLAY:WINDOW:TEXT:DATA] <Message>

Examples
```
:DISPLAY:WINDOW:TEXT:DATA "ABCD"
```
sends the text ”ABCD” to be displayed in the message display area.

*ESE (?)

The *ESE common command sets the bits of the ESER (Event Status Enable Register) used in the status and events reporting system of the data generator. The *ESE? query returns the contents of the ESER. Refer to Section 3 *Status and Events* for more information about the ESER.

If the power on status flag is TRUE, the power-on default for the ESER is to reset all bits to zero. If this flag is set to FALSE, the ESER bits do not change value during the power-on cycle.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESR?, EVENT?, EVMsg?, EVQty?, *SRE, *STB?

Syntax
```
*ESE <Bit Value>
*ESE?
```

Arguments
<Bit Value>::=<NR1>
where <NR1> is a decimal integer that ranges from 0 to 255. The ESER bits will be set to the binary equivalent of the decimal integer sent.

Examples
*ESE 177
sets the ESER to 177 (binary 10110001), which sets the PON, CME, EXE and OPC bits.

*ESE?
might return 176, which indicates that the ESER contains the binary number 11010000.

*ESR?

The *ESR? common query returns the contents of SESR (Standard Event Status Register) used in the status and events reporting system. Refer to Section 3 *Status and Events* for more information about *ESR? or SESR.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE?, EVENT?, EVMsg?, EVQty?, *SRE, *STB?

Syntax
```
*ESR?
```

Arguments None

Examples
*ESR?
might return 181, which indicates that the SESR contains the binary number 10110101.

EVENT?

The EVENT? query dequeues the event code of the event that has been in the Event Queue the longest out of all available events. Use the *ESR? query to make the events available for dequeuing using EVENT?. Refer to Section 3 *Status and Events.*

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR?, EVMsg?, EVQty?, *SRE, *STB?

Syntax EVENT?

Arguments None

Examples EVENT?
might return :EVENT 113

EVMsg?

The EVMsg? query dequeues the event code and event message of the event that has been in the Event Queue the longest out of all available events. Use the *ESR? query to make the events available for dequeuing using EVMsg? For more details, refer to Section 3 *Status and Events.*

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR?, EVENT?, EVQty?, *SRE, *STB?

Syntax EVMsg?

Arguments None

Examples :EVMSG?
might return :EVMSG 420,"Query UNTERMINATED".

EVQty?

The EVQty? query returns the number of events currently in the Event Queue. If no event is being queued, 0 is returned.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR, EVMsg?, EVENT?, *SRE, *STB?

Syntax EVQty?

Arguments None

Examples :EVQty?
might return :EVQTY 5.

FACTory

The FACTory command resets the data generator to its factory default settings and purges all stored settings. (See Appendix D, page D–1, for a list of the factory settings.)

Group SYSTEM

Related Commands *RST, SECUre

Syntax FACTory

Arguments None

Examples :FACTORY
resets the data generator to its factory default settings.

HCOPy?

The HCOPy? query returns the set image data format and output port for hardcopy output.

Group HARDCOPY

Related Commands HCOPy:FORMat, HCOPy:PORT

Syntax HCOPy?

Arguments None

Responses [:HCOPY:FORMAT] {BMP | EPSON | EPSMONO | THINKJET | TIFF};[:PORT] {DISK | GPIB | RS232C}
where

BMP
the Windows monochrome file format.

EPSOn
the format used by 9-pin and 24-pin dot matrix printers in ESC/P graphics mode.

EPSMono
the encapsulated Postscript format monochrome image file format.

THINkjet
the format used by HP inkjet printers.

TIFF
the TIFF format.

Examples HCOPY?
might return :HCOPY:FORMAT TIFF ; PORT DISK

In this case the instrument outputs hardcopy data to file on the floppy disk in the TIFF format.

HCOPy:ABORt

The HCOPy:ABORt command aborts hardcopy output.

Group HARDCOPY

Related Commands HCOPy:STARt

Syntax HCOPy:ABORt

Arguments None

Examples :HCOPY:ABORT
aborts hardcopy output.

HCOPy:DATA?

The HCOPy:DATA? query outputs the hard copy data to the output queue. However, note that this command has no effect on (and is not affected by) the hard copy output port setting.

Group HARDCOPY

Related Commands HCOPy:PORT

Syntax HCOPy:DATA?

Arguments None

Responses [:HCOPY:DATA] <Image>
where

<Image>::=<block> the hardcopy image data block

Examples :HCOPY:DATA?
outputs hard copy data to the output queue.

HCOPy:FORMat (?)

The HCOPy:FORMat command sets the hard copy output format.

The HCOPy:FORMat? query returns the currently specified hard copy output format.

Group HARDCOPY

Related Commands HCOPy?

Syntax
```
HCOPy:FORMAT {BMP | EPSOn | EPSMono | THINkjet | TIFF}
HCOPy:FORMAT?
```

Arguments

BMP
the Windows monochrome file format.

EPSOn
the format used by 9-pin and 24-pin dot matrix printers in ESC/P graphics mode.

EPSMono
the encapsulated Postscript format monochrome image file format.

THINkjet
the format used by HP inkjet printers.

TIFF
the TIFF format.

Responses
```
[:HCOPY:FORMAT] {BMP | EPSON | EPSMONO | THINKJET | TIFF}
```

Examples
```
:HCOPY:FORMAT TIFF
```
sets the data generator to output hard copy in the TIFF format.

HCOPy:PORT (?)

The HCOPy:PORT command sets the hard copy output port.

The HCOPy:PORT? query returns the currently specified hard copy output port.

Group HARDCOPY

Related Commands HCOPy?

Syntax
```
HCOPy:PORT {DISK | GPIB | RS232c}
HCOPy:PORT?
```

Arguments

DISK
outputs to a file on the floppy disk.

GPIB
outputs to the GPIB port.

RS232c
outputs to the RS-232C port.

Responses [HCOPy:PORT] {DISK | GPIB | RS232c}

Examples
```
:HCOPY:PORT DISK
```
sets the hard copy output to be to a file on the floppy disk.

HCOPy:STARt

The HCOPy:STARt command starts hardcopy output.

Group HARDCOPY

Related Commands HCOPy:ABORt

Syntax HCOPy:STARt

Arguments None

Examples :HCOPY:START
starts hardcopy output.

HEADer (?)

The HEADer command enables or disables the command header responses to all queries except IEEE Std 488.2 common commands. The HEADer? query returns the status indicating whether the command header responses are enabled or not.

Group SYSTEM

Related Commands VERBose

Syntax HEADer {ON | OFF | <NR1>}
HEADer?

Arguments ON or nonzero value
enables the command header responses.

OFF or zero value
disables the command header responses.

Responses

1	command header responses are currently enabled.
0	command header responses are currently disabled.

Examples

```
:HEADER OFF
```
disables the command header responses.

```
:HEADER?
```
might return 1 which indicates command headers are currently enabled for return in query responses.

ID?

The ID? query returns the ID information of the data generator.

Group SYSTEM

Related Commands *IDN?

Syntax

```
ID?
```

Arguments None

Responses

```
ID <Manufacturer>/<Model>, <Firmware Level>
```
where
<Manufacturer>::=SONY_TEK,
<Model>::=DG2020A
<Firmware Level>::=CF:<Code and Format Version>, and
FV:<Firmware Version>.

Examples

```
:ID?
```
returns SONY_TEK/DG2020A,CF:91.1CN,FV:1.00

*IDN?

The *IDN? common query returns the ID information of the data generator.

Group SYSTEM

Related Commands ID?

Syntax *IDN?

Arguments None

Responses <Manufacturer>, <Model>, <Serial Number>, <Firmware Level>
where
<Manufacturer>::=SONY/TEK,
<Model>::=DG2020A,
<Serial Number>::=0,
<Firmware Level>::=CF:<Code and Format Version>,
<sp>FV:<Firmware Version>, and
<sp>::= Space.

Examples *IDN?
might return SONY/TEK,DG2020A,0,CF:91.1CN FV:1.00

LOCk (?)

The LOCk command enables or disables the knob and all front panel buttons except the ON/STBY button.

The LOCk? query returns a status indicating whether the knob and the buttons are locked or not.

These data generators do not switch between remote control and local control modes, but rather allow simultaneous setting from an external controller and from the front panel. Use this command to lock the functions of the front panel buttons and knobs to disable front panel operations during operation from an external controller or during external controller software execution.

***NOTE**. When the front panel control operations are locked out by the LOCk command, the instrument displays the character string "FP: LOCKED" at the upper right of the screen.*

Group SYSTEM

Related Commands `UNLock`

Syntax
```
LOCk {ALL | NONe}
LOCk?
```

Arguments

`ALL`
disables the front panel buttons and the knob except the ON/STBY button.

`NONe`
enables the front panel buttons and the knob.

Examples
```
:LOCk ALL
```
disables the front panel buttons and the knob.

MMEMory:CATalog[:ALL]?

The MMEMory:CATalog[:ALL]? query returns a list of all files and directories in the current directory on the floppy disk.

Group MEMORY

Related Commands MMEMory:CATalog:ORDer

Syntax MMEMory:CATalog[:ALL]?

Arguments None

Responses [:MMEMORY:CATALOG:ALL] <File Entry>[,<File Entry>]...
where

<File Entry>::=<File Name>,<File Size>,<Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

***NOTE**. A file size of 0 is returned for subdirectories.*

MMEMory:CATalog:ORDer (?)

The MMEMory:CATalog:ORDer command sets the display order for file information in disk directory listings. The MMEMory:CATalog:ORDer? query returns the display order for file information in disk directory listings.

Group MEMORY

Related Commands MMEMory:CATalog[:ALL]?

Syntax MMEMory:CATalog:ORDer {NAME1 | NAME2 | TIME1 | TIME2}
MMEMory:CATalog:ORDer?

Arguments NAME1
orders the display according to the ASCII collating sequence of the file names.

NAME2
orders the display in the reverse order of the NAME1 order.

TIME1
orders the display with older (Date and Time) files first.

TIME2
orders the display with more recent (Date and Time) files first.

Responses [:MMEMORY:CATALOG:ORDER] {NAME1 | NAME2 | TIME1 | TIME2}

Examples :MMEMORY:CATALOG:ORDER NAME1
sets the order of file information recorded in disk directory listings to alphabetical order by file name.

MMEMory:CDIRectory (?)

The MMEMory:CDIRectory command changes the current working directory. The MMEMory:CDIRectory? query returns the current working directory path.

Group MEMORY

Related Commands MMEMory:MDIRectory

Syntax
```
MMEMory:CDIRectory <Directory Path>
MMEMory:CDIRectory?
```

Arguments <Directory Path>::=<string>
where <string> is the name of the new current working directory.

Responses [:MMEMORY:CDIRECTORY] <Directory Path>

Examples :MMEMORY:CDIRECTORY "\DG\WORK3"
changes the current working directory to \DG\WORK3.

MMEMory:COPY

The MMEMory:COPY command copies a file on the disk and creates a new file. If the copy destination file already exists, an error is issued and the existing file is not overwritten.

Group MEMORY

Related Commands MMEMory:DELete:ALL, MMEMory:DELete[:NAME]

Syntax MMEMory:COPY <From-path>,<To-path>

Arguments <From-path>::=<string>
where <string> is the path name of the source file.

<To-path>::=<string>
where <string> is the path name of the destination file.

Examples :MMEMORY:COPY "MYDATA.PDA","MYWORK.PDA"
copies the file MYDATA.PDA in the current directory and creates a new file, MYWORK.PDA, in the current directory.

MMEMory:DELete:ALL

The MMEMory:DELete:ALL command deletes all files and subdirectories in the current directory. However, non–empty subdirectories are not deleted.

Group MEMORY

Related Commands MMEMory:DELete[:NAME]

Syntax MMEMory:DELete:ALL

Arguments None

Examples :MMEMORY:DELETE:ALL
deletes all files and empty subdirectories in the current directory.

MMEMory:DELete[:NAME]

The MMEMory:DELete[:NAME] command deletes the file or subdirectory with the specified path name. However, non–empty subdirectories are not deleted.

Group MEMORY

Related Commands MMEMory:DELete:ALL

Syntax MMEMory:DELete[:NAME] <Path Name>

Arguments <Path Name>::=<string>
where <string> is the path name of the file or subdirectory to be deleted.

Examples :MMEMORY:DELETE "NOMORE.PDA"
deletes the file NOMORE.PDA in the current directory.

MMEMory:FREE?

The MMEMory:FREE? query returns used size and unused size of the mass memory. This query is equivalent to the MMEMory:FREE:ALL? query.

Group MEMORY

Related Commands

Syntax MMEMory:FREE?

Arguments None

Responses :MMEMORY:FREE <Used Size>, <Unused Size>
where
<Used Size>::=<NR1> and
<Unused Size>::=<NR1>.

Examples :MMEMORY:FREE?
might return :MMEMORY:FREE 104584,1352704

MMEMory:INITialize

The MMEMory:INITialize command formats a floppy disk. The format type is specified by the argument.

Group MEMORY

Related Commands

Syntax `MMEMory:INITialize {DD1 | DD2 | HD1 | HD2 | HD3}`

Arguments You can select from the following formats:

Argument	Description
DD1	2DD, 720 KB, 80 tracks, 9 sectors/track, 512 bytes/sector. Format for IBM PC 2DD and Toshiba J3100 2DD.
DD2	2DD, 640 KB, 80 tracks, 8 sectors/track, 512 bytes/sector. Format for NEC PC–9800 2DD.
HD1	2HD, 1.232 MB, 77 tracks, 15 sectors/track, 1,024 bytes/sector. Format for NEC PC–9800 2HD.
HD2	2HD, 1.200 MB, 80 tracks, 15 sectors/track, 512 bytes/sector. Format for Toshiba J3100 2HD.
HD3	2HD, 1.440 MB, 80 tracks, 18 sectors/track, 512 bytes/sector. Format for IBM PC 2HD.

Examples `:MMEMORY:INITIALIZE HD3`
formats a floppy disk for IBM PC 2HD.

MMEMory:LOAD

The MMEMory:LOAD command loads in pattern data and block, group, sequence, and setup information in DG2020A format into the instrument's internal memory from a disk file.

Group MEMORY

Related Commands MMEMory:SAVE

Syntax `MMEMory:LOAD <File Name>`

Arguments `<File Name>::=<string>`
where `<string>` is the name of the file to be loaded.

Examples `:MMEMORY:LOAD "MYDATA.PDA"`
loads all information from the file MYDATA.PDA in the current directory into the instrument's internal memory.

MMEMory:LOCK (?)

The `MMEMory:LOCK` command sets and clears file locks. When a file is locked, it cannot be deleted or written to. The `MMEMory:LOCK?` query returns whether or not the file is locked.

Group MEMORY

Related Commands

Syntax
```
MMEMory:LOCK <Path Name>,{ON | OFF | 1 | 0}
MMEMory:LOCK? <Path Name>
```

Arguments `<Path Name>::=<string>`
where `<string>` is the name of the file to be locked or unlocked.

`ON or 1`
locks the file.

`OFF or 0`
unlocks the file.

Responses

0	the file is not locked.
1	the file is locked.

Examples `:MMEMORY:LOCK "COUNT1.PDA",ON`
locks the file COUNT1.PDA in the current directory.

MMEMory:MDIRectory

The MMEMory:MDIRectory command creates a new subdirectory. The command is invalid if a directory with the specified name already exists.

Group MEMORY

Related Commands MMEMory:CDIRectory

Syntax MMEMory:MDIRectory <Directory Path>

Arguments <Directory Path>::=<string>
where <string> is the name or path of the new directory.

Examples :MMEMORY:MDIRECTORY "WORK4"
creates the new directory WORK4 in the current working directory.

MMEMory:RDIRectory

The MMEMory:RDIRectory command removes a subdirectory. If a file exist in the subderectory, this command will not be performed.

Group MEMORY

Related Commands MMEMory:CDIRectory, MMEMory:MDIRectory

Syntax MMEMory:RDIRectory <Directory Path>

Arguments <Directory Path>::=<string>
where <string> is the name of the directory to be removed.

Examples :MMEMORY:RDIRECTORY "WORK4"
removes the directory WORK4 in the current working directory.

MMEMory:RENname

The MMEMory:RENname command changes the name of the specified file. A file that is locked using the MMEMory:LOCk command cannot be renamed.

Group MEMORY

Related Commands MMEMory:COPY

Syntax MMEMory:RENname <From-filename>, <To-filename>

Arguments <From-filename>::=<string>
where <string> is the name of the file to be changed.

<To-filename>::=<string>
where <string> is the name of the file after it is changed.

Examples :MMEMORY:RENAME "COUNT1.PDA","COUNT2.PDA"
changes the name of the file COUNT1.PDA in the current working directory to COUNT2.PDA.

MMEMory:SAVE

The MMEMory:SAVE command saves the pattern data and block, group, sequence, and setup information stored in the internal memory into a disk file in DG2020A format.

Group MEMORY

Related Commands MMEMory:LOAD

Syntax MMEMory:SAVE <Path Name>

Arguments <Path Name>::=<string>
where <string> is the path name of the file.

Examples :MMEMORY:SAVE "NEWDATA.PDA"
saves all the information in internal memory to the file NEWDATA.PDA in the current working directory.

MODE?

The MODE? query returns all the setting states related to the pattern generation mode.

Group MODE

Related Commands MODE:STATe, MODE:UPDate

Syntax MODE?

Arguments None

Responses [:MODE:STATE] {REPEAT | SINGLE | STEP | ENHANCED};[UPDATE] {AUTO | MANUAL}

Examples MODE?
might return :MODE:STATE REPEAT;UPDATE AUTO

Here, the run mode is set to repeat and the output pattern update method is set to automatic. (See the items on the MODE:STATe and MODE:UPDate commands.)

MODE:STATe (?)

The MODE:STATe command sets the run mode for pattern generation. The MODE:STATe? query returns the pattern generation run mode setting.

Group MODE

Related Commands MODE?

Syntax
MODE:STATe {REPeat | SINGle | STEp | ENHanced}
MODE:STATe?

Arguments

REPeat	Pattern data output is repeated.
SINGle	Pattern data output is performed exactly once.
STEp	Pattern data is output not according to the internal clock, but rather by a clock signal created by the STEP key.
ENHanced	Pattern data is output according to the defined sequence.

Responses `[:MODE:STATE] {REPEAT | SINGLE | STEP | ENHANCED}`

Examples `:MODE:STATE SINGLE`
sets the run mode to single.

MODE:UPDate (?)

The `MODE:UPDate` command sets the output pattern update method used when data related to pattern generation is changed. The `MODE:UPDate?` query returns the output pattern update method used when data related to pattern generation is changed.

Group MODE

Related Commands `MODE?`

Syntax
```
MODE:UPDate {AUTO | MANual}
MODE:UPDate?
```

Arguments

`AUTO`	Pattern output reflects changes each time the data is changed in any way.
`MANual`	Pattern output is not changed when data is changed until an update forcing command is received.

Responses `[:MODE:UPDATE] {AUTO | MANUAL}`

Examples `:MODE:UPDATE AUTO`
sets the output pattern update method to AUTO.

*OPC (?)

The *OPC common command causes bit 0 in the SESR (Standard Event Status Register) to be set, and the operation complete message to be issued, when all pending operations are finished.

The *OPC? query waits until all pending operations are finished and returns a “1” ASCII character.

Group SYNCHRONIZATION

Related Commands *WAI

Syntax
```
*OPC
*OPC?
```

Arguments None

Examples
```
HCOPY:PORT DISK;HCOPY START;*OPC
```
causes the SESR bit 0 to be set and the operation complete message to be issued on the completion of hardcopy.

*OPT?

The *OPT common query returns the implemented options of the data generator.

Group SYSTEM

Related Commands None

Syntax *OPT?

Arguments None

Responses <Option>[,<Option>]...
where

0	indicates no option,
UNIT1	indicates the option 01 (12 additional output channels) ,and
UNIT1,UNIT2	indicates the option 02 (24 additional output channels)

Examples *OPT?
might return UNIT1,UNIT2 to indicate that the option 02 is installed in the instrument.

OUTPut?

The OUTPUT? query returns all settings rerated to the channel and clock outputs.

Group OUTPUT

Related Commands DATA?

Syntax OUTPut?

Arguments None

Examples OUTPUT?

might return (when the POD A is only available and the TTL pod is connected to the POD A connector)

```
:OUTPUT:PODA:CH0:INHIBIT 0;ASSIGN 0;:OUTPUT:PODA:CH1:INHIBIT 0;ASSIGN 1;
:OUTPUT:PODA:CH2:INHIBIT 0;ASSIGN 2;:OUTPUT:PODA:CH3:INHIBIT 0;ASSIGN 3;
:OUTPUT:PODA:CH4:INHIBIT 0;ASSIGN 4;:OUTPUT:PODA:CH5:INHIBIT 0;ASSIGN 5;
:OUTPUT:PODA:CH6:INHIBIT 0;ASSIGN 6;:OUTPUT:PODA:CH7:INHIBIT 0;ASSIGN 7;
:OUTPUT:PODA:CH8:DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 8;
:OUTPUT:PODA:CH9:DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 9;
:OUTPUT:PODA:CH10:DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 10;
:OUTPUT:PODA:CH11:DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 11;
:OUTPUT:PODA:TYPE TTL;DEFINE #2750,0,0<LF>1,1,0<LF>2,2,0<LF>3,3,0<LF>
4,4,0<LF>5,5,0<LF>6,6,0<LF>7,7,0<LF>8,8,0<LF>9,9,0<LF>10,10,0<LF>11,11,0
```

or might return (when the POD A is only available and the Variable pod is connected to the POD A connector)

```
:OUTPUT:PODA:CH0:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 0;
:OUTPUT:PODA:CH1:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 1;
:OUTPUT:PODA:CH2:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 2;
:OUTPUT:PODA:CH3:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 3;
:OUTPUT:PODA:CH4:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 4;
:OUTPUT:PODA:CH5:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 5;
:OUTPUT:PODA:CH6:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 6;
:OUTPUT:PODA:CH7:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 7;
:OUTPUT:PODA:CH8:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 8;
:OUTPUT:PODA:CH9:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 9;
:OUTPUT:PODA:CH10:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 10;
:OUTPUT:PODA:CH11:HIGH 0.500;LOW -0.500;DELAY 0.00E-0.9;INHIBIT 0;ASSIGN 11;
:OUTPUT:PODA:TYPE VAR;DEFINE #2750,0,0<LF>1,1,0<LF>2,2,0<LF>3,3,0<LF>
4,4,0<LF>5,5,0<LF>6,6,0<LF>7,7,0<LF>8,8,0<LF>9,9,0<LF>10,10,0<LF>11,11,0;
:OUTPUT:ELEVEL 0.5;ILEVEL 0.5
```

OUTPut:ELEVel (?)

The OUTPut:ELEVel command sets the pod event input threshold level. An error is issued if the pod is not a variable level pod. The OUTPut:ELEVel? query returns the event input threshold level setting.

Group OUTPUT

Related Commands

Syntax
```
OUTPut:ELEVel <Volt>
OUTPut:ELEVel?
```

Arguments
<Volt>::=<NR2>[<Unit>]
where <Unit>::={V | mV} with a range of –5.0 V to 5.0 V in 0.1 V steps.

Responses
[OUTPUT:ELEVEL] <Volt>

Examples
:OUTPUT:ELEVEL 500mV
sets the event input threshold level to 500 mV.

OUTPut:ILEVel (?)

The OUTPut:ILEVel command sets the pod high–impedance control input (inhibit input) threshold level. An error is issued if the pod is not a variable level pod. The OUTPut:ILEVel? query returns the pod high–impedance control input threshold level setting.

Group OUTPUT

Related Commands

Syntax
```
OUTPut:ILEVel <Volt>
OUTPut:ILEVel?
```

Arguments
<Volt>::=<NR2>[<Unit>]
where <Unit>::={V | mV} with a range of –5.0 V to 5.0 V in 0.1 V steps.

Responses
[OUTPUT:ILEVEL] <Volt>

Examples

```
:OUTPUT:ILEVEL 300mV
```

sets the high–impedance control input threshold level to 300 mV.

OUTPut:POD<s>:CH<n>:ASSIGn (?)

The OUTPut:POD<s>:CH<n>:ASSIGn command assigns a data bit to the specified channel of the specified pod. The OUTPut:POD<s>:CH<n>:ASSIGn? query returns the data bit assigned to the specified channel of the specified pod. A bit number of –1 is returned if no data bit is assigned to the channel.

Group OUTPUT

Related Commands OUTPut:POD<s>:CH<n>:RELEase, OUTPut:POD<s>:DEFine

Syntax

```
OUTPut:POD<s>:CH<n>:ASSIGn <Bit>
OUTPut:POD<s>:CH<n>:ASSIGn?
(<s>::={A | B | C}, <n>::={0 to 11})
```

Arguments

<Bit>::=<NR1>
where <NR1> is a data bit number (0 to 35).

Responses

```
[:OUTPUT:POD<s>:CH<n>:ASSIGN] <Bit>
```

Examples

```
:OUTPUT:PODA:CH5:ASSIGN 3
```

assigns data bit D03 to pod A channel 5.

OUTPut:POD<s>:CH<n>:DELAy (?)

The OUTPut:POD<s>:CH<n>:DELAy command sets the delay time for the specified channel of the specified pod. The delay time can only be set for channels 8 through 11. The OUTPut:POD<s>:CH<n>:DELAy? query returns delay time setting for the specified channel of the specified pod.

Group OUTPUT

Related Commands

Syntax

```
OUTPut:POD<s>:CH<n>:DELAy <Time>
OUTPut:POD<s>:CH<n>:DELAy?
(<s>::={A | B | C}, <n>::={8 to 11})
```

Arguments

<Time>::=<NR2>[<unit>]
where <NR2> combined with [<unit>] specifies a time, and [<unit>]::= {s|ms|µs|ns}, for seconds, milliseconds, microseconds, or nanoseconds.

Responses

```
[:OUTPUT:POD<s>:CH<n>:DELAY] <Time>
```

Examples

```
:OUTPUT:PODB:CH8:DELAY 10ns
```

sets the delay time for pod B channel 8 to 10ns.

OUTPut:POD<s>:CH<n>:HIGH (?)

The OUTPut:POD<s>:CH<n>:HIGH command sets the high–level output voltage for . This command is only valid for variable level pods. The OUTPut:POD<s>:CH<n>:HIGH? query returns the high–level output voltage setting for the specified channel of the specified pod.

Group OUTPUT

Related Commands

```
OUTPut:POD<s>:CH<n>:LOW
```

Syntax

```
OUTPut:POD<s>:CH<n>:HIGH <Volt>
OUTPut:POD<s>:CH<n>:HIGH?
(<s>::={A | B | C}, <n>::={0 to 11})
```

Arguments

<Volt>::=<NR2>[<Unit>]
where <NR2> combined with [<Unit>]specifies a high–level voltage; and <Unit>::={V | mV}, for volt or millivolt.

Responses

```
[:OUTPUT:POD<s>:CH<n>:HIGH] <Volt>
```

Examples

```
:OUTPUT:PODA:CH11:HIGH 1V
```

sets the pod A channel 11 high–level output voltage to 1 V.

OUTPut:POD<s>:CH<n>:INHibit (?)

The OUTPut:POD<s>:CH<n>:INHibit command sets the control method for the output impedance of the specified channel of the specified pod. The OUTPut:POD<s>:CH<n>:INHibit? query returns the control method currently used for the output impedance of the pod's channel specified in the header.

Group OUTPUT

Related Commands

Syntax

```
OUTPut:POD<s>:CH<n>:INHibit {OFF | INTernal | EXTernal | BOTH | 0
| 1 | 2 | 3}
OUTPut:POD<s>:CH<n>:INHibit?
(<s>::={A | B | C}, <n>::={0 to 11})
```

Arguments

{OFF | 0}
No output impedance control

{INTernal | 1}
The output impedance is controlled by the pod's channel 0 signal.

{EXTernal | 2}
The output impedance is controlled by an external input signal (INH).

{BOTH | 3}
The output impedance is controlled by the logical OR of the pod's channel 0 signal and an external input signal (INH).

Responses

```
[:OUTPUT:POD<s>:CH<n>:INHIBIT] {0 | 1 | 2 | 3}
```

Examples

```
:OUTPUT:PODA:CH6:INHIBIT EXTERNAL
```

sets the output impedance of pod A channel 6 to be controlled by an external input signal (INH).

OUTPut:POD<s>:CH<n>:LOW (?)

The OUTPut:POD<s>:CH<n>:LOW command sets the low–level output voltage for the specified channel of the specified pod. This command is only valid for variable level pods. The OUTPut:POD<s>:CH<n>:LOW? query returns the low–level output voltage setting for the specified channel of the specified pod.

Group OUTPUT

Related Commands OUTPut:POD<s>:CH<n>:HIGH

Syntax
```
OUTPut:POD<s>:CH<n>:LOW <Volt>
OUTPut:POD<s>:CH<n>:LOW?
(<s>::={A | B | C}, <n>::={0 to 11})
```

Arguments
<Volt>::=<NR2>[<Unit>]
where <NR2> combined with [<Unit>]specifies a low–level voltage; and <Unit>::={V | mV}, for volt or millivolt.

Responses [:OUTPUT:POD<s>:CH<n>:LOW] <Volt>

Examples
:OUTPUT:PODC:CH7:LOW −1V
sets the pod C channel 7 low–level output voltage to –1 V.

OUTPut:POD<s>:CH<n>:RELEase

The OUTPut:POD<s>:CH<n>:RELEase command clears the data bit assignment for the specified channel of the specified pod.

Group OUTPUT

Related Commands OUTPut:POD<s>:CH<n>:ASSIGn, OUTPut:POD<s>:DEFine

Syntax
```
OUTPut:POD<s>:CH<n>:RELEase
(<s>::={A | B | C}, <n>::={0 to 11})
```

Arguments None

Examples
```
:OUTPUT:PODA:CH3:RELEASE
```
clears data bit assignment for pod A channel 3.

OUTPut:POD<s>:DEFine (?)

The OUTPut:POD<s>:DEFine command assigns data bits to all the channels of the pod specified in the header. The data bit assignment is cleared for any data bit not specified in the argument. The OUTPut:POD<s>:DEFine? query returns the data bits assigned to the channels of the pod specified in the header.

Group OUTPUT

Related Commands OUTPut:POD<s>:CH<n>:ASSIGn, OUTPut:POD<s>:CH<n>:RELEase

Syntax
```
OUTPut:POD<s>:DEFine <Assigninfo>
OUTPut:POD<s>:DEFine?
(<s>::={A | B | C})
```

Arguments

<Assigninfo>::=<blockheader><Assign>[<LF><Assign][<LF><Assign>]... arbitrary block data that defines the pod channel assignments

where,

<blockheader>::=<byte count digit><byte count>

<Assign>::=<AChannel>,<ABit>,<AHoldE>

The <AChannel>, <ABit>, and <AHoldE> items are ASCII character strings that express the following information.

<AChannel>	channel number (0 to 11)
<ABit>	data bit number (0 to 35)
<AHoldE>	high–impedance control selection (0: no control, 1: channel 0 control, 2: external input signal, 3: logical or of the channel 0 signal and the external input signal) (See the OUTPut:POD<s>:CH<n>:INHibit command.)

<LF>::= <ASCII line feed code (10)>

Responses

[:OUTPUT:POD<s>:DEFINE] <Assigninfo>
where <Assigninfo> is an arbitrary data block with the same format as the argument.

Examples

OUTPut:PODA:DEFine #2170,4,1<LF>1,5,2<LF>2,7,0
assigns the pod A channels as follows when executed.

Channel 0: Bit 4, high–impedance state controlled by the channel 0 signal
Channel 1: Bit 5, high–impedance state controlled by an external input signal
Channel 2: Bit 7, no high–impedance control
Other channels: Assignments cleared

OUTPut:POD<s>:TYPE?

The OUTPut:POD<s>:TYPE? query returns the type of the pod specified in the header.

Group OUTPUT

Related Commands

Syntax
```
OUTPut:POD<s>:TYPE?
(<s>::={A | B | C})
```

Arguments None

Responses
```
[OUTPut:POD<s>:TYPE] {NONE | TTL | VAR}
```
where

NONE	no pod
TTL	a TTL level pod
VAR	a variable level pod

Examples
```
:OUTPut:POD<s>:TYPE?
```
might return :OUTPUT:PODA:TYPE TTL, which indicates that pod A is a TTL level pod.

*PSC (?)

The *PSC common command controls the automatic power-on clearing of the ESER (Event Status Enable Register), the SRER (Service Request Enable Register), and DESER (Device Event Status Enable Register). These registers are used in the status and event reporting system.

The *PSC? common query returns status of the power-on status clear flag.

Group STATUS & EVENT

Related Commands DESE, *ESE, FACTory, *SRE

Syntax

```
*PSC <Power-On Status Clear>
*PSC?
```

Arguments

<Power-On Status Clear>::=<NR1>
where <NR1> is a decimal integer that must range from –32767 to 32767, the value of which determines whether power on clearing occurs as follows:

- Zero value — sets the power-on status clear flag to FALSE. When this flag is set FALSE, the values of the DESER, the SESR, and the ESER are restored at power on. With these values restored, the instrument can assert SRQ after powering on.
- Nonzero value — sets the power-on status clear flag to TRUE. When this flag is set TRUE, all the bits in the DESER are set and are reset in the SESR and ESER. This action prevents the instrument from asserting any SRQs after powering on.

Responses

- 1 — the power-on status clear flag is currently set to TRUE.
- 0 — the power-on status clear flag is currently set to FALSE.

Examples

```
*PSC 1
```
sets the power-on status flag to TRUE.

```
*PSC?
```
might return :0 to indicate that the power-on status clear flag is currently set to FALSE.

*RST

The *RST common command resets this data generator to the default state (default values are listed in Appendix D).

Group SYSTEM

Related Commands FACTory, SECUre

Syntax *RST

Arguments None

Examples *RST
resets the instrument.

RUNNing?

The RUNNing? query returns status that indicates whether or not pattern data (or a sequence) is being output.

Group MODE

Related Commands STARt, STOP

Syntax RUNNing?

Arguments None

Responses

1	pattern data or a sequence is being output.
0	nothing is being output.

Examples :RUNNING?
might return :RUNNING 1.

SOURce[:OSCillator]?

The SOURce[:OSCillator]? query returns all the settings states related to clock signals.

Group SOURCE

Related Commands SOURce:OSCillator:EXTernal:FREQuency, SOURce:OSCillator[:INTernal]:FREQuency, SOURce:OSCillator[:INTernal]:PLLlock, SOURce:OSCillator:SOURce

Syntax SOURce[:OSCillator]?

Arguments None

Responses [:SOURCE:OSCILLATOR:SOURCE] {INTERNAL | EXTERNAL};[EXTERNAL:FREQUENCY] <NR3>HZ;[:SOURCE:OSCILLATOR:INTERNAL:FREQUENCY] <NR3>HZ;[PLLLOCK] {1 | 0}

Examples :SOURce:OSCillator?
might return :SOURCE:OSCILLATOR:SOURCE INTERNAL;EXTERNAL:FREQUENCY 1.000E+8HZ;:SOURCE:OSCILLATOR:INTERNAL:FREQUENCY 2.000E+8HZ;PLLLOCK 1

SOURce:OSCillator:EXTernal:FREQuency (?)

The SOURce:OSCillator:EXTernal:FREQuency command enters the frequency of the externally supplied clock signal. The SOURce:OSCillator:EXTernal:FREQuency? query returns the externally supplied clock signal frequency setting.

Group SOURCE

Related Commands SOURce[:OSCillator]?, SOURce:OSCillator[:INTernal]:FREQuency, SOURce:OSCillator[:INTernal]:PLLlock, SOURce:OSCillator:SOURce

Syntax SOURce:OSCillator:EXTernal:FREQuency <Frequency>
SOURce:OSCillator:EXTernal:FREQuency?

Arguments <Frequency>::=<NR3>[<Unit>]
where <NR3> is a decimal number that combines with [<Unit>] to have a range of 10.00E–3～200.0E+6Hz, and [<Unit>]::={Hz | kHz | MHz}, for hertz, kilohertz or megahertz.

Responses [:SOURCE:OSCILLATOR:EXTERNAL:FREQUENCY] <NRf>HZ

Examples :SOURCE:OSCILLATOR:EXTERNAL:FREQUENCY 10.0MHZ
sets the external clock signal frequency input setting to 10 MHz.

SOURce:OSCillator[:INTernal]:FREQuency (?)

The SOURce:OSCillator[:INTernal]:FREQuency command sets the frequency of the internal clock oscillator. The SOURce:OSCillator[:INTernal]:FREQuency? query returns the internal clock oscillator frequency setting.

Group SOURCE

Related Commands SOURce[:OSCillator]?, SOURce:OSCillator:EXTernal:FREQuency, SOURce:OSCillator[:INTernal]:PLLlock, SOURce:OSCillator:SOURce

Syntax SOURce:OSCillator[:INTernal]:FREQuency <Frequency>
SOURce:OSCillator[:INTernal]:FREQuency?

Arguments <Frequency>::=<NR3>[<Unit>]
where <NR3> is a decimal number that combines with [<Unit>] to have a range of 10.00E–3～200.0E+6Hz, and [<Unit>]::={Hz | kHz | MHz}, for hertz, kilohertz or megahertz.

Responses [:SOURCE:OSCILLATOR:INTERNAL:FREQUENCY] <NRf>HZ

Examples :SOURCE:OSCILLATOR:INTERNAL:FREQUENCY 100MHZ
sets the internal clock oscillator frequency to 100 MHz.

SOURce:OSCillator[:INTernal]:PLLlock (?)

The SOURce:OSCillator[:INTernal]:PLLlock command sets whether or not the internal clock oscillator is phase synchronized (by PLL operation) with the reference oscillator. The SOURce:OSCillator[:INTernal]:PLLlock? query returns whether or not the internal clock oscillator is phase synchronized (by PLL operation) with the reference oscillator.

Group SOURCE

Related Commands SOURce[:OSCillator]?, SOURce:OSCillator:EXTernal:FREQuency, SOURce:OSCillator[:INTernal]:FREQuency, SOURce:OSCillator:SOURce

Syntax
```
SOURce:OSCillator[:INTernal]:PLLlock {ON | OFF | 1 | 0}
SOURce:OSCillator[:INTernal]:PLLlock?
```

Arguments

ON or 1	The phase is synchronized. (PLL on)
OFF or 0	The phase is not synchronized. (PLL off)

Responses [:SOURCE:OSCILLATOR:INTERNAL:PLLLOCK] {1 | 0}

Examples
```
:SOURCE:OSCILLATOR:INTERNAL:PLLLOCK ON
```
synchronizes the internal clock oscillator with the reference oscillator.

SOURce:OSCillator:SOURce (?)

The SOURce:OSCillator:SOURce command sets whether the internal clock oscillator or an external clock input signal is used as the clock signal source. The SOURce:OSCillator:SOURce? query returns whether the internal clock oscillator or an external clock input signal is used as the clock signal source.

Group SOURCE

Related Commands SOURce[:OSCillator]?, SOURce:OSCillator:EXTernal:FREQuency, SOURce:OSCillator[:INTernal]:FREQuency, SOURce:OSCillator[:INTernal]:PLLlock

Syntax
```
SOURce:OSCillator:SOURce {INTernal | EXTernal}
SOURce:OSCillator:SOURce?
```

Arguments

INTernal
use the internal clock source.

EXTernal
use an external clock source connected to the external clock input.

Responses

[:SOURCE:OSCILLATOR:SOURCE] {INTERNAL | EXTERNAL}

Examples

:SOURCE:OSCILLATOR:SOURCE INTERNAL
sets the internal clock oscillator to be used as the clock signal source.

SOURce:POD<s>:EVENT:ENABle (?)

The SOURce:POD<s>:EVENT:ENABle command enables or disables the EVENT input of the pod specified in the header. The SOURce:POD<s>:EVENT:ENABle? query returns whether or not the EVENT input of the specified pod is enabled.

Group

SOURCE

Related Commands

Syntax

SOURce:POD<s>:EVENT:ENABle {ON | OFF | 1 | 0}
SOURce:POD<s>:EVENT:ENABle?
(<s>::={A | B | C})

Arguments

ON or 1
enables the EVENT input.

OFF or 0
disables the EVENT input.

Responses

[:SOURCE:POD<s>:EVENT:ENABLE] {1 | 0}

Examples

:SOURCE:PODB:EVENT:ENABLE ON
enables the EVENT input of the pod B.

*SRE (?)

The *SRE common command sets the bits of the SRER (Service Request Enable Register). The *SRE? common query returns the contents of SRER.

The power-on default for the SRER is all bits reset if the power-on status flag is TRUE. If this flag is set to FALSE, the SRER maintains its value through a power cycle.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR?, EVENT?, EVMsg?, EVQty?, *STB?

Syntax

```
*SRE <Bit Value>
*SRE?
```

Arguments

<Bit Value>::=<NR1>
where the argument must be decimal number from 0 to 255. The SRER bits are set in binary bit according to the decimal number.

Examples

*SRE 48
sets the SRER to 48 (binary 00110000), which sets the ESB and MAV bits.

*SRE?
might return 32 which indicates that the SRER contains the binary number 00100000.

STARt

The STARt command sets the instrument to the start state. If the run mode is set to repeat or step, pattern data or sequence output starts. If the run mode is set to single, then the instrument goes to the trigger wait state.

Group MODE

Related Commands RUNNing?, STOP, *TRG

Syntax STARt

Arguments None

Examples :START
sets the instrument to the start state.

*STB?

The *STB? common query returns the value of the SBR (Status Byte Register). Bit 6 of the SBR is read as a MSS (Master Status Summary) bit. Refer to Section 3 *Status and Events*, for more details on the SBR.

Group STATUS & EVENT

Related Commands *CLS, DESE, *ESE, *ESR, EVENT?, EVMsg?, EVQty?, *SRE

Syntax *STB?

Arguments None

Responses <NR1>
which is a decimal number.

Examples *STB?
might return 96, which indicates that the SBR contains the binary number 01100000.

STOP

The STOP command stops pattern data or sequence output. If the run mode is set to single, the trigger wait state is cancelled.

Group MODE

Related Commands RUNNing?, STARt, *TRG

Syntax STOP

Arguments None

Examples :STOP
stops pattern data or sequence output.

SYSTem:DATE (?)

The SYSTem:DATE command sets the internal clock date. The SYSTem:DATE? query returns the internal clock date.

Group SYSTEM

Related Commands SYSTem:TIME

Syntax
```
SYSTem:DATE <Year>,<Month>,<Day>
SYSTem:DATE?
```

Arguments

<Year>::=<NR1>	the year
<Month>::=<NR1>	the month
<Day>::=<NR1>	the day

Responses [:SYSTEM:DATE] <Year>,<Month>,<Day>

Examples :SYSTEM:DATE 95,5,15
sets the date.

SYSTem:PPAUse (?)

The SYSTem:PPAUse command sets whether or not the instrument goes to the operator key input wait state (power–up pause) when an error is detected by the power–up diagnostics or no output pod is connected. The SYSTem:PPAUse? query returns the power–up pause setting (on or off).

Group SYSTEM

Related Commands

Syntax
```
SYSTem:PPAUse {ON | OFF | 1 | 0}
SYSTem:PPAUse?
```

Arguments

ON or 1
enables the power–up pause.

OFF or 0
disables the power–up pause.

Responses
```
[:SYSTEM:PPAUSE?] {1 | 0}
```

Examples
```
:SYSTEM:PPAUSE ON
```
turns power–up pause on.

SYSTem:SECurity:IMMediate

The SYSTem:SECurity:IMMediate command sets all internal settings to the factory setting state (the same state that results when the FACTory command is executed) and completely erases all data. Bit patterns, groups, blocks, and sequences are included in the erased data. The GPIB and RS-232-C settings, and the data and time settings are not reset.

Group SYSTEM

Related Commands FACTory, *RST

Syntax
```
SYSTem:SECurity:IMMediate
```

Arguments None

SYSTem:SECurity:STATe (?)

The SYSTem:SECurity:STATe command sets security to on or off. The SYSTem:SECurity:STATe? query returns whether the security setting is on or off. When the security setting is changed from on to off, the contents of internal memory are completely erased. The security on/off setting is not changed by executing the FACTory command.

Group SYSTEM

Related Commands SYSTem:SECurity:IMMediate

Syntax
```
SYSTem:SECurity:STATe {ON | OFF | 1 | 0}
SYSTem:SECurity:STATe?
```

Arguments

ON or 1
sets the security state to on.

OFF or 0
sets the security state to off.

Responses [:SYSTEM:SECURITY:STATE] {1 | 0}

Examples
```
:SYSTEM:SECURITY:STATE ON
```
sets the security state to on.

SYSTem:TIME (?)

The SYSTem:TIME command sets the internal clock time. The SYSTem:TIME? query returns the internal clock time.

Group SYSTEM

Related Commands SYSTem:DATE

Syntax
```
SYSTem:TIME <Hour>,<Minute>,<Second>
SYSTem:TIME?
```

Arguments

<Hour>	the hours
<Minute>	the minutes
<Second>	the seconds

Responses [:SYSTEM:TIME] <Hour>,<Minute>,<Second>

Examples

```
:SYSTEM:TIME 11, 23, 58
```

sets the time.

*TRG

The *TRG common command generates trigger event.

Group MODE

Related Commands RUNNing?, STARt, STOP

Syntax

```
*TRG
```

Arguments None

Examples

```
*TRG
```

generates trigger event.

TRIGger?

The TRIGger? query returns all of the currently specified settings related to the trigger function.

Group MODE

Related Commands TRIGger:IMPedance, TRIGger:LEVel, TRIGger:SLOPe

Syntax

```
TRIGger?
```

Arguments None

Examples

```
:TRIGGER?
```
might return `:TRIGGER:IMPEDANCE HIGH;LEVEL 1.400;SLOPE POSITIVE`

TRIGger:IMPedance (?)

The `TRIGger:IMPedance` command selects high impedance (1 kΩ) or low impedance (50 Ω) for the external trigger input connector.

The `TRIGger:IMPedance?` query returns currently selected impedance.

Group MODE

Related Commands `TRIGger:LEVel`, `TRIGger:SLOPe`

Syntax

```
TRIGger:IMPedance {HIGH | LOW}
TRIGger:IMPedance?
```

Arguments

`HIGH`
selects high impedance: 1 kΩ

`LOW`
selects low impedance: 50 Ω

Examples

```
:TRIGGER:IMPEDANCE LOW
```
selects low impedance.

TRIGger:LEVel (?)

The `TRIGger:LEVel` command sets the level on the external trigger at which the trigger event is generated. The `TRIGger:LEVel?` query returns the level currently set.

Group MODE

Related Commands `TRIGger:IMPedance`, `TRIGger:SLOPe`

Syntax

```
TRIGger:LEVel <Level>
TRIGger:LEVel?
```

Arguments <Level>::=<NR2>[<unit>]
where <unit>::={V | mV} with a range of –5.0 V to 5.0 V, in 0.1 V steps.

Examples :TRIGGER:LEVEL 200mV
sets the level to 200 mV.

TRIGger:SLOpe (?)

The TRIGger:SLOpe command selects the rising or falling edge of the external signal which generates the trigger event. The TRIGger:SLOPe? query returns status indicating which slope is currently selected.

Group MODE

Related Commands TRIGger:IMPedance, TRIGger:LEVel

Syntax TRIGger:SLOPe {POSitive | NEGative}
TRIGger:SLOPe?

Arguments POSitive
selects rising edge.

NEGative
selects falling edge.

Examples :TRIGGER:SLOPE POSITIVE
selects rising edge for trigger.

*TST?

The *TST? common query performs the self test and returns the results. If an error is detected during self test, execution stop immediately. This command takes up to 90 seconds to run the self test, and the data generator will not respond to any commands and queries while it runs.

Group DIAGNOSTIC

Related Commands DIAGnostic:RESUlt?, DIAGnostic:SELect, DIAGnostic:STATe

Syntax *TST?

Arguments None

Responses <Result>
where <Result>::=<NR1> and <NR1> is one of following arguments.

0	Terminated without error.
100	Detected an error in the CPU unit.
200	Detected an error in the display unit.
300	Detected an error in the front panel unit.
400	Detected an error in the clock unit.
500	Detected an error in the trigger unit.
600	Detected an error in the sequence memory.
700	Detected an error in the pattern memory.

Examples *TST?
might return 200 to indicate that errors were detected in the display unit.

UNLock

The UNLock command enables all front panel buttons and knob. This command is equivalent to the command LOCk NONe.

Group SYSTEM

Related Commands LOCk

Syntax UNLOCK ALL

Arguments ALL
enables the front panel buttons and knob.

Examples :UNLOCk ALL
enables the front panel buttons and knob.

UPTime?

The UPTIme? query returns the time elapsed since the data generator was powered on.

Group SYSTEM

Related Commands None

Syntax UPTime?

Arguments None

Examples :UPTIME 7.016
indicates the instrument has been powered on for 7.016 hours.

VERBose (?)

The VERBose command selects the long headers or the short headers to be returned with response messages. Longer response headers enhance readability for other programmers; shorter response headers provide faster bus transfer speed.

Group SYSTEM

Related Commands HEADer

Syntax

```
VERBose {ON | OFF | <NR1>}
VERBose?
```

Arguments

ON or nonzero value
selects long response header.

OFF or zero value
selects short response header.

Responses Responses are decimal numbers (<NR1>) and are defined as follows.

1 Long header is currently selected.
0 Short header is currently selected.

Examples

:VERBOSE ON
sets long header for query responses.

:VERBOSE?
might return :VERBOSE 1, which indicates that the long response header is currently selected.

*WAI

The *WAI common command prevents the data generator from executing any further commands or queries until all pending operations are completed.

Group SYNCHRONIZATION

Related Commands *OPC

Syntax *WAI

Arguments None

Examples *WAI
prevents the execution of any commands or queries until all pending operations complete.

Retrieving Response Messages

The method used for retrieving response messages differs depending on whether a GPIB interface or an RS-232-C interface is used. Figures 2-3 and 2-4 give an overview of these methods.

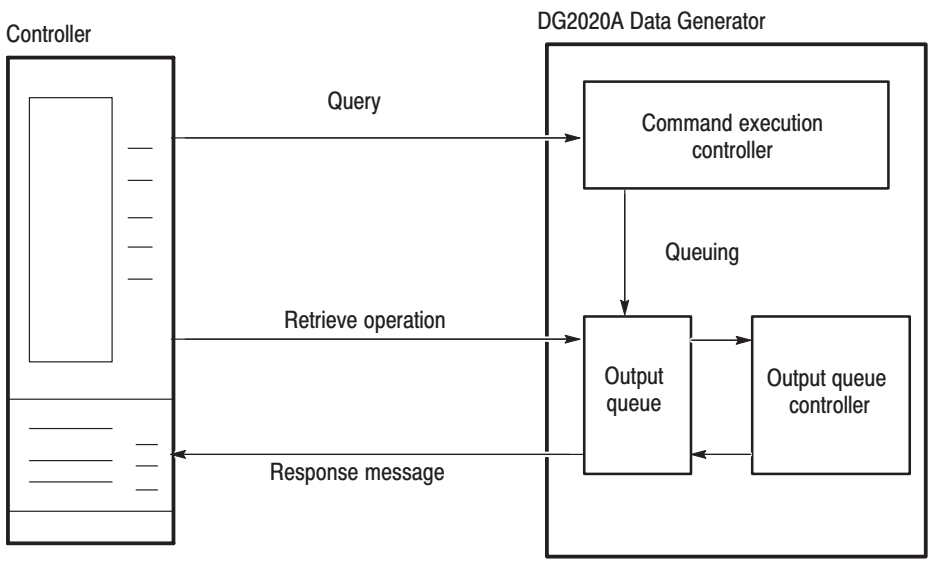


Figure 2-3: GPIB: Retrieving response messages

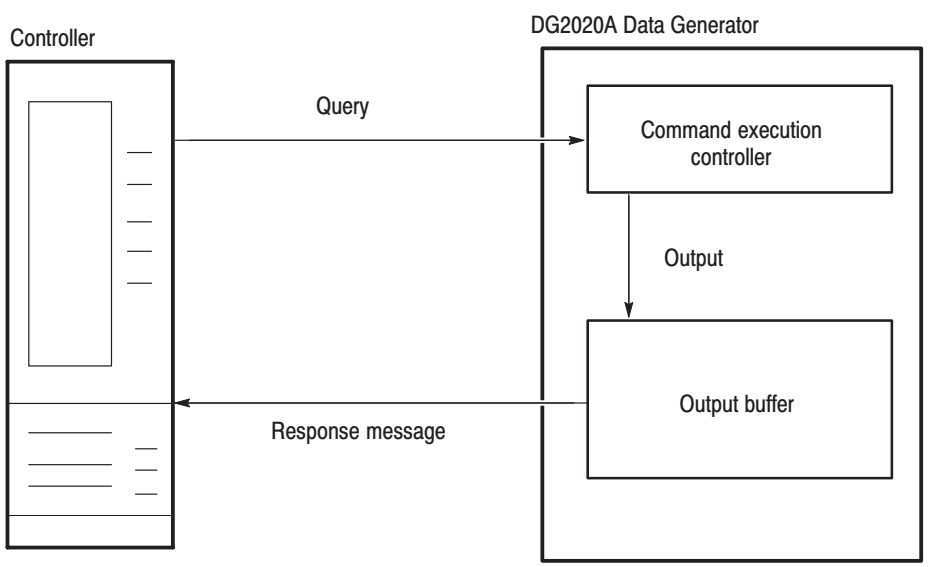


Figure 2-4: RS-232-C: Retrieving response messages

Figure 2-3 shows the response message retrieval operation when a GPIB interface is used. When a query command is sent from the external controller the data generator puts the response message for the query on the output queue. This response message cannot be retrieved unless the user performs a retrieval operation through the external controller.

If there is a response message queued in the output queue and another query command is sent from the external controller before a retrieval operation for the earlier message is performed, the data generator will delete the queued response message and put the response message for the more recently sent query command in the output queue.

The SBR (status byte register) MAV bit can be used to check the response message queuing state. See Section 3, “Status and Events”, for more information on the output queue, SBR, and control methods.

Figure 2-4 shows the response message retrieval operation when an RS-232-C interface is used. When a query command is sent from the external controller, the data generator immediately sends the response message to the external controller through an output buffer. As a result, when either a dumb terminal or a terminal emulator program running on a PC is used as the external controller, the response message will be displayed on the CRT immediately after the query command is typed in.

Unlike the GPIB interface, if an RS-232-C interface is used, response messages will never be deleted even if query commands are sent one after another.

Status and Event Reporting

Status and Event Reporting

This section describes how the DG2020A Data Generator reports its status and internal events for both the GPIB and RS-232-C interfaces. It describes the elements that comprise the status and events reporting system and explains how status and events are handled.

The status and event reporting system reports certain significant events that occur within the data generator. It is made up of five registers plus two queues. Four of the registers and one of the queues are compatible with IEEE Std 488.2-1987; the other register and queue are specific to Tektronix.

Registers

The registers fall into two functional groups:

- Status registers which store information about the status of data generator. They include the Standard Event Status Register (SESR) and the Status Byte Register (SBR).
- Enable registers which determine whether certain events are reported to the Status Registers and the Event Queue. They include the Device Event Status Enable Register (DESER), the Event Status Enable Register (ESER), and the Service Request Enable Register (SRER).

Status Registers

The Standard Event Status Register (SESR) and the Status Byte Register (SBR) record certain types of events that may occur while the data generator is in use. IEEE Std 488.2-1987 defines these registers.

Each bit in a Status Register records a particular type of event, such as an execution error or service request. When an event of a given type occurs, the data generator sets the bit that represents that type of event to a value of one. (You can disable bits so that they ignore events and remain at zero. See the Enable Registers section on page 3-4.) Reading the status registers tells you what types of events have occurred.

The Standard Event Status Register (SESR). The SESR, shown in Figure 3-1, records eight types of events that can occur within the data generator. Use the `*ESR?` query to read the SESR register. Reading the register clears the bits of the register, so that the register can accumulate information about new events.

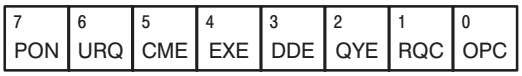


Figure 3-1: Standard event status (SESR)

Table 3-1: SESR bit functions

Bit	Function
7 (MSB)	**PON** (Power On). Indicates that the data generator was powered on.
6	**URQ** (User Request). Indicates an event occurred and because of that event the data generator needs attention from the operator.
5	**CME** (Command Error). Indicates that an error occurred while the data generator was parsing a command or query. Command error messages are listed in Table 3-5 on page 3-10.
4	**EXE** (Execution Error). Indicates that an error occurred while the data generator was executing a command or query. An execution error occurs for either of the following reasons: ■ A value designated for the argument is out of the range allowed by the data generator, is not valid for the command, or is incorrect in some other sense. ■ Execution took place improperly under conditions different from those which should have been requested. Execution error messages are listed in Table 3-6 on page 3-12.
3	**DDE** (Device Dependent Error). Indicates that a device-specific error occurred. Device error messages are listed in Table 3-7 on page 3-14.
2	**QYE** (Query Error). Indicates that an error occurred upon attempting to read the output queue. Such an error occurs for one of the following two reasons. ■ An attempt was made to retrieve a message from the output queue even through it is empty or pending. ■ Output queue message was cleared while it was being retrieved from the output queue.
1	**RQC** (Request Control).The data generator does not use this bit. Request Control (RQC) is used to show that an instrument has requested to transfer bus control back to the controller. (This is the usage prescribed by the IEEE Std. 488.1.)
0 (LSB)	**OPC** (Operation Complete). Indicates that the operation is complete. This bit is set when all pending operations complete following a `*OPC` command.

The Status Byte Register (SBR). shown in Figure 3-2, records whether output is available in the Output Queue, whether the data generator requests service, and whether the SESR has recorded any events.

Use a Serial Poll or the *STB? query to read the contents of the SBR. The bits in the SBR are set and cleared depending on the contents of the SESR, the Event Status Enable Register (ESER), and the Output Queue. When you use a Serial Poll to obtain the SBR, bit 6 is the RQS bit. When you use the *STB? query to obtain the SBR, bit 6 is the MSS bit. Reading the SBR does not clear the bits, including the MSS bit.

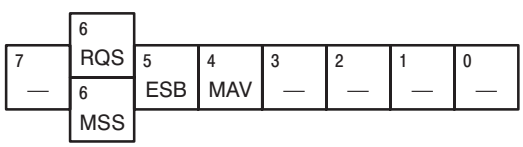


Figure 3-2: Status byte register (SBR)

Table 3-2: SBR bit functions

Bit	Function
7 (MSB)	Not used. (Must be set to zero for data generator operation.
6	The **RQS** (Request Service) bit, when obtained from a serial poll. Shows that the data generator requests service from the GPIB controller (that is, the SRQ line is asserted on the GPIB). This bit is cleared when the serial poll completes.
6	The **MSS** (Master Status Summary) bit, when obtained from *STB? query. Summarizes the ESB and MAV bits in the SBR. (In other words, that status is present and enabled in the SESR or a message is available at the Output Queue or both.)
5	The **ESB** (Event Status Bit). Shows that status is enabled and present in the SESR.[1]
4	The **MAV** (Message Available) bit . Shows that output is available in the Output Queue.
3 – 0	Not used. (Must be set to zero for data generator operation.

[1] **When operating over the RS-232-C interface, you can read the contents of the SBR using the *STB? query. However, this bit (ESB) is the only SBR bit of any significance to RS-232-C operation.**

Enable Registers

You use the DESER (Device Event Status Enable Register), the ESER (Event Status Enable Register), and the SRER (Service Request Enable Register) to select which events are reported to the Status Registers and the Event Queue. Each of these Enable Registers acts as a filter to a Status Register (the DESER also acts as a filter to the Event Queue) and can allow or prevent information from being recorded in the register or queue.

Each bit in an Enable Register corresponds to a bit in the Status Register it controls. In order for an event to be reported to its bit in the Status Register, the corresponding bit in the Enable Register must be set to one. If the bit in the Enable Register is set to zero, the event is not recorded.

Various commands set the bits in the Enable Registers. The Enable Registers and the commands used to set them are described below.

The Device Event Status Enable Register (DESER). Shown in Figure 3-3. This register controls which events of those shown are reported to the SESR and the Event Queue. The bits in the DESER correspond to those in the SESR, as was described earlier.

Use the DESE command to enable and disable the bits in the DESER. Use the DESE? query to read the DESER.

7	6	5	4	3	2	1	0
PON	URQ	CME	EXE	DDE	QYE	RQC	OPC

Figure 3-3: Device event status enable register (DESER)

The Event Status Enable Register (ESER). Shown in Figure 3-4. It controls which events of those shown are allowed to be summarized by the Event Status Bit (ESB) in the SBR.

Use the *ESE command to set the bits in the ESER. Use the *ESE? query to read it.

7	6	5	4	3	2	1	0
PON	URQ	CME	EXE	DDE	QYE	RQC	OPC

Figure 3-4: event status enable register (ESER)

The Service Request Enable Register (SRER). Shown in Figure 3-5. It controls which bits in the SBR generate a Service Request and are summarized by the Master Status Summary (MSS) bit.

Use the *SRE command to set the SRER. Use the *SRE? query to read it. The RQS bit remains set to one until either the Status Byte Register is read with a Serial Poll or the MSS bit changes back to a zero.

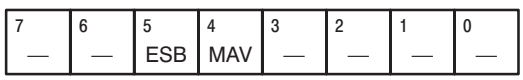


Figure 3-5: Service request enable register (SRER)

Queues

The status and event reporting system contains two queues, the Event Queue and the Output Queue. The Event Queue which is used when operating with either the GPIB and RS-232-C interface, while the Output Queue is used only when operating over the GPIB interface. (Instead of using an output queue, an output buffer buffers query-response messages for immediate transfer to the data transmission line for RS-232-C operation.)

Output Queue

The Output Queue is a FIFO (First In First Out) queue that hold response messages while until they are requested. When a message is put in the queue, the MAV bit of the Status Byte Register (SBR) is set.

The Output Queue empties each time the data generator receives a new command or query. Therefore the controller must read the output queue before it sends the next command or query command or it will lose responses to earlier queries. If a command or query command is given without taking it out, an error results and the Output Queue is emptied.

Event Queue

The Event Queue is a FIFO queue which can hold up to 20 data generator-generated events. When the number of events exceeds 20, the 20th event is replaced by the event code 350, “Queue overflow”.

To read out from the Event Queue, do the following steps.

1. Send *ESR? To read out the contents of SESR. When the contents of SESR are read out, SESR is cleared allowing you to take out events from the Event Queue.

2. Send one of the following queries:

   - ALLEv? To read out and returns all events made available by *ESR?. Returns both the event code and message text.
   - EVENT? To read out and return the oldest event of those made available by *ESR?. Returns only the event code.
   - EVMsg? To read out and return the oldest event of those made available by *ESR?. Returns both the event code and message text.

Reading the SESR erases any events that were made available by previous *ESR? reads, but that were not read from the Event Queue. Events that occur after an *ESR? read are put in the Event Queue but are not available until *ESR? is used again.

Processing Sequence

Figure 3-6 shows the status and event processing flow.

1. An event occurs, which causes the DESR to be checked. Based on the state of the DESR, the following actions occur:
   - If the control bit for that event is set in the DESER, the SESR bit that corresponds to this event becomes set to 1.
   - The set control bit lets the event be placed into the Event Queue. Placing the event in the Event Queue sets the MAV bit in the SBR to one.
   - If the control bit for that event is also set in the ESER, the ESB bit of SBR becomes set also.
2. When either bit of SBR has been set to 1 and the corresponding control bit of SRER is also set, the MSS bit of SBR becomes set and a service request is generated for use with GPIB interface operation.

As noted earlier, the RS-232-C interface does not use the output queue; therefore, the MAV bit would not become set in the sequence just described. Rather, response messages are sent to the output buffer for immediately transfer to the external controller on the output line. Message transfer is automatic and it is not necessary to use commands to retrieve these messages.

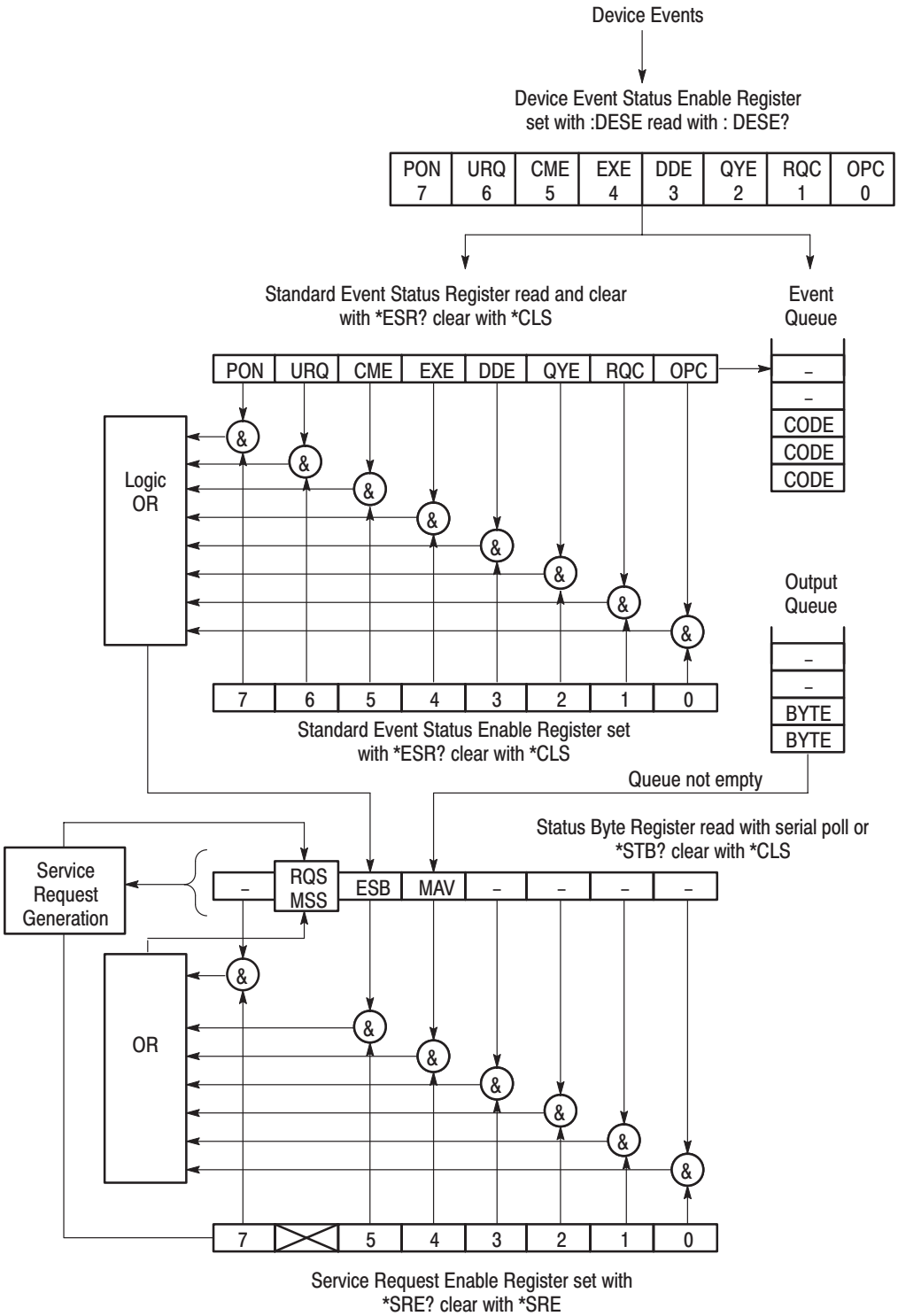


Figure 3-6: Status and event handling process overview

Messages

Tables 3-3 through 3-11 list the status and event messages used in the GPIB/ RS-232-C status and event reporting system. You use the *ESR? query to make the messages available for dequeuing; you use the :EVENT?, EVMsg?, and ALLEv? queries to dequeue and return the messages. The messages return as follows:

- The :EVENT? query command returns the event code only. When using these query commands, use the *ESR? query to make the events available for return.
- The EVMsg?, and ALLEv? queries return both the event code and event message in the following format:

  <event code>, "<event message ; secondary message>"

Most messages returned have both an event message, followed by a semicolon (;), and a second message which contains more detailed information. Although these secondary messages are not listed in this manual, you can use the EVMsg? and ALLEv? queries to display them.

Table 3-3 lists the definition of event codes.

Table 3-3: Definition of event codes

Event class	Event code ranges	Descriptions
No Events	0–1	No event nor status
Reserved	2–99	(unused)
Command Errors	100–199	Command errors
Execution Errors	200–299	Command execution errors
Device-Specific Errors	300–399	Internal device errors (Hardware errors)
Query Errors	400–499	System event and query errors
Execution Warnings	500–599	Execution warnings
Reserved	600–1999	(unused)
Extended Execution Errors	2000–2999	Device dependent command execution errors
Extended Device-Specific Errors	3000–3999	Device dependent device errors
Reserved	4000–	(unused)

Table 3-4 lists the message when the system has no events nor status to report. These have no associated SESR bit.

Table 3-4: Normal condition

Code	Description
0	No events to report — queue empty
1	No events to report — new events pending *ESR?

Table 3-5 lists the error messages generated due to improper command syntax. In this case, check that the command is properly formed and that it follows the syntax.

Table 3-5: Command errors (CME bit:5)

Code	Description
100	Command error
101	Invalid character
102	Syntax error
103	Invalid separator
104	Data type error
105	GET not allowed
106	Invalid program data separator
108	Parameter not allowed
109	Missing parameter
110	Command header error
111	Header separator error
112	Program mnemonic too long
113	Undefined header
114	Header suffix out of range
118	Query not allowed
120	Numeric data error
121	Invalid character in number
123	Exponent too large
124	Too many digits
128	Numeric data not allowed
130	Suffix error

Table 3-5: Command errors (CME bit:5) (Cont.)

Code	Description
131	Invalid suffix
134	Suffix too long
138	Suffix not allowed
140	Character data error
141	Invalid character data
144	Character data too long
148	Character data not allowed
150	String data error
151	Invalid string data
152	String data too long
158	String data not allowed
160	Block data error
161	Invalid block data
168	Block data not allowed
170	Expression error
171	Invalid expression
178	Expression data not allowed
180	Macro error
181	Invalid outside macro definition
183	Invalid inside macro definition
184	Macro parameter error

Table 3-6 lists the execution errors that are detected during execution of a command.

Table 3-6: Execution errors (EXE bit:4)

Code	Description
200	Execution error
201	Invalid while in local
202	Settings lost due to RTL
203	Command protected
210	Trigger error
211	Trigger ignored
212	Armed ignored
213	Init ignored
214	Trigger deadlock
215	ARM deadlock
220	Parameter error
221	Settings conflict
222	Data out of range
223	Too much data
224	Illegal parameter value
225	Parameter under range
226	Parameter over range
227	Parameter rounded
230	Data corrupt or stale
231	Data questionable
240	Hardware error
241	Hardware missing
250	Mass storage error
251	Missing mass storage
252	Missing media
253	Corrupt media
254	Media full
255	Directory full
256	File name not found
257	File name error

Table 3-6: Execution errors (EXE bit:4) (Cont.)

Code	Description
258	Media protected
260	Expression error
261	Math error in expression
262	Expression syntax error
263	Expression execution error
270	Macro error
271	Macro syntax
272	Macro execution error
273	Illegal macro label
274	Macro parameter error
275	Macro definition too long
276	Macro recursion error
277	Macro redefinition not allowed
278	Macro header not found
280	Program error
281	Cannot create program
282	Illegal program name
283	Illegal variable name
284	Program currently running
285	Program syntax error
286	Program run time error

Table 3-7 lists the internal errors that can occur during operation of the data generator. These errors may indicate that the data generator needs repair.

Table 3-7: Internal device errors (DDE bit:3)

Code	Description
300	Device-specific error
310	System error
311	Memory error
312	PUD memory lost
313	Calibration memory lost
314	Save/recall memory lost
315	Configuration memory lost
330	Self-test failed
350	Queue overflow (does not affect the DDE bit)

Table 3-8 lists the system event messages. These messages are generated whenever certain system conditions occur.

Table 3-8: System event and query errors

Code	Description
401	Power on
402	Operation complete
403	User request
404	Power fail
405	Request control
410	Query INTERRUPTED
420	Query UNTERMINATED
430	Query DEADLOCKED
440	Query UNTERMINATED after indefinite response

Table 3-9 lists warning messages that do not interrupt the flow of command execution. These messages warn you that you may get unexpected results.

Table 3-9: Warnings (EXE bit:4)

Code	Description
500	Execution warning

Table 3-10 lists status messages that are specific to the data generator. These messages appear when a operation starts, ends, or is in process. These messages have no associated SESR bit.

Table 3-10: Device-dependent command execution errors

Code	Description
2000	File error
2001	Directory not empty
2002	Too many files
2003	File locked
2004	File already exists
2005	File already opened
2006	Invalid file type
2007	File type mismatch
2008	Internal memory full
2009	Invalid file format
2010	Comment error
2012	Invalid data in comment string
2020	Pattern data error
2021	To much pattern data
2022	Pattern data byte count error
2023	Pattern data load error
2024	Internal pattern memory full
2025	Invalid pattern size
2026	Invalid pattern data
2030	Sequence error
2032	Too much sequence data
2033	Invalid sequence repeat count

Table 3-10: Device-dependent command execution errors (Cont.)

Code	Description
2034	Invalid sequence syntax
2035	Sequence load error
2036	Internal sequence memory full
2037	No sequence
2038	Invalid sequence number
2039	Sequence incomplete
2040	Data error
2041	Invalid data syntax
2042	Invalid data value
2050	Time error
2051	Invalid time syntax
2052	Invalid time value
2060	Invalid group name
2061	Group name is empty
2062	Same name already exists
2063	Too much group
2064	Group name not found
2065	Group number is not found
2066	Invalid group data
2067	Invalid group syntax
2070	Invalid block position
2071	To much block
2072	Block already exists
2073	Block is not found
2074	Illegal block name
2075	Illegal block size
2076	Block name already exists
2077	Block is not defined
2078	Too much block data
2079	Invalid block syntax
2080	Import error
2081	Code table syntax error
2082	Too much table data

Table 3-10: Device-dependent command execution errors (Cont.)

Code	Description
2100	Hardcopy error
2101	Hardcopy busy
2102	Hardcopy timeout error
2200	Message error

Table 3-11 lists device error messages that are specific to the device.

Table 3-11: Extended device specific errors

Code	Description
3001	RS-232-C input buffer overflow

Programming Examples

Programming Examples

This section presents sample programs that show specific examples of techniques for controlling the DG2020A over a GPIB interface. The sample programs are stored on the floppy disk (labeled Software Library) included with the DG2020A. Since this manual does not include a listing of these programs you may want to reference those files while reading this manual. The majority of these programs are provided in C versions, and thus can be used in the Microsoft Visual C++ environment. Some of the programs are also provided in Quick BASIC or LabVIEW versions.

Overview of the Sample Programs

getbit Reads out bit patterns from the DG2020A data memory in bit units and displays that data on the screen. Since standard output is used for data display, the output can be redirected to a file for storage. This is an example of a program that reads out bit pattern data. This program is provided in both C and BASIC versions.

putbit Transfers bit units pattern data stored in a file by the `getbit` program described above to DG2020A data memory. This is an example of a program that writes bit pattern data.

getword Reads out bit patterns from the DG2020A memory in word units and displays that data on the screen. Since standard output is used for data display, the output can be redirected to a file for storage. This is an example of a program that reads out bit pattern data.

putword Transfers word units pattern data stored in a file by the `getword` program described above to DG2020A data memory. This is an example of a program that writes bit pattern data.

putblk Sets up a block definition section in the DG2020A data memory using data from a file prepared in advance. This program presents an example of the use of the block definition command.

putgrp Sets up a group definition section in the DG2020A data memory using data from a file prepared in advance. This program presents an example of the use of the group definition command.

putseq Sets up a sequence definition section in the DG2020A data memory using data from a file prepared in advance. This program presents an example of the use of the sequence definition command.

putsub Sets up a subsequence definition section in the DG2020A data memory using data from a file prepared in advance. This program presents an example of the use of the subsequence definition command.

intcom Supports interactive GPIB command and message exchange with the DG2020A. This program allows the operator to easily confirm the actual operation of the GPIB commands. This program also serves as an example of the communications protocols used between the DG2020A and the GPIB interface.

Required Execution Environment

These programs run on an IBM PC/AT compatible personal computer that has a National Instruments, Inc. GPIB interface installed. The C sample programs run in the MS-DOS prompt window and require a Windows 95 operating system, and the National Instruments, Inc. GPIB95 driver software. The BASIC sample programs require a MS-DOS, version 5.0 or later, and the National Instruments, Inc. AT-GPIB driver software. In addition, Microsoft Visual C++, Quick BASIC 4.5 or LabVIEW is required to compile and run the sample programs. You should provide an environment that meets these conditions and install the respective software according to their manuals.

Floppy Disk Files

The floppy disk contains the following files. The *README.TXT* file stored in the floppy disk also provides you the detailed information.

MSVC Directory

Filename	Description
getbit.c	The `getbit` C source file
putbit.c	The `putbit` C source file
getword.c	The `getword` C source file
putword.c	The `ptword` C source file
putblk.c	The `putblk` C source file

Filename	Description
putgrp.c	The `putgrp` C source file
putseq.c	The `putseq` C source file
putsub.c	The `putsub` C source file
intcom.c	The `intcom` C source file
gpiblib.c	The GPIB library used with the above programs

QBASIC Directory

Filename	Description
getbit.bas	The `getbit` BASIC source file
putbit.bas	The `putbit` BASIC source file
makeexe.bat	The batch file used to compile these programs with Quick BASIC

DATA Directory

Each of the sample programs uses its own unique input or output format. Several sample files with examples of those formats are stored on the floppy disk in this directory. These files are all ASCII text files and can be viewed and edited with a text editor.

Filename	Description
patbit.dat	Output file for the `getbit` sample program, or input file for `putbit`.
patword.dat	Output file for the `getword` sample program, or input file for `putword`.
blkdef.dat	Input file for the `putblk` sample program
grpdef.dat	Input file for the `putgrp` sample program
seqdef.dat	Input file for the `putseq` sample program
subdef.dat	Input file for the `putsub` sample program
podassig.cmd	GPIB command file that performs output pod bit allocation
poddelay.cmd	GPIB command file that sets the output pod delay times
podinhib.cmd	GPIB command file for output pod high–impedance control
podlevel.cmd	GPIB command file that sets the output pod output voltage levels

Installing and Compiling the Programs

Executable programs must be created by compiling the source files provided on the floppy disk. The programs are compiled after copying the source files to the hard disk. To prevent any possibility of damaging the original during these operations, it is recommended that you first make a copy of the floppy disk, store the original in a safe place, and use the copy for the following procedures.

Making Copy

Create a directory on the hard disk, in which you install all of the sample programs. This procedure assumes that the hard drive is drive c: and the floppy drive is drive a:.

In the DOS prompt window, type the following commands.

```
mkdir  c:\DGSAMPLE.20A
```

Copy the folders and files in the floppy disk with the directory tree structure kept intact. You can simply do this operation by drag and drop method in the Windows 95 Explorer window.

1. Click **Drive A:** (floppy) icon to display the files in the floppy disk.
2. Select **Select All** from the File menu. (Alternatively, you can make this operation by pressing Control + A keys on the keyboard.)
3. Drag the selected files in the floppy to the new directory created in step 1.

Compiling the source codes

Do the following procedures to compile the sample program source code. The procedures are different depending on the type of source codes: C or BASIC.

In case of C programs.

1. You need the National Instrument GPIB library file to use the sample program source codes. The library is assumed to be resided in the following default path.

   ```
   c:\GPIB95\LANGINT\C
   ```

   When you have a different environment, change the default path setting defined in the project file or to make the directory as required.

2. To compile the C sample source codes, you need Microsoft Developer Studio. Select **File** from the Open Workspace menu in the Microsoft Developer.

   When the compiler environment has been installed properly, you can just click the project workspace file in the Explorer window to compile the sample program source code.

For example,

- Double-click the C:\DGSAMPLE.20A\GETBIT\GETBIT.MDP file icon in the Explorer window to open the `getbit` sample program.

  The Developer Studio will automatically be invoked and the project workspace will be opened.

- Select **Build** from the Build menu to compile the sample program source code.

3. Execute the compiled programs in the MS-DOS prompt window. Type the following commands, for example.

```
cd  c:\DGSAMPLE.20A\GETBIT
.\GETBIT <parameters>
```

In case of BASIC programs.

1. In the MS-DOS prompt window, move to the directory in which you have copied the sample BASIC files from the floppy disk in the procedures described in *Making Copy* on page 4-4. Select a name that does not duplicate an existing name in the file system.

```
cd  c:\DGSAMPLE.20A\QBASIC
```

2. Next, copy the necessary files associated with the GPIB drivers. This procedure assumes that the National Instruments drivers are installed in the `c:\at-gpib` directory.

```
copy  c:\at-gpib\qbasic\qbdecl.bas  .
copy  c:\at-gpib\qbasic\qbib.obj  .
```

3. Compile the sample programs using the batch file

```
makeexe.bat
```

4. If following the above procedures did not result in the compilation completing correctly, check that there is adequate free space on the hard disk and that the compiler is installed correctly. In particular, check that the path setting is correct.

Installing the Sample Data

Create an appropriate directory on the hard disk. Select a name that does not duplicate an existing file or directory name in the file system. (This procedure uses the directory "`sample_d`".) This procedure assumes that the hard drive is drive c:, that the floppy drive is drive a:, and that the current directory is an appropriate directory on the hard disk.

```
mkdir  sample_d
cd  sample_d
copy  a:\data\*.*  .
```

Sample Program Functions and Usage

This section describes the functions of the sample programs and their use. Words set in italics are abstract parameters that must be replaced with actual character strings.

Getbit

This program reads out bit pattern data from DG2020A data memory in bit units and displays that data on the screen. The bit number (0 to 35) to be read out is specified as command line arguments. If multiple arguments are specified separated by spaces, the bit data is displayed in the specified order. This command has the following syntax:

getbit *bit_number* [*bit_number* ...]

Bit data is read out from the whole range of memory data set up in the DG2020A, that is, data is read out from address 0 to the maximum address.

The program displays the memory size, the starting address (always 0) and the bit number parameters and then the bit data for those parameters. The parameters are displayed starting with a number sign (#) character, each on its own line. The bit data is expressed as a sequence of the characters 0 and 1 representing those bit values. The listing below shows the output from reading out bits 3 and 2 when the memory size is 64.

```
#size 64
#start 0
#bit 3
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,
0,0,0,0,1,0,0,0,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,
#bit 2
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
```

Since the result is displayed on standard output, the data can be saved by redirecting the output to a file. This file can be used as an input to the `putbit` program described below.

Putbit

This program sends bit pattern data to the DG2020A data memory in bit units. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putbit *filename*

The contents of the input file must express the data length, start address, and bit number parameters, as well as the bit data itself in ASCII text. Each parameter must appear on a separate line and start with a number sign (#) character. Bit data is expressed as a sequence of the characters 1 and 0 representing the bit values. This format is the same format as that produced by redirecting the output of the `getbit` program. The file `patbit.dat` is a sample data file in this format.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct.

Getword

This program reads out bit pattern data from DG2020A data memory in word units and displays that data on the screen. The start address for the read and the number of words are specified as command line arguments in decimal, separated by a space. This command has the following syntax:

getword *address length*

The program displays the memory size and the starting address parameters and then the bit pattern data for those parameters. The parameters are displayed starting with a number sign (#) character, each on its own line. The bit pattern data is displayed as 36–bit words of 5 bytes each, with each byte displayed in hexadecimal. The bytes of each word are displayed on a single line starting with the most significant byte. Only the lower 4 bits are used in the most significant byte. The listing below shows the output for reading out the words from addresses 0 to 64.

```
#size 64
#start 0
00,00,00,00,00,
00,00,00,00,08,
08,00,00,00,00,
00,00,00,00,0c,
. . . . . .
00,00,00,00,00,
00,00,00,00,00,
00,00,00,00,00,
```

Since the result is displayed on standard output, the data can be saved by redirecting the output to a file. This file can be used as an input to the `putword` program described below.

Putword

This program sends bit pattern data to the DG2020A data memory in word units. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putword *filename*

The contents of the input file must express the data length and start address parameters, as well as the bit pattern data itself in ASCII text. Each parameter must appear on a separate line and start with a number sign (#) character. The bit pattern data is expressed as 36–bit words of 5 bytes each starting with the most significant byte, with each byte expressed in hexadecimal. A newline character is required for each data word. This format is the same format as that produced by redirecting the output of the `getword` program. See the `patword.dat` sample data file.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct.

Putgrp

This program sends group definition data to the DG2020A data memory. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putgrp *filename*

The contents of the input file must express the group name, the group's highest and lowest bit numbers in ASCII with one group per line. This format corresponds to the parameter block supplied to the `DATA:GROUP:DEFINE` command with the delimiter codes replaced by the newline code used in normal text files. See the `grpdef.dat` sample data file.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct. In particular, note that commas are used to delimit the parameters in this input file and that spaces may not be inserted.

Putblk

This program sends block definition data to the DG2020A data memory. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putblk *filename*

The contents of the input file must express the block name and the block starting address in ASCII with one block per line. This format corresponds to the parameter block supplied to the `DATA:BLOCK:DEFINE` command with the delimiter codes replaced by the newline code used in normal text files. See the `blkdef.dat` sample data file.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct. In particular, note that commas are used to delimit the parameters in this input file and that spaces may not be inserted.

Putseq This program sends sequence definition data to the DG2020A data memory. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putseq *filename*

The contents of the input file must express the block name, the repeat count, the line number of the event jump destination, the trigger wait on/off setting, the event jump on/off setting and the infinite loop on/off setting, in ASCII with one step per line. This format corresponds to the parameter block supplied to the `DATA:SEQUENCE:DEFINE` command with the delimiter codes replaced by the newline code used in normal text files. The first line in this file corresponds to sequence line number 0, and each following line to the sequence line number incremented by 1. See the `seqdef.dat` sample data file.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct. In particular, note that commas are used to delimit the parameters in this input file and that spaces may not be inserted.

Putsub This program sends subsequence definition data to the DG2020A data memory. Data input is from a file in a unique format. This command has the following syntax, with the input data file name being specified as a parameter.

putsub *filename*

The contents of the input file must express the block name and the repeat count, in ASCII with one step per line. This format corresponds to the parameter block supplied to the `DATA:SUBSEQUENCE:DEFINE` command with the delimiter codes replaced by the newline code used in normal text files. The first line in this file corresponds to subsequence line number 0, and each following line to the subsequence line number incremented by 1. See the `subdef.dat` sample data file.

Input format checking has been dispensed with to make this sample program easier to understand. This program may not operate correctly if the format of the input file is not correct. In particular, note that commas are used to delimit the parameters in this input file and that spaces may not be inserted.

Intcom This program implements interactive communication between an external controller and the DG2020A. That is, it transmits GPIB commands entered from the keyboard to the DG2020A and displays messages returned from the

DG2020A on the screen. The command has the following syntax, in which the argument specifies the device name registered in the GPIB driver system. The device dev1 is used as the default if the argument is omitted.

intcom [*device*]

When this program is started it displays its own prompt and waits for input. When a command is entered, it executes the processing for that command and then returns to the command wait state. It iterates this sequence until the termination command is entered. The prompt indicates the GPIB device name, as shown below.

dev1>>

The DG2020A GPIB commands, the program's internal (built–in) commands, and redirection commands can be used as intcom commands. These commands are described in detail below.

- GPIB commands

  All commands and queries defined in this manual may be used. If a question mark character (?) appears in the input character string, the command is interpreted as a query command. The program waits for the DG2020A response, automatically extracts that response, and displays it on the screen. If an error occurs, the program extracts the event code and event message from the event queue and displays them on the screen.

- Built–in commands

  Intcom supports the following built–in commands:

exec *filename*	Reads in commands from a file one line at a time and executes them through the end of the file. This allows a sequence of commands to be prepared in advance and used as a batch file. This differs from standard input redirection described below in that the contents of the file are first interpreted by this program's command processing routine. The result is that while the built–in commands and the redirection commands can be used, GPIB commands that include binary data and newline codes cannot be used.
help	Displays command descriptions on the screen.
resets	Returns the registers used by the event and status reporting system to the standard state set up by this program. If the set values of the GPIB commands registers such as DESE and *ESE are changed, this command should be used as soon as possible to return their values to the standard values.

view *filename*	Outputs the contents of the file specified by filename to standard output, i.e., displays the file on the screen.
!!	Executes the immediately preceding command once again.

- Redirection commands

  The following commands can be used to switch standard input or standard output to a file and thus realize communications between the DG2020A and a file or files.

<filename	Sends the contents of the file specified by filename to the DG2020A directly without modification. This allows a sequence of commands to be prepared in advance and used as a batch file. In particular, only this command can be used to send GPIB commands that include binary data blocks to the DG2020A.
>filename	Intercepts the data output to standard output and outputs it to the file specified by filename. If the file already exists, it is overwritten. If the file does not exist, a new file is created.
>>filename	Intercepts the data output to standard output and outputs it to the file specified by filename in the same way as the '>' command. However, if the file already exists it does not overwrite the file but rather appends the new data at the end of the file.

Appendices

Appendix A: Character Charts

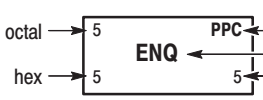
Table A–1: DG2020A character set

	0	1	2	3	4	5	6	7
0	NUL 0	16	space 32	0 48	@ 64	P 80	‘ 96	p 112
1	1	Ω 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	Δ 18	” 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	µ 22	& 38	6 54	F 70	V 86	f 102	v 118
7	‘ 7	23	’ 39	7 55	G 71	W 87	g 103	w 119
8	8	— 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	— 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	LF 10	∞ 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	± 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	CR 13	≠ 29	— 45	= 61	M 77	] 93	m 109	} 125
E	14	~ 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	● 15	31	/ 47	? 63	O 79	_ 95	o 111	rubout 127

Table A–2: ASCII & GPIB code chart

B7 B6 B5 BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	20 DLE 10 16	40 LA0 SP 20 32	60 LA16 0 30 48	100 TA0 @ 40 64	120 TA16 P 50 80	140 SA0 ` 60 96	160 SA16 p 70 112
0 0 0 1	1 GTL SOH 1 1	21 LL0 DC1 11 17	41 LA1 ! 21 33	61 LA17 1 31 49	101 TA1 A 41 65	121 TA17 Q 51 81	141 SA1 a 61 97	161 SA17 q 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 LA2 " 22 34	62 LA18 2 32 50	102 TA2 B 42 66	122 TA18 R 52 82	142 SA2 b 62 98	162 SA18 r 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 LA3 # 23 35	63 LA19 3 33 51	103 TA3 C 43 67	123 TA19 S 53 83	143 SA3 c 63 99	163 SA19 s 73 115
0 1 0 0	4 SDC EOT 4 4	24 DCL DC4 14 20	44 LA4 $ 24 36	64 LA20 4 34 52	104 TA4 D 44 68	124 TA20 T 54 84	144 SA4 d 64 100	164 SA20 t 74 116
0 1 0 1	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 LA5 % 25 37	65 LA21 5 35 53	105 TA5 E 45 69	125 TA21 U 55 85	145 SA5 e 65 101	165 SA21 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 LA6 & 26 38	66 LA22 6 36 54	106 TA6 F 46 70	126 TA22 V 56 86	146 SA6 f 66 102	166 SA22 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 LA7 ' 27 39	67 LA23 7 37 55	107 TA7 G 47 71	127 TA23 W 57 87	147 SA7 g 67 103	167 SA23 w 77 119
1 0 0 0	10 GET BS 8 8	30 SPE CAN 18 24	50 LA8 (28 40	70 LA24 8 38 56	110 TA8 H 48 72	130 TA24 X 58 88	150 SA8 h 68 104	170 SA24 x 78 120
1 0 0 1	11 TCT HT 9 9	31 SPD EM 19 25	51 LA9) 29 41	71 LA25 9 39 57	111 TA9 I 49 73	131 TA25 Y 59 89	151 SA9 i 69 105	171 SA25 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 LA10 * 2A 42	72 LA26 : 3A 58	112 TA10 J 4A 74	132 TA26 Z 5A 90	152 SA10 j 6A 106	172 SA26 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 LA11 + 2B 43	73 LA27 ; 3B 59	113 TA11 K 4B 75	133 TA27 [5B 91	153 SA11 k 6B 107	173 SA27 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 LA12 , 2C 44	74 LA28 < 3C 60	114 TA12 L 4C 76	134 TA28 \ 5C 92	154 SA12 l 6C 108	174 SA28 ¦ 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 LA13 – 2D 45	75 LA29 = 3D 61	115 TA13 M 4D 77	135 TA29] 5D 93	155 SA13 m 6D 109	175 SA29 } 7D 125
1 1 1 0	16 SO E 14	36 RS 1E 30	56 LA14 . 2E 46	76 LA30 > 3E 62	116 TA14 N 4E 78	136 TA30 ^ 5E 94	156 SA14 n 6E 110	176 SA30 ~ 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 LA15 / 2F 47	77 UNL ? 3F 63	117 TA15 O 4F 79	137 UNT – 5F 95	157 SA15 o 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY



GPIB code (with ATN asserted)
ASCII character
decimal

Tektronix
REF: ANSI STD X3.4-1977
IEEE STD 488.1-1987
ISO STD 646-2973

Appendix B: Reserved Words

The words in the following list are reserved words for use with the DG2020A Data Generator.

ABORt
ABSTouch
ADD
ALL
ALLEv
ASSIGn
BIT
BLOCk
BRIGhtness
CATalog
CDIRectory
CH<n>
CLEAr
CLEar
CLOCk
CLS
COPY
DATA
DATE
DEBug
DEFine
DELAy
DELete
DESE
DIAGnostic
DIMmer
DISPlay
ELEVel
ENABle
ESE
ESR
EVENT
EVJ
EVJTO
EVMsg
EVQty
EXTernal
FACTory
FORMat
FREE
FREQuency
GROUp
HCOPy
HEADer
HIGH
ID
IDN
ILEVel
IMMediate
IMPedance
INHibit
INITialize
INTernal
LEVel
LOAD
LOCK
LOCk
LOOP
LOW
MDIRectory
MENU
MMEMory
MODE
MSIZe
NAME
OPC
OPT
ORDer
OSCillator
OUTPut
PATTern
PLLlock
POD<s>
PORT
PPAUse
PSC
RELEase
REName
REPeat
RESUlt
RST
RUNNing
SAVE
SECurity
SELect
SEQuence
SIZe
SLOpe
SNOop
SOURce
SRE
STARt
STATe
STB
STOP
SUBSequence
SYSTem
TEXT
TIME
TRG
TRIGger
TST
TWAIT
TYPE
UNLock
UPDate
UPTime
VERBose
WAI
WINDow
WORD

Appendix C: Interface Specification

This appendix lists and describes the GPIB functions and messages that the DG2020A Data Generator implements.

Interface Functions

Table C–1 shows which GPIB interface functions are implemented in this instrument. Following the table is a brief description of each function.

Table C–1: GPIB interface function implementation

Interface function	Implemented subset	Capability
Acceptor Handshake (AH)	AH1	Complete
Source Handshake (SH)	SH1	Complete
Listener (L)	L4	Basic Listener Unaddress if my talk address (MTA) No talk only mode
Talker (T)	T5	Basic Talker, Serial Poll Unaddress if my-listen-address (MLA)
Device Clear (DC)	DC1	Complete
Remote/Local (RL)	RL1	Complete
Service Request (SR)	SR1	Complete
Parallel Poll (PP)	PP0	None
Device Trigger (DT)	DT1	Complete
Controller (C)	C0	None
Electrical Interface	E2	Three-state driver

- Accepter Handshake (AH). Allows a listening device to help coordinate the the proper reception of data. The AH function holds off initiation or termination of a data transfer until the listening device is ready to receive the next data byte.
- Source Handshake (SH). Allows a talking device to help coordinate the proper transfer of data. The SH function controls the initiation and termination of the transfer of data bytes.

- Listener (L). Allows a device to receive device-dependent data over the interface. This capability exists only when the device is addressed to listen. This function uses a one-byte address.
- Talker (T). Allows a device to send device-dependent data over the interface. This capability exists only when the device is addressed to talk. The function uses a one-byte address.
- Device Clear (DC). Allows a device to be cleared or initialized, either individually or as part of a group of devices.
- Remote/Local (RL). Allows a device to select between two sources for operating control. This function determines whether input information from the front panel controls (local) or GPIB commands (remote) control the data generator.
- Service Request (SR). Allows a device to request service from the controller.
- Controller (C). Allows a device with the capability to send the device address, universal commands, and addressed commands to other device over the interface to do so.
- Electrical Interface (E) Identifies the type of the electrical interface. The notation E1 indicates the electrical interface uses open collector drivers, while E2 indicates the electrical interface uses three-state drivers.

Interface Messages

Table C–2 lists the GPIB Universal and Addressed commands that the DG2020A Data Generator implements. A brief description of each function follows the table.

Table C–2: GPIB interface messages

Interface message	Implemented
Device Clear (DC)	Yes
Local Lockout (LLO)	Yes
Serial Poll Disable (SPD)	Yes
Serial Poll Enable (SPE)	Yes
Parallel Poll Unconfigure (PPU)	No
Go To Local (GTL)	Yes
Selected Device Clear (SDC)	Yes
Group Execute Trigger (GET)	Yes

Table C–2: GPIB interface messages (Cont.)

Interface message	Implemented
Take Control (TCT)	No
Parallel Poll Configure (PPC)	No

- Device Clear (DCL). Clears (initializes) all devices on the bus that have a device clear function, whether the controller has addressed them or not.
- Local Lockout (LLO). Disables the return to local function.
- Serial Poll Enable (SPE). Puts all devices on the bus, that have a service request function, into the serial poll enabled state. In this state, each device sends the controller its status byte, instead of the its normal output, after the device receives its talk address on the data lines. This function may be used to determine which device sent a service request.
- Serial Poll Disable (SPD). Changes all devices on the bus from the serial poll state to the normal operating state.
- Go To Local (GTL). Causes the listen-addressed device to switch from remote to local (front-panel) control.
- Select Device Clear (SDC). Clears or initializes all listen-addressed devices.
- Group Execute Trigger (GET). Triggers all applicable devices and causes them to initiate their programmed actions.
- Take Control (TCT).Allows controller in charge to pass control of the bus to another controller on the bus.
- Parallel Poll Configure (PPC). Causes the listen-addressed device to respond to the secondary commands Parallel Poll Enable (PPE) and Parallel Poll Disable (PPD), which are placed on the bus following the PPC command. PPE enables a device with parallel poll capability to respond on a particular data line. PPD disables the device from responding to the parallel poll.

Appendix D: Factory Initialization Settings

The following table lists the commands affected by a factory initialization and their factory initialization settings.

Table D–1: Factory initialized settings

Header	Default settings
DATA commands	
DATA:MSIZe	1000
DIAGNOSTIC commands	
DIAG:SELect	ALL
DISPLAY commands	
DISPlay:BRIGhtness	0.7
DISPlay:CLOCk	0
DISPlay:DIMmer	0
DISPlay:ENABle	1
DISPlay:MENU[:NAME]	EDIT
DISPlay:MENU:STATe	1
HARDCOPY commands	
HCOPy:FORMat	BMP
HCOPy:PORT	DISK
MEMORY commands	
MMEMory:CATalog:ORDer	NAME1
MODE commands	
MODE:STATe	REPEAT
MODE:UPDate	AUTO
TRIGger:IMPedance	HIGH
TRIGger:LEVel	1.4
TRIGger:SLOPe	POSITIVE

Table D–1: Factory initialized settings (Cont.)

Header	Default settings
OUTPUT commands	
OUTPut:ELEVel	1.4
OUTPut:ILEVel	1.4
OUTPut:POD<s>:CH<n>:DELAy	0.0
OUTPut:POD<s>:CH<n>:HIGH	3.0
OUTPut:POD<s>:CH<n>:INHibit	0
OUTPut:POD<s>:CH<n>:LOW	0.0
SOURCE commands	
SOURce:OSCillator:EXTernal:FREQuency	1.0E+8
SOURce:OSCillator[:INTernal]:FRE-Quency	1.0E+8
SOURce:OSCillator[:INTernal]:PLLlock	INTERNAL
SOURce:OSCillator:SOURce	1
SOURce:POD<s>:EVENT:ENABle	1
STATUS & EVENT commands	
DESE	256
*ESE	0
*PSC	1
*SRE	0
SYSTEM commands	
DEBug:SNOop:DELAy:TIME	0.2
DEBug:SNOop:STATe	0
HEADer	1
LOCk	NONE
SYSTem:PPAUse	1
SYSTem:SECurity:STATe	0
VERBose	1

Glossary & Index

Glossary

ASCII
Acronym for the American Standard Code for Information Interchange. Controllers transmit commands to the instrument using ASCII character encoding.

Address
A 7-bit code that identifies an instrument on the communication bus. The instrument must have a unique address for the controller to recognize and transmit commands to it.

BNF (Backus-Naur Form)
A standard notation system for command syntax diagrams. The syntax diagrams in this manual use BNF notation.

Controller
A computer or other device that sends commands to and accepts responses from the digitizing oscilloscope.

EOI
A mnemonic referring to the control line "End or Identify" on the GPIB interface bus. One of the two possible end-of-message terminators.

EOM
A generic acronym referring to the end-of-message terminator. The end-of-message terminator can be either an EOI or the ASCII code for line feed (LF).

GPIB
Acronym for General Purpose Interface Bus, the common name for the communications interface system defined in IEEE Std 488.

IEEE
Acronym for the Institute for Electrical and Electronic Engineers.

QuickC
A computer language (distributed by Microsoft) that is based on C.

Index

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W