Programmer Manual

Tektronix

DTG5000 Series
Data Timing Generator

071-1283-01

This document supports program version 1.0 of
DTG5000 Series.

www.tektronix.com

Tektronix Japan, Ltd., 5-9-31 Kitashinagawa, Shinagawa-ku, Tokyo 141-0001 Japan
Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077

WARRANTY

Tektronix warrants that this product will be free from defects in materials and workmanship for a period of one (1) year from the date of shipment. If any such product proves defective during this warranty period, Tektronix, at its option, either will repair the defective product without charge for parts and labor, or will provide a replacement in exchange for the defective product.

In order to obtain service under this warranty, Customer must notify Tektronix of the defect before the expiration of the warranty period and make suitable arrangements for the performance of service. Customer shall be responsible for packaging and shipping the defective product to the service center designated by Tektronix, with shipping charges prepaid. Tektronix shall pay for the return of the product to Customer if the shipment is to a location within the country in which the Tektronix service center is located. Customer shall be responsible for paying all shipping charges, duties, taxes, and any other charges for products returned to any other locations.

This warranty shall not apply to any defect, failure or damage caused by improper use or improper or inadequate maintenance and care. Tektronix shall not be obligated to furnish service under this warranty a) to repair damage resulting from attempts by personnel other than Tektronix representatives to install, repair or service the product; b) to repair damage resulting from improper use or connection to incompatible equipment; or c) to service a product that has been modified or integrated with other products when the effect of such modification or integration increases the time or difficulty of servicing the product.

THIS WARRANTY IS GIVEN BY TEKTRONIX WITH RESPECT TO THIS PRODUCT IN LIEU OF ANY OTHER WARRANTIES, EXPRESSED OR IMPLIED. TEKTRONIX AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TEKTRONIX' RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. TEKTRONIX AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER TEKTRONIX OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Table of Contents

Status and Events

List of Tables

List of Figures

Preface

This is the programmer manual for the DTG5000 Series Data Timing Generators. This manual provides information necessary for operating the instrument over the General Purpose Interface Bus (GPIB) interface.

This manual provides the following information:

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

Related Manuals

Other documentation for the data timing generator includes:

- The DTG5000 User Manual Vol.1(071-1281-xx) describes the how to use the instrument.
- The DTG5000 User Manual Vol.2(071-1282-xx) describes the operation of the instrument.

Getting Started

The DTG5000 Series Data Timing Generator has GPIB interface capability. You can write computer programs that remotely set the front panel controls.

To help you get started with programming the data timing generator, this section includes the following subsections:

- *Manual Overview* – summarizes the type of programming information contained in each major section in this manual.
- *Setting Up Remote Communications Using GPIB* – describes how to connect the data timing generator to a controller through the GPIB interface.

Manual Overview

A summary of the information provided in each major section of this manual follows:

Syntax and Commands

The *Command Syntax* subsection, which begins on page 2–1, describes the structure and content of the messages your program sends to the data timing generator. You can use the Standard Commands for Programmable Instruments (SCPI) and IEEE 488.2 Common Commands. Figure 1–1 is an example of the syntax and command parts diagrams used in the *Command Syntax* subsection.

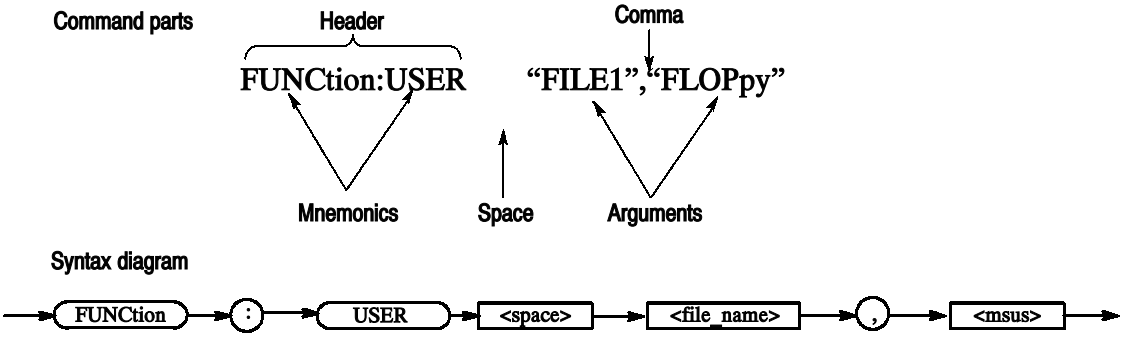


Figure 1–1: Common message elements

The *Command Syntax* subsection also describes the result of each command, and provides examples of how you might use it. The *Command Groups* subsection, which begins on page 2–13, provides a command list by functional area. The *Command Descriptions* subsection, which begins on page 2–21, arranges commands alphabetically.

Status and Events Reporting

The program may request information from the data timing generator. The data timing generator provides information in the form of status and error messages. Figure 1–2 on page 1–2 illustrates the basic operation of this system.

The *Status and Events Reporting* subsection, which begins on page 3–1, describes how to use the status reporting functions that conform to SCPI and IEEE–488.2 in your programs.

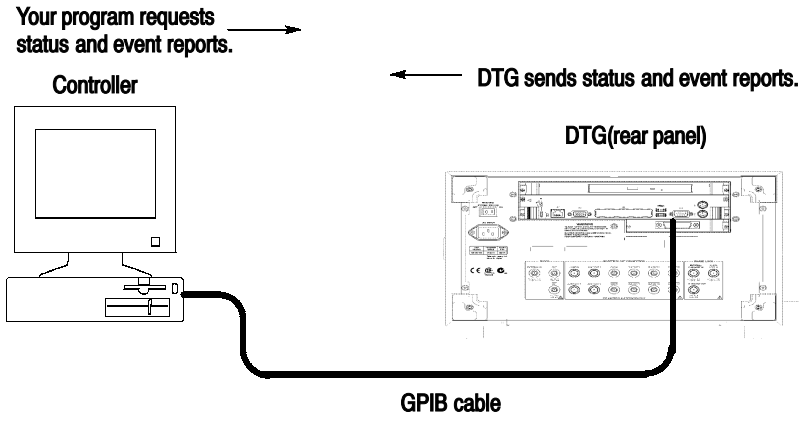


Figure 1–2: Basic operation of status and events reporting

Programming Examples

The *Programming Examples* section, which begins on page 4–1, provides a sample data timing generator program.

Setting Up Remote Communications Using GPIB

For remote operations, the instrument must be connected to the controller.

The data timing generator has a 24-pin GPIB connector on its rear panel, as shown in Figure 1-3. This connector has a D-type shell and conforms to IEEE Std 488.2-1992.

Attach an IEEE Std 488.2-1992 GPIB cable (Tektronix Part Number 012-0991-xx) to the GPIB connector.

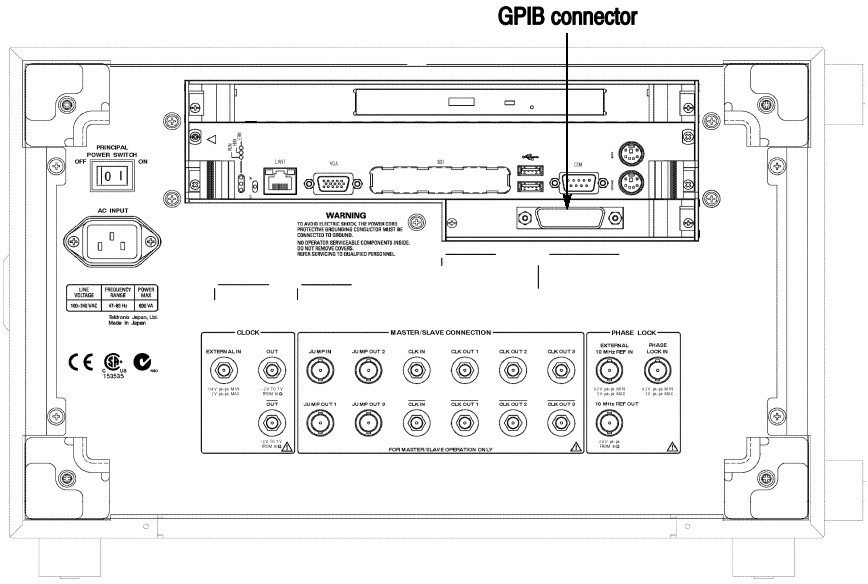


Figure 1-3: GPIB connector location

GPIB Requirements

Follow these rules when you use your data timing generator with a GPIB network:

- Assign a unique device address to each device on the bus. Two devices can not share the same device address.
- Do not connect more than 15 devices to one bus.
- Connect one device for every 2 meters (6 feet) of cable used.
- Do not use more than 20 meters (65 feet) of cable to connect devices to a bus.
- While using the network, turn on at least two-thirds of the devices on the network.
- Connect the devices on the network in a star or linear configuration, as shown in Figure 1–4. Do not use loop or parallel configurations.

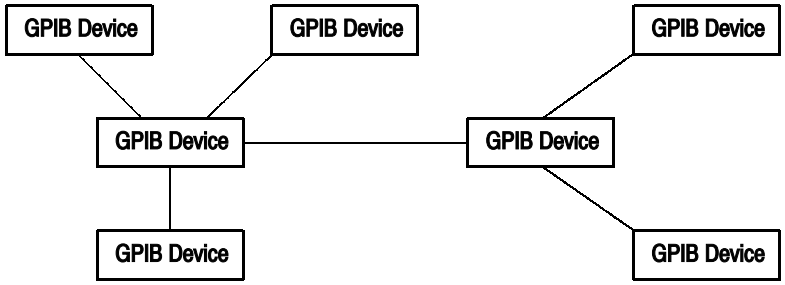


Figure 1–4: Typical GPIB network configurations

Command Syntax

This section contains general information about command structure and syntax usage. You should familiarize yourself with this material before using the data timing generator command descriptions.

This manual describes commands and queries using Backus-Naur Form (BNF) notation. Table 2–1 defines standard BNF symbols.

Table 2–1: BNF symbols and meanings

Symbol	Meaning
< >	Defined element
::=	Is defined as
|	Exclusive OR
{ }	Group; one element is required
[ ]	Optional; can be omitted
. . .	Previous element(s) may be repeated
( )	Comment

SCPI Commands and Queries

The data timing generator uses a command language based on the SCPI standard. The SCPI (Standard Commands for Programmable Instruments) standard was created by a consortium to provide guidelines for remote programming of instruments. These guidelines provide a consistent programming environment for instrument control and data transfer. This environment uses defined programming messages, instrument responses and data formats that operate across all SCPI instruments, regardless of manufacturer.

The SCPI language is based on a hierarchical or tree structure that represents a subsystem (see Figure 2–1). The top level of the tree is the root node; it is followed by one or more lower-level nodes.

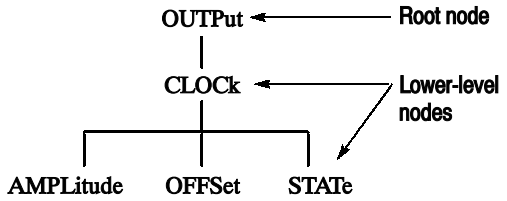


Figure 2–1: Example of SCPI subsystem hierarchy tree

You can create commands and queries from these subsystem hierarchy trees. Commands specify actions for the instrument to perform. Queries return measurement data and information about parameter settings.

Creating Commands

SCPI commands are created by stringing together the nodes of a subsystem hierarchy and separating each node by a colon.

In Figure 2-1 on page 2-2, OUTPut is the root node and CLOCk, AMPLitude, OFFSet, and STATe are lower-level nodes. To create an SCPI command, start with the root node OUTPut and move down the tree structure adding nodes until you reach the end of a branch. Most commands and some queries have parameters; you must include a value for these parameters. The command descriptions, which begin on page 2-21, list the valid values for all parameters.

For example, **OUTPut:CLOCk:AMPLitude 2.0** is a valid SCPI command created from the hierarchy tree in Figure 2-1 on page 2-2.

Creating Queries

To create a query, start at the root node of a tree structure, move down to the end of a branch, and add a question mark. **OUTPut:CLOCk:AMPLitude?** is an example of a valid SCPI query using the hierarchy tree in Figure 2-1 on page 2-2.

Query Responses

The query causes the data timing generator to return information about its status or settings. When a query is sent to the data timing generator, only the values are returned. When the returned value is a mnemonic, it is noted in abbreviated format, as shown in Table 2-2.

Table 2-2: Query response examples

Query	Response
SYSTem:VERSion?	1999.0
DIAGnostic:SELect?	ALL

A few queries also initiate an operation action before returning information. For example, the ***CAL?** query runs a calibration.

Parameter Types

Parameters are indicated by angle brackets, such as <file_name>. There are several different types of parameters, as listed in Table 2-3. The parameter type is listed after the parameter. Some parameter types are defined specifically for the DTG5000 series command set and some are defined by SCPI.

Table 2-3: Parameter types used in syntax descriptions

Parameter Type	Description	Example
arbitrary block	A block of data bytes	#512234xxxxx... where 5 indicates that the following 5 digits (12234) specify the length of the data in bytes; xxxxx... indicates the data or #0xxxxx...<LF><&EOI>
boolean	Boolean numbers or NRf	ON or ≠ 0 OFF or 0
discrete	A list of specific values	MIN, MAX
binary	Binary numbers	#B0110
octal	Octal numbers	#Q75, #Q3
hexadecimal	Hexadecimal numbers (0-9, A- F)	#HAA, #H1
NR1 numeric	Integers	0, 1, 15, -1
NR2 numeric	Decimal numbers	1.2, 3.141516, -6.5
NR3 numeric	Floating point numbers	3.1415E-9, -16.1E5
NRf numeric	Flexible decimal number that may be type NR1, NR2, or NR3	See NR1, NR2, NR3 examples in this table
Numeric numeric	Flexible decimal number that may be type NR1, NR2, NR3, or specific value (MIN, MAX).	See NR1, NR2, NR3 discrete examples in this table
string	Alphanumeric characters (must be within quotation marks)	"Testing 1, 2, 3"

About MIN, MAX

You can use MINimum, MAXimum keywords in addition to Numeric in the commands with "Numeric" parameter. You can set the minimum value or the maximum value by the use of this keywords. You can query the minimum value or the maximum value at that time.

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.

Abbreviating Commands, Queries, and Parameters

You can abbreviate most SCPI commands, queries, and parameters to an accepted short form. This manual shows these commands as a combination of upper and lower case letters. The upper case letters indicate the accepted short form of a command, as shown in Figure 2–2. The accepted short form and the long form are equivalent and request the same action of the instrument.

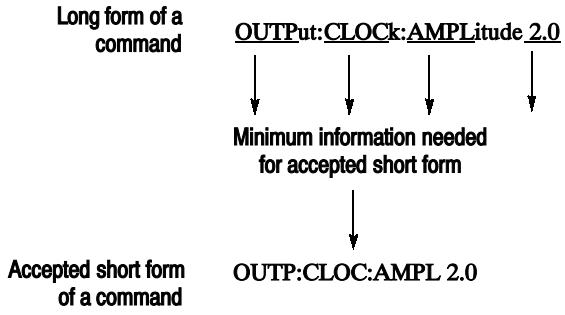


Figure 2-2: Example of abbreviating a command

Chaining Commands and Queries

You can chain several commands or queries together into a single message. To create a chained message, first create a command or query, then add a semicolon (;), and finally add more commands or queries and semicolons until you are done. If the command following a semicolon is a root node, precede it with a colon (:). Figure 2–3 illustrates a chained message consisting of several commands and queries. The chained message should end in a command or query, not a semicolon. Responses to any queries in your message are separated by semicolons.

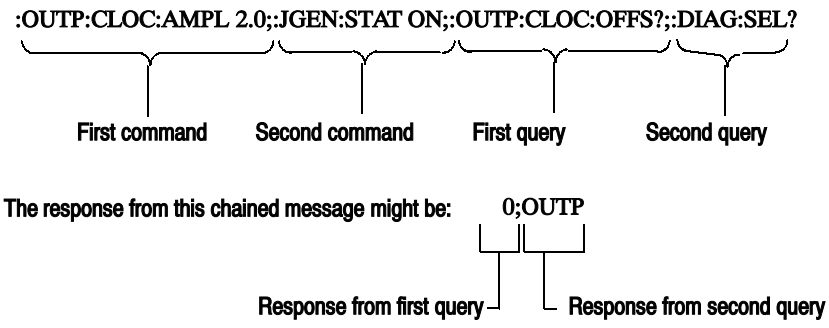


Figure 2–3: Example of chaining commands and queries

If a command or query has the same root and lower-level nodes as the previous command or query, you can omit these nodes. In Figure 2–4, the second command has the same upper node (OUTP:DC) as the first command, so these nodes can be omitted.

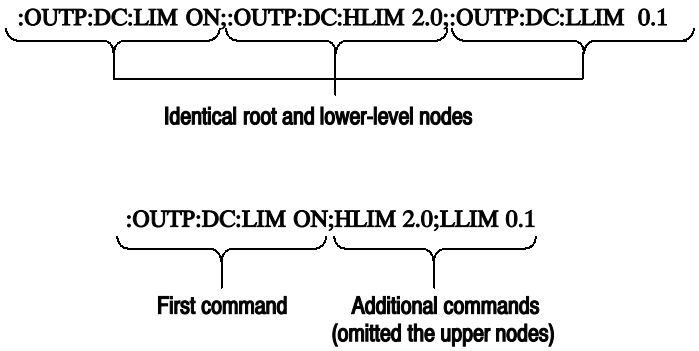


Figure 2–4: Example of omitting upper and lower-level nodes in a chained message

Unit and SI Prefix

If the decimal numeric argument refers to voltage, frequency, impedance, or time, you can express it using SI units instead of using the scaled explicit point input value format <NR3>. (SI units are units that conform to the System International d'Unites standard.) For example, you can use the input format 200 mV or 1.0 MHz 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 for voltage (V).

HZ for frequency (Hz).

OHM for impedance (ohm).

S for time (s).

DBM for power ratio

PCT for %

VPP for Peak-to-Peak Voltage (V p-p).

UIPP for Peak-to-Peak, Unit is UI (UI p-p).

UIRMS for RMS, Unit is UI (UIrms).

SPP for Peak-to-Peak, Unit is second (s p-p).

SRMS for RMS, Unit is second (srms).

V/NS for SLEW's unit (V/ns).

In the case of angle, you can use RADian and DEGree. The default unit is RADian.

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

SI prefix *	Corresponding power
EX	10^{18}
PE	10^{15}
T	10^{12}
G	10^{9}
MA	10^{6}
K	10^{3}
M	10^{-3}

SI prefix *	Corresponding power
U	10^{-6}
N	10^{-9}
P	10^{-12}
F	10^{-15}
A	10^{-18}

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

*** Note that the prefix u/U is used instead of "μ".**

Use **mV** for **V**, and **MHz** for **Hz**.

The SI prefixes need units.

correct: 10MHz, 10E+6Hz, 10E+6

incorrect: 10M

General Rules

Here are three general rules for using SCPI commands, queries, and parameters:

- You can use single (‘ ’) or double (“ ”) quotation marks for quoted strings, but you cannot use both types of quotation marks for the same string.

  correct: “This string uses quotation marks correctly.”

  correct: ‘This string also uses quotation marks correctly.’

  incorrect: “This string does not use quotation marks correctly.’

- You can use upper case, lower case, or a mixture of both cases for all commands, queries, and parameters.

  :OUTPUT:DC:LEVEL 0,1.1V

  is the same as

  output:dc:level 0,1.1V

  and

  OUTPUT:dc:LEVEL 0,1.1V

NOTE. *Literal strings (quoted) are case sensitive. For example: file names.*

- No embedded spaces are allowed between or within nodes.

  correct: OUTPUT:DC:LEVEL 0,1.1V

  incorrect: OUTPUT: DC: LEVEL 0,1.1V

IEEE 488.2 Common Commands

ANSI/IEEE Standard 488.2 defines the codes, formats, protocols, and usage of common commands and queries used on the interface between the controller and the instruments. The data timing generator complies with this standard.

The syntax for an IEEE 488.2 common command is an asterisk (*) followed by a command and, optionally, a space and parameter value. The syntax for an IEEE 488.2 common query is an asterisk (*) followed by a query and a question mark. All of the common commands and queries are included in the *Syntax and Commands* section of this manual. The following are examples of common commands:

- *ESE 16
- *CLS

The following are examples of common queries:

- *ESR?
- *IDN?

Specifying a Physical Channel

On a DTG5000 Series instrument, for example, you can set the high level as follows:

PGEN<x><m>:CH<n>:HIGH 2.0

<x> represents one of modules A to H

<m> represents one of mainframe numbers 1 to 3

<n> represents one of channels 1 to 4
If the mainframe number is 1, you can omit **<m>**.

Examples

```
PGENA:CH2:AMPLitude 1.2
```
Sets the amplitude of Mainframe 1, Slot A, Channel 2 to 1.2 V.

```
PGENA2:CH2:AMPLitude 1.2
```
Sets the amplitude of Mainframe 2, Slot A, Channel 2 to 1.2 V.

```
PGENB1:CH2:BDATa? 2,10
```
Reads the data for 10 vectors from Address 2 of Mainframe 1, Slot B, Channel 2.

```
PGENH:CH2:DATA? 2,10
```
Reads the data for 10 vectors from Address 2 of Mainframe 1, Slot H, Channel 2.

```
PGENG3:CH2:DCYCle 1
```
Sets the duty cycle of Mainframe 3, Slot G, Channel 2 to 1%.

```
PGENA:CH1:DTOFset:STATe ON
```
Turns on the differential timing offset of Mainframe 1, Slot A, Channel 1.

Syntax Diagrams

The syntax of each command and query is explained by both syntax diagrams and BNF notation. Figure 2–5 shows some typical syntax diagram structures. The syntax diagrams are described by the following symbols and notation:

- Oval symbols contain literal elements, such as a command or query header and a nonquoted string argument.
- Circle symbols contain separators or special symbols, such as (:), (,), and (?).
- Box symbols contain the defined element, such as <NR1>.
- Arrow symbols connect elements to show the paths that can be taken through the diagram and, thereby, the order in which the elements can be sent in a command structure.
- Parallel paths show that only one of the paths can be taken in the command. See diagram 1 in Figure 2–5.
- A loop around an element(s) shows the element can be repeated. See diagram 2 in Figure 2–5.
- A path around a group of elements shows that those elements are optional. See diagram 3 in Figure 2–5.

NOTE. *The unit and SI prefix that can be added to decimal numeric arguments are not described in the syntax diagram. See* Unit and SI Prefix *on page 2–7.*

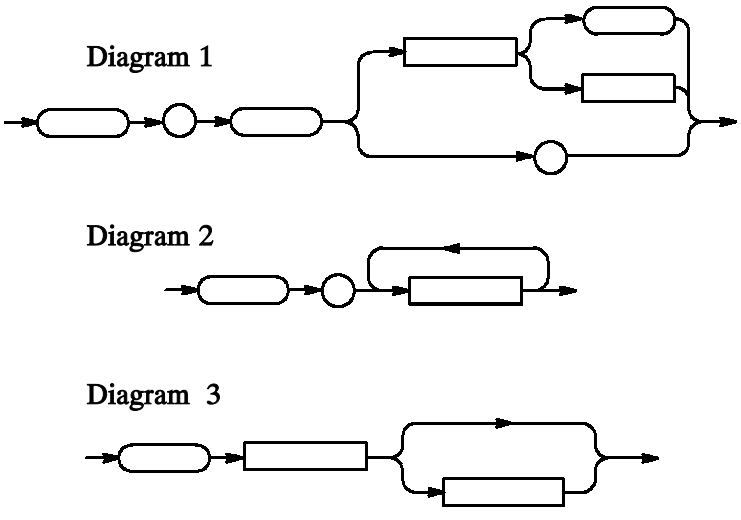


Figure 2–5: Typical syntax diagrams

Command Groups

This section lists commands in two ways, by functional groups and alphabetically. The functional group list starts below. The alphabetical list provides more detail on each command and starts on page 2–21.

The GPIB interface of DTG5000 Series conforms to SCPI (Standard Commands for Programmable Instruments) 1999.0 and IEEE Std 488.2–1992, except where noted.

Functional Groups

Table 2–4 lists the functional groups into which the DTG5000 Series Data Timing Generator commands are classified.

Table 2–4: Functional groups in the DTG command set

Group	Function
Common Commands	General commands to a GPIB instrument
Device Commands	Specific to the DTG5000 Series.

Command Quick Reference

Be sure that this page through page 2–112, list all the commands in each functional group and can be copied for use as a quick reference.

Common Commands

- *CAL?
- CALibration[:ALL] (?)
- *CLS
- DIAGnostic:DATA?
- DIAGnostic:IMMediate (?)
- DIAGnostic:SELect (?)
- *ESE (?)
- *ESR?
- *IDN?
- *OPC (?)
- *OPT?
- *RST
- *SRE (?)
- *STB?
- SYSTem:ERRor[:NEXT]?
- SYSTem:KLOCk (?)
- SYSTem:VERSion?
- *TRG
- *TST?
- *WAI

Device Commands

- BLOCk:DELete
- BLOCk:DELete:ALL
- BLOCk:LENGth (?)
- BLOCk:NEW
- BLOCk:SELect (?)
- GROup:DELete
- GROup:DELete:ALL
- GROup:NEW
- GROup:WIDTh (?)
- JGENeration:AMPLitude (?)
- JGENeration:AMPLitude:UNIT (?)
- JGENeration:EDGE (?)
- JGENeration:FREQuency (?)
- JGENeration:GSOurce (?)
- JGENeration:MODE (?)
- JGENeration:PROFile (?)
- JGENeration[:STATe] (?)
- MMEMory:LOAD
- MMEMory:STORe
- OUTPut:CLOCk:AMPLitude (?)
- OUTPut:CLOCk:OFFSet (?)
- OUTPut:CLOCk[:STATe] (?)
- OUTPut:CLOCk:TIMPedance (?)
- OUTPut:CLOCk:TVOLtage (?)
- OUTPut:DC:HLIMit (?)
- OUTPut:DC:LEVel (?)
- OUTPut:DC:LIMit (?)
- OUTPut:DC:LLIMit (?)
- OUTPut:DC[:STATe] (?)
- OUTPut:STATe:ALL
- PGEN<x>[m]:CH<n>:AMODe (?)
- PGEN<x>[m]:CH<n>:AMPLitude (?)
- PGEN<x>[m]:CH<n>:BDATa (?)
- PGEN<x>[m]:CH<n>:CPOint (?)
- PGEN<x>[m]:CH<n>:DATA (?)
- PGEN<x>[m]:CH<n>:DCYCle (?)
- PGEN<x>[m]:CH<n>:DTOFfset (?)
- PGEN<x>[m]:CH<n>:DTOFfset:STATe (?)
- PGEN<x>[m]:CH<n>:HIGH (?)
- PGEN<x>[m]:CH<n>:HLIMit (?)
- PGEN<x>[m]:CH<n>:LDELay (?)
- PGEN<x>[m]:CH<n>:LHOLd (?)
- PGEN<x>[m]:CH<n>:LIMit (?)
- PGEN<x>[m]:CH<n>:LLIMit (?)
- PGEN<x>[m]:CH<n>:LOW (?)
- PGEN<x>[m]:CH<n>:OFFSet (?)
- PGEN<x>[m]:CH<n>:OUTPut (?)
- PGEN<x>[m]:CH<n>:PHASe (?)
- PGEN<x>[m]:CH<n>:POLarity (?)
- PGEN<x>[m]:CH<n>:PRATe (?)
- PGEN<x>[m]:CH<n>:SLEW (?)
- PGEN<x>[m]:CH<n>:TDELay (?)
- PGEN<x>[m]:CH<n>:THOLd (?)
- PGEN<x>[m]:CH<n>:TIMPedance (?)
- PGEN<x>[m]:CH<n>:TVOLtage (?)
- PGEN<x>[m]:CH<n>:TYPE (?)
- PGEN<x>[m]:CH<n>:WIDTh (?)
- PGEN<x>[m]:ID?
- SEQuence:DATA (?)
- SEQuence:LENGth (?)
- SIGNal:ASSign (?)
- SIGNal: <parameter> (?)
- SIGNal:BDATa (?)
- SIGNal:DATA (?)
- SUBSequence:DATA (?)
- SUBSequence:DELete
- SUBSequence:DELete:ALL
- SUBSequence:LENGth (?)
- SUBSequence:NEW
- SUBSequence:SELect (?)
- TBAS:COUNt (?)
- TBAS:CRANge (?)
- TBAS:DOFFset (?)
- TBAS:EIN:IMMediate
- TBAS:EIN:IMPedance (?)
- TBAS:EIN:LEVel (?)
- TBAS:EIN:POLarity (?)
- TBAS:FREQuency (?)
- TBAS:JMODe (?)
- TBAS:JTIMing (?)
- TBAS:JUMP
- TBAS:LDELay (?)
- TBAS:MODE (?)
- TBAS:OMODe (?)
- TBAS:PERiod (?)
- TBAS:PRATe?
- TBAS:RSTate?
- TBAS:RUN (?)
- TBAS:SMODe (?)
- TBAS:SOURce (?)
- TBAS:TIN:IMPedance (?)
- TBAS:TIN:LEVel (?)
- TBAS:TIN:SLOPe (?)
- TBAS:TIN:SOURce (?)
- TBAS:TIN:TIMer (?)
- TBAS:TIN:TRIGger
- TBAS:VRATe?
- VECTor:BDATa (?)
- VECTor:BIOFormat (?)
- VECTor:DATA (?)
- VECTor:IMPort
- VECTor:IOFormat (?)

Command Summaries

Tables 2–5 through 2–6 describe each command in each of the 2 functional groups.

Common Commands

The Common Commands are general commands to a GPIB instrument or other equivalent equipment.

Table 2–5: Common Commands

Header	Description
*CAL?	Runs all the calibrations and returns the result.
CALibration[:ALL](?)	Runs all the calibrations.
*CLS	Clears the event-related registers and queues.
DIAGnostic:DATA?	Reads the result of a self-test.
DIAGnostic:IMMediate(?)	Starts a self-test.
DIAGnostic:SELect (?)	Selects the self-test item to run.
*ESE(?)	Sets the Service Request Enable Register (SRER).
*ESR?	Queries the value of the Standard Event Status Register (SESR).
*IDN?	Returns the model name and other information.
*OPC(?)	Inserts a waiting time before all the processes end.
*OPT?	Queries the instrument options.
*RST	Initializes the instrument settings.
*SRE(?)	Sets the Service Request Enable Register (SRER).
*STB?	Queries the value of the Status Byte Register (SBR).
SYSTem:ERRor[:NEXT]?	Fetches the next item from the Error/Event Queue.
SYSTem:KLOCk(?)	Locks the controls of the front panel and keyboard.
SYSTem:VERSion?	Queries the SCPI version.
*TRG	Generates a trigger.
*TST?	Runs a self-test and returns the result.
*WAI	Inserts a waiting time before all the currently active commands end.

Device Commands

The device commands are peculiar to the DTG5000 Series.

Table 2-6: Device Commands

Header	Description
BLOCk:DELete	Deletes a block.
BLOCk:DELete:ALL	Deletes all the blocks.
BLOCk:LENGth(?)	Sets the block length.
BLOCk:NEW	Creates a new block.
BLOCk:SELect(?)	Selects a block used to transfer or import pattern data.
GROup:DELete	Deletes a group.
GROup:DELete:ALL	Deletes all the groups.
GROup:NEW	Creates a new group.
GROup:WIDTh(?)	Sets the width of a group in bits.
JGENeration:AMPLitude(?)	Sets the amplitude used for jitter generation.
JGENeration:AMPLitude:UNIT(?)	Sets the default unit of the amplitude used for jitter generation.
JGENeration:EDGE(?)	Sets the edge used for jitter generation.
JGENeration:FREQuency(?)	Sets the frequency used for jitter generation.
JGENeration:GSOurce(?)	Sets the gating source used for jitter generation.
JGENeration:MODE(?)	Sets the mode used for jitter generation.
JGENeration:PROFile(?)	Sets the profile used for jitter generation.
JGENeration[:STATe] (?)	Turns on or off jitter generation.
MMEMory:LOAD	Loads the settings file.
MMEMory:STORe	Saves the current settings in a file.
OUTPut:CLOCk:AMPLitude(?)	Sets the clock output amplitude.
OUTPut:CLOCk:OFFSet(?)	Sets the clock output offset.
OUTPut:CLOCk[:STATe](?)	Turns on or off the clock output.
OUTPut:CLOCk:TIMPedance(?)	Sets the clock output termination impedance.
OUTPut:CLOCk:TVOLtage(?)	Sets the clock output termination voltage.
OUTPut:DC:HLIMit(?)	Sets the high limit of the DC output.
OUTPut:DC:LEVel(?)	Sets the DC output level.
OUTPut:DC:LIMit(?)	Turns on or off the DC output limit.
OUTPut:DC:LLIMit(?)	Sets the low limit of the DC output.
OUTPut:DC[:STATe](?)	Turns on or off the DC output.

Table 2-6: Device Commands (cont.)

Header	Description
OUTPut:STATe:ALL	Turns on or off all outputs.
PGEN<x>[m]:CH<n>:AMODe(?)	Sets the channel composition mode for the data output.
PGEN<x>[m]:CH<n>:AMPLitude(?)	Sets the data output amplitude.
PGEN<x>[m]:CH<n>:BDATa(?)	Transfers pattern data in binary format.
PGEN<x>[m]:CH<n>:CPOint(?)	Sets the Cross Point of the NRZ.
PGEN<x>[m]:CH<n>:DATA(?)	Transfers pattern data.
PGEN<x>[m]:CH<n>:DCYCle(?)	Sets the data output duty cycle.
PGEN<x>[m]:CH<n>:DTOFfset(?)	Sets the differential timing offset value for the data output.
PGEN<x>[m]:CH<n>:DTOFfset:STATe(?)	Turns on or off the data output differential timing.
PGEN<x>[m]:CH<n>:HIGH(?)	Sets the high level of the data output.
PGEN<x>[m]:CH<n>:HLIMit(?)	Sets the high limit of the data output.
PGEN<x>[m]:CH<n>:LDELay(?)	Sets the lead delay of the data output.
PGEN<x>[m]:CH<n>:LHOLd(?)	Specifies how the data output leading edge is held.
PGEN<x>[m]:CH<n>:LIMit(?)	Sets whether the limit is applied.
PGEN<x>[m]:CH<n>:LLIMit(?)	Sets the low limit of the data output level.
PGEN<x>[m]:CH<n>:LOW(?)	Specifies the low level of the data output.
PGEN<x>[m]:CH<n>:OFFSet(?)	Sets the offset level of the data output.
PGEN<x>[m]:CH<n>:OUTPut(?)	Turns on or off the data output.
PGEN<x>[m]:CH<n>:PHASe(?)	Sets the data output phase.
PGEN<x>[m]:CH<n>:POLarity(?)	Sets the polarity of the data output.
PGEN<x>[m]:CH<n>:PRATe(?)	Sets the pulse rate.
PGEN<x>[m]:CH<n>:SLEW(?)	Sets the slew rate of the data output.
PGEN<x>[m]:CH<n>:TDELay(?)	Sets the trail delay of the data output.
PGEN<x>[m]:CH<n>:THOLd(?)	Specifies how to hold the data output trailing edge.
PGEN<x>[m]:CH<n>:TIMPedance(?)	Sets the data output termination impedance.
PGEN<x>[m]:CH<n>:TVOLtage(?)	Sets the data output termination voltage.
PGEN<x>[m]:CH<n>:TYPE(?)	Sets the data output format in DG mode.
PGEN<x>[m]:CH<n>:WIDTh(?)	Sets the data output pulse width.
PGEN<x>[m]:ID?	Examines the module.

Table 2-6: Device Commands (cont.)

Header	Description
SEQuence:DATA(?)	Sets the data corresponding to one line of a sequence.
SEQuence:LENGth(?)	Sets the sequence length.
SIGNal:ASSign(?)	Assigns a physical channel to the logical channel specified with the group name and bit number.
SIGNal:<parameter>(?)	Sets the data output parameters using a signal name.
SIGNal:BDATa(?)	Transfers pattern data in binary format.
SIGNal:DATA(?)	Transfers pattern data.
SUBSequence:DATA(?)	Sets the data corresponding to one line of a subsequence.
SUBSequence:DELete	Deletes a subsequence.
SUBSequence:DELete:ALL	Deletes all the subsequences.
SUBSequence:LENGth(?)	Changes the subsequence length.
SUBSequence:NEW	Creates a new subsequence.
SUBSequence:SELect(?)	Selects a subsequence.
TBAS:COUNt(?)	Sets the burst count.
TBAS:CRANge(?)	Sets the clock range.
TBAS:DOFFset(?)	Sets the delay offset.
TBAS:EIN:IMMediate	Generates an event.
TBAS:EIN:IMPedance(?)	Sets the event input impedance.
TBAS:EIN:LEVel(?)	Sets the event input level.
TBAS:EIN:POLarity(?)	Sets the polarity of the event input.
TBAS:FREQuency(?)	Sets the frequency.
TBAS:JMODe(?)	Sets the jump mode.
TBAS:JTIMing(?)	Sets the jump timing.
TBAS:JUMP	Causes a software jump.
TBAS:LDELay(?)	Sets the long delay.
TBAS:MODE(?)	Sets the PG run mode.
TBAS:OMODe(?)	Sets the operating mode.
TBAS:PERiod(?)	Sets the frequency.
TBAS:PRATe?	Queries the PLL multiplier rate.
TBAS:RSTate?	Queries the sequencer status.

Table 2-6: Device Commands (cont.)

Header	Description
TBAS:RUN(?)	Starts and stops the sequencer.
TBAS:SMODe(?)	Sets the sequencer mode.
TBAS:SOURce(?)	Sets the clock source.
TBAS:TIN:IMPedance(?)	Sets the trigger input impedance.
TBAS:TIN:LEVel(?)	Sets the trigger input level.
TBAS:TIN:SLOPe(?)	Sets the polarity of the trigger input.
TBAS:TIN:SOURce(?)	Sets the trigger input source.
TBAS:TIN:TIMer(?)	Sets the cycle of the internal trigger.
TBAS:TIN:TRIGger	Generates a trigger.
TBAS:VRATe?	Queries the vector rate.
VECTor:BDATa(?)	Transfers pattern data in binary format.
VECTor:BIOFormat(?)	Sets the data items to be transferred with VECTor:BDATa.
VECTor:DATA(?)	Transfers pattern data in ASCII format.
VECTor:IMPort	Read pattern data from a file.
VECTor:IOFormat(?)	Sets the data items to be transferred with VECT or DATA and their format.

Command Descriptions

This subsection lists each command and query in the data timing generator command set in alphabetical order. Each command entry includes a command description and command group, related commands (if any), syntax, and 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 `BLOCk:DELete` command, just type `BLOC:DEL`.

The symbol "`(?)`" follows the command header of commands that can be used as either a command or a query; the symbol "`?`" follows commands that can only be used as a query. Commands that are command-only or query-only are noted as such.

BLOCk:DELete (No Query Form)

This command deletes a block.

Syntax `BLOCk:DELete <block_name>`

Arguments `<block_name>` ::= <string> – Block name

Examples `BLOCk:DELete "Group1"`
Deletes the block named "Group1".

BLOCk:DELete:ALL (No Query Form)

This command deletes all the blocks.

Syntax `BLOCk:DELete:ALL`

Arguments None

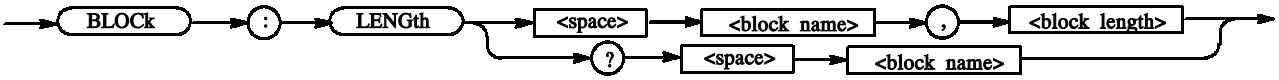
Examples `BLOCk:DELete:ALL`
Deletes all the blocks.

BLOCk:LENGth (?)

This command sets the block length.

Syntax `BLOCk:LENGth <block_name>, <block_length>`

`BLOCk:LENGth? <block_name>`



Arguments <block_name> ::= <string> - Block name

<block_length> ::= <Numeric> - The range is as follows:

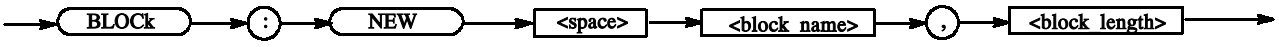
DTG 5078: 1 to 8,000,000
DTG 5274: 1 to 32,000,000

Examples `BLOCk:LENGth "Block1",960`
Sets the block length of “Block1” to 960.
`BLOCk:LENGth? "Block2"`
Queries the block length of “Block2”.
If the block name is not found, the following will be returned: −1

BLOCk:NEW (No Query Form)

This command creates a new block.

Syntax `BLOCk:NEW <block_name>, <block_length>`



Arguments <block_name> ::= <string> - The block name consists of 32 characters or less.

<block_length> ::= <Numeric> - The range is as follows:

DTG 5078: 1 to 8,000,000
DTG 5274: 1 to 32,000,000

You can create up to 8,000 blocks.

Examples

```
BLOCk:NEW "Block1",960
```

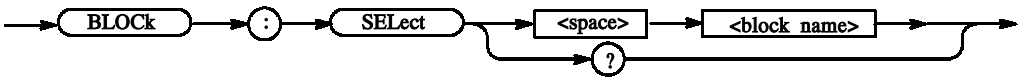
Creates a block with a length of 960 under the name of “Block1”.

BLOCk:SELect (?)

This command selects a block used to transfer or import pattern data.

Syntax

```
BLOCk:SELect <block_name>
BLOCk:SELect?
```



Arguments

<block_name> ::= <string> - Block name

*RST returns the setting to “”.

Examples

```
BLOCk:SELected "Block1"
```

Selects a block named “Block1”.

*CAL? (Query Only)

The *CAL? query performs a level calibration and returns a status that indicates whether or not the data timing generator completed the calibration successfully. If an error is detected during calibration, execution immediately stops, and an error code is returned. This query performs the same function as the CALibration[:ALL]? query.

NOTE. *A period of time is required to complete the internal calibration. During this time, the data timing generator does not respond to any commands or queries issued.*

Syntax

```
*CAL?
```

Arguments None

Returns <NR1>

0 Terminated without error.
-340 Calibration failed.

Examples *CAL?
performs an internal calibration and returns the results. For example, the query might return 0, which indicates the calibration terminated without any errors.

CALibration[:ALL] (?)

The CALibration[:ALL] command performs a level calibration of the data timing generator.

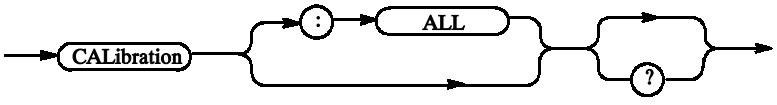
The CALibration[:ALL]? query performs a level calibration and responds with an <NR1> indicating the success of the calibration. This query has the same function as the *CAL? query.

If an error is detected during calibration, a message is queued in the error/event queue, and the error code "-340" is returned.

NOTE. *A period of time is required to complete the internal calibration. During this time, the data timing generator does not respond to any commands or queries issued.*

Syntax CALibration[:ALL]

CALibration[:ALL]?



Arguments None

Returns <NR1>

0 Terminated without error.
-340 Calibration failed.

Examples `CALibration[:ALL]?`
performs a level calibration and returns the results. For example, it might return 0, which indicates the calibration terminated without any errors.

`CALibration[:ALL]`
performs a level calibration.
In the case, when it becomes a error, the `SYSTem:ERRor[:NEXT]?` command can be checked the error information.
Detailed information continues after that by an event number -340 and "Calibration failed".

*CLS (No Query Form)

This command clears all the event registers and queues, used by the data timing generator status and event reporting system. For more details, refer to the, *Status and Events* section.

Syntax `*CLS`

Arguments None

Examples `*CLS`
clears all the event registers and queues.

DIAGnostic:DATA? (Query Only)

This command returns the results of a self-test.

Syntax `DIAGnostic:DATA?`

Arguments None

Returns <NR1>

0 Terminated without error.
-330 Self-test failed.

Examples DIAGnostic:DATA?
might return 0.

DIAGnostic:IMMediate (?)

The DIAGnostic:IMMediate command executes the self-test routine(s) selected by the DIAGnostic:SELect command. The query DIAGnostic:IMMediate? executes the routine(s) and returns the results.

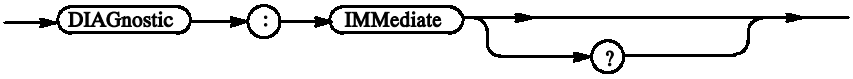
If an error is detected during execution, the routine that detected the error terminates. If all of the self-test routines are selected, self-testing continues with execution of the next self-test routine.

The command without "?" perform a self test simply. In the case, when it becomes a dialog error, an event occurs. Detailed information continues after that by an event number -330 and "Self test failed". Detailed information is the set of a dialog error code and auxiliary information, and is the same contents as what is displayed on a screen.

A result can be checked by DIAGnostic:DATA?.

Syntax DIAGnostic:IMMediate

DIAGnostic:IMMediate?



Arguments None

Returns <NR1>

0 Terminated without error.
-330 Self-test failed.

Examples DIAGnostic:SELect ALL;IMMediate?
executes all of the self-test routines. After all self-test routines finish, the results of the self-tests are returned.

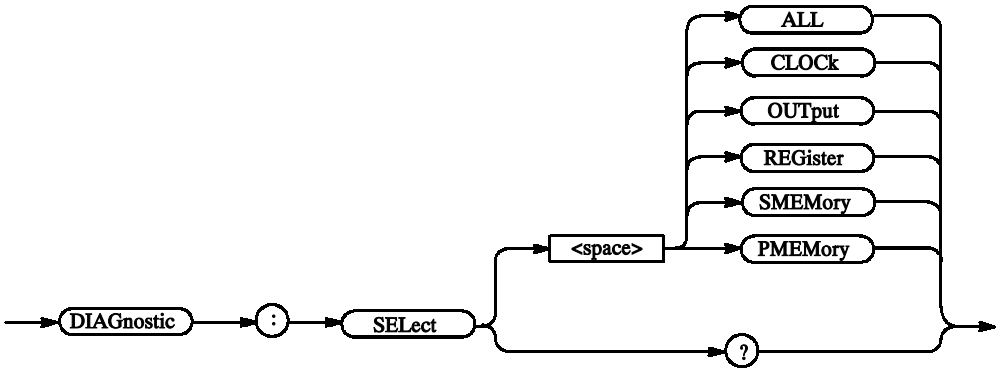
DIAGnostic:SELect (?)

This command selects the self-test routine(s).

Syntax

```
DIAGnostic:SELect { ALL | CLOCk | OUTPut | REGister | SMEMory |
PMEMory }

DIAGnostic:SELect?
```



Arguments You can select the following self-test routines:

Table 2-7: Self-test routines

Argument	Description
ALL	Checks all routines that follow
CLOCk	Checks the clock unit
OUTput	Checks the output unit
REGister	Checks the register unit
SMEMory	Checks the sequence memory
PMEMory	Checks the pattern memory

At *RST, this parameter is set to ALL.

Examples

```
DIAGnostic:SELect SMEMory;IMMediate
```

executes the sequence memory self-test routine.

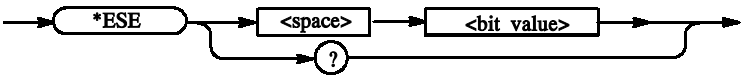
*ESE (?)

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

Syntax

```
*ESE <bit_value>
```

```
*ESE?
```



Arguments

<bit_value>::=<NR1>
where <NR1> is a decimal integer in the range 0 to 255. The binary bits of the ESER are set according to this value.

The power-on default for ESER is 0 if *PSC is 1. If *PSC is 0, the ESER maintains its value through a power cycle.

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 10110000.

*ESR? (Query Only)

This command returns the contents of the Standard Event Status Register (SESR) used in the status and events reporting system in the data timing generator. *ESR? also clears the SESR (since reading the SESR clears it). Refer to Section 3 *Status and Events* for more information.

Syntax

```
*ESR?
```

Returns

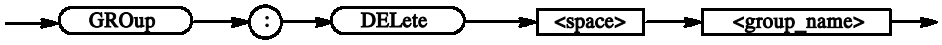
<NR1> indicates the content of the SESR in a decimal integer.

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

GROup:DELete (No Query Form)

This command deletes a group.

Syntax GROup:DELete <group_name>



Arguments <group_name> ::= <string> - Group name

Examples GROup:DELete "Group1"
Deletes a group named “Group1”.

GROup:DELete:ALL (No Query Form)

This command deletes all the groups.

Syntax GROup:DELete:ALL

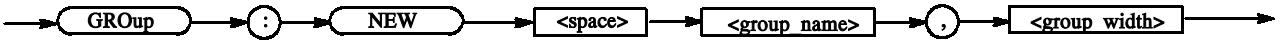
Arguments None

Examples GROup:DELete:ALL
Deletes all the groups.

GROup:NEW (No Query Form)

This command creates a new group.

Syntax `GROup:NEW <group_name>, <group_width>`



Arguments <group_name> ::= <string> - The group name consists of 32 characters or less.

<group_width> ::= <Numeric> - The range is 1 to 96.

You can create up to 96 groups.

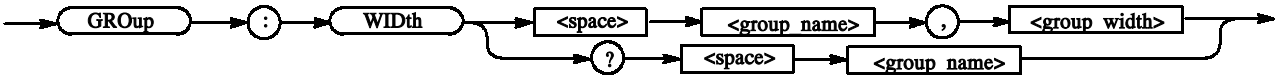
Examples `GROup:NEW "group1",8`
Creates a group with an 8-bit width under the name of “group1”.

GROup:WIDTh (?)

This command sets the width of a group in bits.

Syntax `GROup:WIDth <group_name>, <group_width>`

`GROup:WIDth? <group_name>`



Arguments <group_name> ::= <string> - Group name

<group_width> ::= <Numeric> - The range is 1 to 96.

Returns <NR1>

Examples `GROup:WIDTh "group1",4`
Sets the width of “group1” to 4 bits.
If the group name is not found when queries the width of a group, the following will be returned: −1

*IDN? (Query Only)

This command returns identification information for the data timing generator.

Syntax `*IDN?`

Arguments None

Returns `<manufacturer>, <model>, <serial_number>, <firmware_level>`
where
`<manufacturer>::=Tektronix`
`<model>::={ DTG5274 | DTG5078 }`
`<serial_number>::=0` - Indicates that 0 is not applied.
`<firmware_level>::=SCPI:99.0, FW:1.0` - System software version

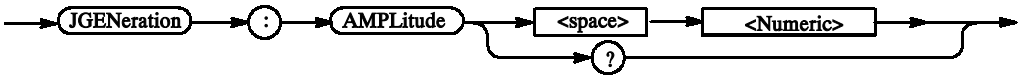
Examples `*IDN?`
might return `TEKTRONIX,DTG5078,0,SCPI:99.0 FW:1.0`

JGENeration:AMPLitude (?)

This command sets the amplitude used for jitter generation.

Syntax `JGENeration:AMPLitude <Numeric>`

`JGENeration:AMPLitude?`



Arguments For the unit for setting, you can specify SPP, SRMS, UIPP, or UIRMS.
If you omit the unit, the system assumes that there is a unit you specified in `JGENeration:AMPLitude:UNIT`. For the meanings of units, see the description of `JGENeration:AMPLitude:UNIT`.

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0 (unit: SPP).

Returns <NR3>

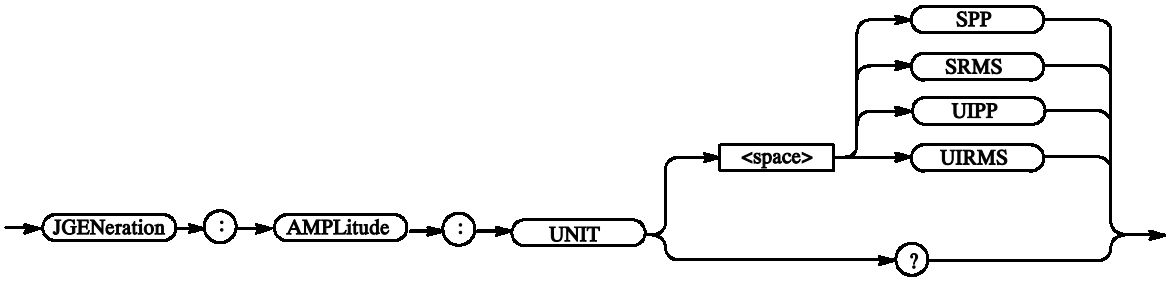
Examples JGENeration:AMPLitude 1e-10
Sets the amplitude for jitter generation to 100ps when UNIT is SPP.
JGENeration:AMPLitude? MAX
Query the maximum amplitude for jitter generation at the current unit.

JGENeration:AMPLitude:UNIT (?)

This command sets the default unit of the amplitude used for jitter generation. This command specifies the default unit that is added when a numeric without unit is sent with JGENeration:AMPLitude. Also when a query is performed with JGENeration:AMPLITude?, the unit you specified in this command is added. In addition, this command also specifies whether s (second) or UI (unit interval) is used as the unit that is used to retain the value when you change the frequency. If you specify SPP or SRMS, s will be used. If you specify UIPP or UIRMS, UI will be used.

Syntax JGENeration:AMPLitude:UNIT <amplitude unit>

JGENeration:AMPLitude:UNIT?



Arguments <amplitude unit> ::= {SPP | SRMS | UIPP | UPRMS}

SPP - Represents the peak to peak value in seconds (s).
SRMS - Represents the root mean square (effective value) in seconds (s).
UIPP - Represents the peak to peak value in unit intervals (UI).
UIRMS - Represents the root mean square (effective value) in unit intervals (UI).

*RST returns the setting to 0 (unit: SPP).

Returns <amplitude unit>

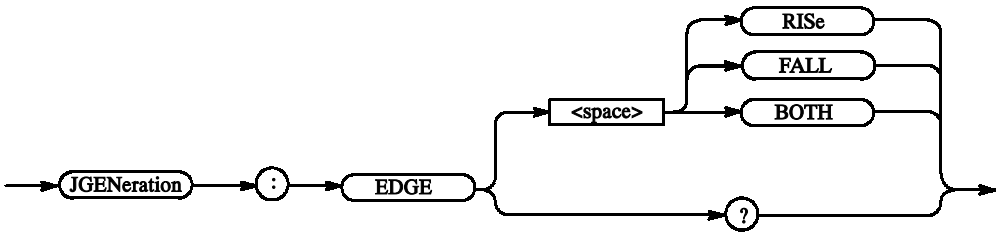
Examples JGENeration:AMPLitude:UNIT SRMS
Sets SRMS for the default unit of the amplitude used for jitter generation.

JGENeration:EDGE (?)

This command sets the edge used for jitter generation.

Syntax JGENeration:EDGE { RISe | FALL | BOTH }

JGENeration:EDGE?



Arguments RISe: Sets the edge to rising.

FALL: Sets the edge to falling.

BOTH: Sets the edge to both rising and falling.

*RST sets the edge to BOTH.

Returns { RISe | FALL | BOTH }

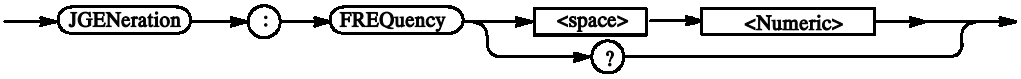
Examples JGENeration:EDGE RISe
Sets rising for the edge used for jitter generation.

JGENeration:FREQuency (?)

This command sets the frequency used for jitter generation (other than GNOise).

Syntax `JGENeration:FREQuency <Numeric>`

`JGENeration:FREQuency?`



Arguments Range: 0.015 Hz to 1.56 MHz

Step: 1e-3 Hz

*RST returns the setting to 1e6 Hz.

Returns <NR3>

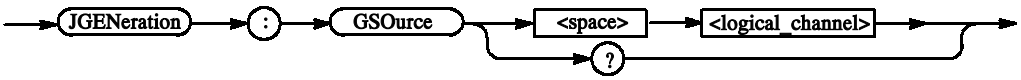
Examples `JGENeration:FREQuency 1MHz`
Sets 1 MHz for the frequency used for jitter generation.

JGENeration:GSOurce (?)

This command sets the gating source, that is, the group and bit to which to apply the jitter.

Syntax `JGENeration:GSOurce <logical_channel>`

`JGENeration:GSOurce?`



Arguments <logical_channel> ::= <string> - Logical channel. Use one of the following formats:

<group_name> - For a group with a 1-bit width

<group_name>[<bit>] - Specified bit number in the specified group(This "[]" can't omit.)

Example:

```
CLK
Addr[0]
```

The jitter will be applied only to Channel 1 of Slot A of the master.

*RST returns the setting to "".

Returns <logical_channel>

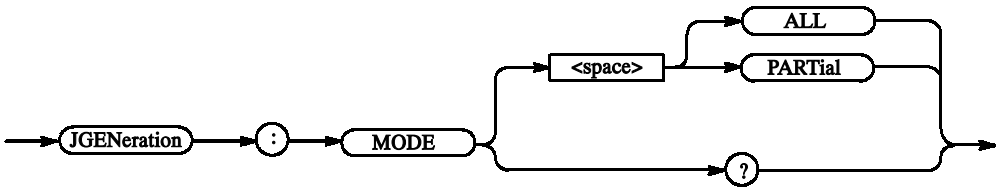
Examples JGENeration:GSOurce "Group1[0]"
Sets the jitter generation gating source in Bit 0 of "Group1".

JGENeration:MODE (?)

This command sets the mode used for jitter generation.

Syntax JGENeration:MODE {ALL | PARTial}

JGENeration:MODE?



Arguments ALL: Applies the jitter to the entire output signal.

PARTial: Applies the jitter to part of the output signal.

*RST returns the setting to ALL.

Returns { ALL | PARTial }

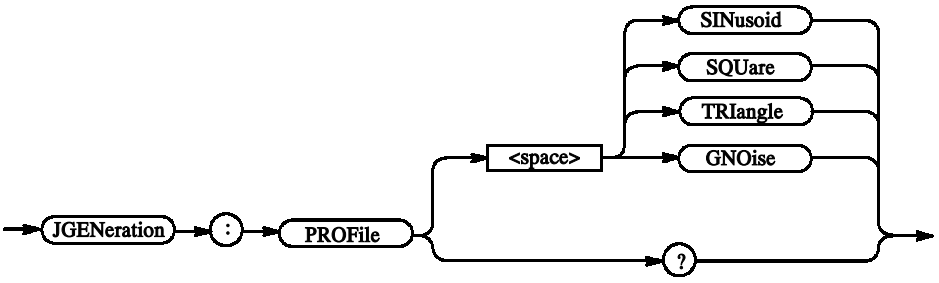
Examples JGENeration:MODE ALL
Applies the jitter to the entire output signal.

JGENeration:PROFile (?)

This command sets the profile used for jitter generation.

Syntax

```
JGENeration:PROFile <jitter_profile>
JGENeration:PROFile?
```



Arguments

<jitter_profile> - Waveform type. You can select one of the following:

```
SINusoid     : Sine wave
SQUare       : Square wave
TRIangle     : Triangular wave
GNOise       : Gaussian noise
```

*RST returns the setting to SINusoid.

Returns

<jitter_profile>

Examples

```
JGENeration:PROFile SQUare
```

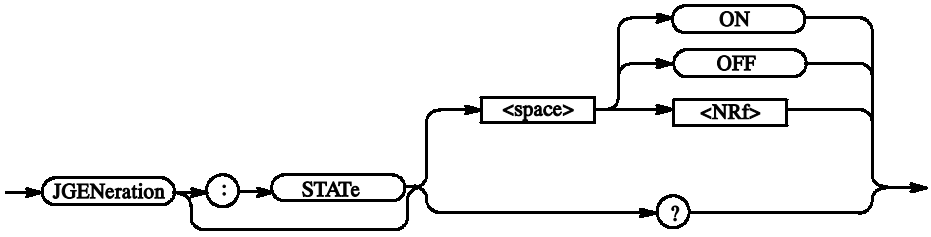
Generates a jitter with a square wave.

JGENeration[:STATe] (?)

This command turns on or off jitter generation.

Syntax

```
JGENeration[:STATe] {ON | OFF | <NRf>}
JGENeration[:STATe]?
```



Arguments

OFF or <NRf> = 0 – Turns off jitter generation.

ON or <NRf> ≠ 0 – Turns on jitter generation.

The jitter cannot be turned on if Long Delay in DG mode is ON and an Output Module is not inserted in Slot A of the master. (You can set the parameters such as the jitter amplitude even in this case.)

The jitter can be applied only to Channel 1 of Slot A of the master. Channel 2 is disabled at this time.

*RST returns the setting to 0 (Off).

Returns

<NR1>

Examples

```
JGENeration:STATe ON
```

Turns on jitter generation.

MMEMory:LOAD (No Query Form)

This command loads the settings file.

Syntax `MMEMory:LOAD <filename>`

Arguments `<filename>` ::= <string> - File name (absolute path)

Examples `MMEMory:LOAD "C:\tmp\abc.dat"`
Loads a setting file named “C:\tmp\abc.dat”.

MMEMory:STORe (No Query Form)

This command saves the current settings in a file.

Syntax `MMEMory:STORe <filename>`

Arguments `<filename>` ::= <string> - File name (absolute path)

Examples `MMEMory:STORe "C:\tmp\abc.dat"`
Saves a setting file named “C:\tmp\abc.dat”.

*OPC (?)

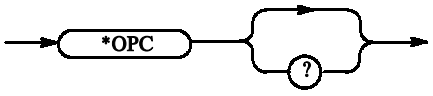
Operation complete command (query). Use this command between two other commands to ensure completion of the first command before processing the second command.

In this application, all commands are designed to be executed in the order in which they are sent from the external controller.

Refer to page 3-5 about the OPC bit of SESR (Standard Event Status Register).

Syntax *OPC

*OPC?



Arguments None

Returns <NR1> ::=1 All the active commands are complete (Operation Complete).

Examples PGENA1:CH1:HIGH 2.0;*OPC
An end can be checked when the event of Operation Complete occurs.

PGENA1:CH1:HIGH 2.0;*OPC?
An end can be checked when 1 should be returned.

*OPT? (Query Only)

This command returns the implemented options of the data timing generator.

Syntax *OPT?

Arguments None

Returns 0

Examples Since 0 is always returned, you do not need to use this query in actual programs.

OUTPut:CLOCk:AMPLitude (?)

This command sets the clock output amplitude.

Syntax `OUTPut:CLOCk:AMPLitude <Numeric>`

`OUTPut:CLOCk:AMPLitude?`

Arguments Range: 0.03 to 1.25 V

Step: 10 mV

*RST returns the setting to 1.0 V.

Returns <NR3>

Examples `OUTPut:CLOCk:AMPLitude 0.5`
Sets the clock output amplitude to 0.5 V.

OUTPut:CLOCk:OFFSet (?)

This command sets the clock output offset.

Syntax `OUTPut:CLOCk:OFFSet <Numeric>`

`OUTPut:CLOCk:OFFSet?`

Arguments Range: -0.985 to 3.485 V (when the amplitude is 30 mV)

Step: 40 mV

*RST returns the setting to 0.48 V.

Returns <NR3>

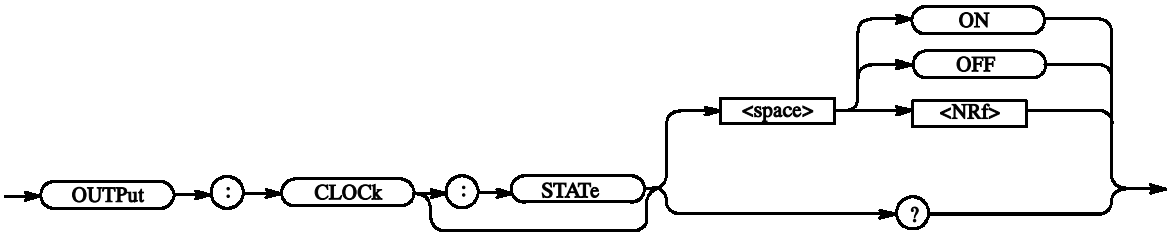
Examples `OUTPut:CLOCk:OFFSet 0.1`
Sets the clock output offset to 0.1 V.

OUTPut:CLOCk[:STATe] (?)

This command turns on or off the clock output.

Syntax `OUTPut:CLOCk[:STATe] {ON | OFF | <NRf>}`

`OUTPut:CLOCk[:STATe]?`



Arguments OFF or <NRf> = 0 – Turns off the clock output.

ON or <NRf> ≠ 0 – Turns on the clock output.

*RST returns the setting to 0 (Off).

Returns <NR1>

Examples `OUTPut:CLOCk:STATe ON`
Turns on the clock output.

OUTPut:CLOCk:TIMPedance(?)

This command sets the clock output termination impedance.

Syntax `OUTPut:CLOCk:TIMPedance <Numeric>`

`OUTPut:CLOCk:TIMPedance?`

Arguments Range: 10 ohm to 1 M ohm

≤ 0: Open

Step: 3 significant digits. The minimum resolution is 1 ohm.

*RST returns the setting to 50.0 ohm.

Returns <NR3>

Examples OUTPut:CLOCk:TIMPedance 40
Sets the clock output termination impedance to 40 ohm.

OUTPut:CLOCk:TVOLtage(?)

This command sets the clock output termination voltage.

Syntax OUTPut:CLOCk:TVOLtage <Numeric>

OUTPut:CLOCk:TVOLtage?

Arguments Range: -2 to +5V

Step: 0.1 V

*RST returns the setting to 0.0 V.

Returns <NR3>

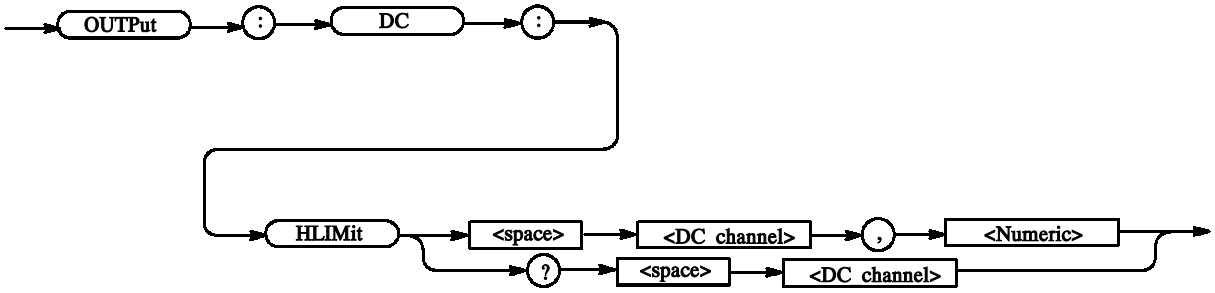
Examples OUTPut:CLOCk:TVOLtage 1.1
Sets the clock output termination voltage 1.1 V.

OUTPut:DC:HLIMit(?)

This command sets the high limit of the DC output.

Syntax OUTPut:DC:HLIMit <DC_channel>, <Numeric>

OUTPut:DC:HLIMit? <DC_channel>



Arguments <DC_channel> ::= <NR1> (0 to 23, if three units are concurrently used)

<Numeric> ::= High limit - Step: 30 mV, Range: -3 to 5V

If the low limit of the DC output exceeds its high limit, the low limit will be set to the same value as the high limit.

*RST returns the setting to 1.0 V.

Returns <NR3>

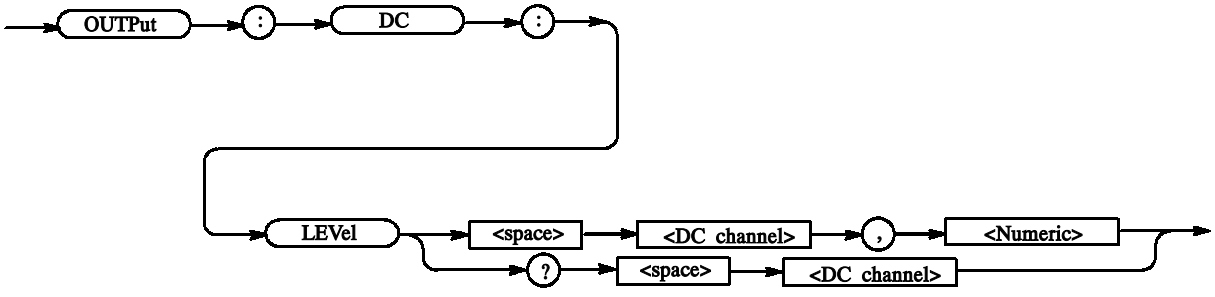
Examples OUTPut:DC:HLIMit 0,1.5
Sets the high limit of the DC output (0) to 1.5 V.

OUTPut:DC:LEVel(?)

This command sets the DC output level.

Syntax OUTPut:DC:LEVel <DC_channel>, <Numeric>

OUTPut:DC:LEVel? <DC_channel>



Arguments <DC_channel> ::= <NR1> (0 to 23, if three units are concurrently used)

<Numeric> ::= Level - Step: 30 mV, Range: Low limit to high limit of DC output

*RST returns the setting to 1.0 V.

Returns <NR3>

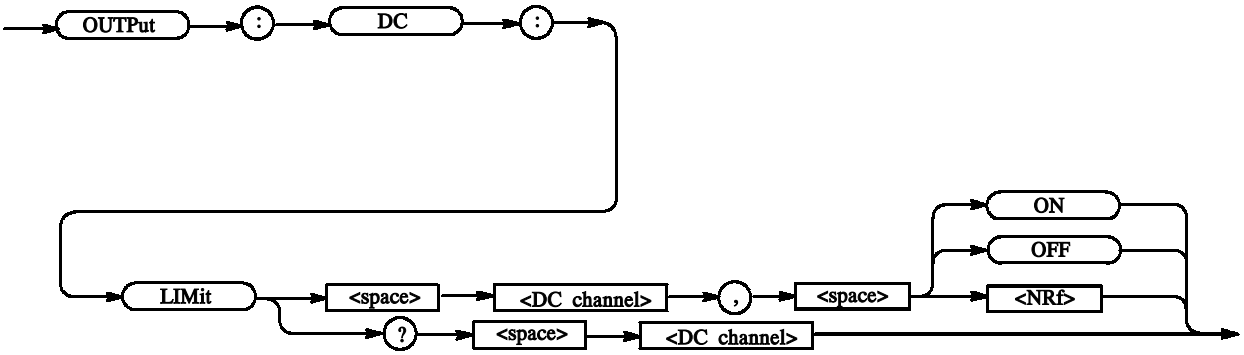
Examples OUTPut:DC:LEVel 0,1.1
Sets the level of the DC output (0) to 1.1 V.

OUTPut:DC:LIMit(?)

This command sets the DC output level.

Syntax OUTPut:DC:LIMit <DC_channel>, { ON | OFF | <NRf> }

OUTPut:DC:LIMit? <DC_channel>



Arguments <DC_channel> ::= <NR1> (0 to 23, if three units are concurrently used)

OFF or <NRf> = 0 – Turns off the DC output limit.

ON or <NRf> ≠ 0 – Turns on the DC output limit.

*RST returns the setting to 0.

Returns <NR1>

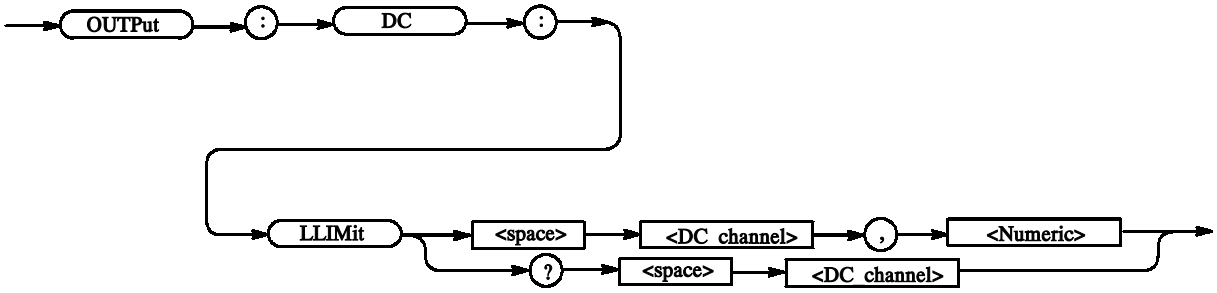
Examples OUTPut:DC:LIMit 1,ON
Turns on the DC output (0) limit.

OUTPut:DC:LLIMit(?)

This command sets the low limit of the DC output.

Syntax

```
OUTPut:DC:LLIMit <DC_channel>, <Numeric>
OUTPut:DC:LLIMit? <DC_channel>
```



Arguments

<DC_channel> ::= <NR1> (0 to 23, if three units are concurrently used)

<Numeric> ::= Low limit - Step: 30 mV, Range: -3 to 5V

If the high limit of the DC output is below its low limit, the high limit will be set to the same value as the low limit.

*RST returns the setting to 0 V.

Returns

<NR3>

Examples

```
OUTPut:DC:LLIMit 0,-1
```

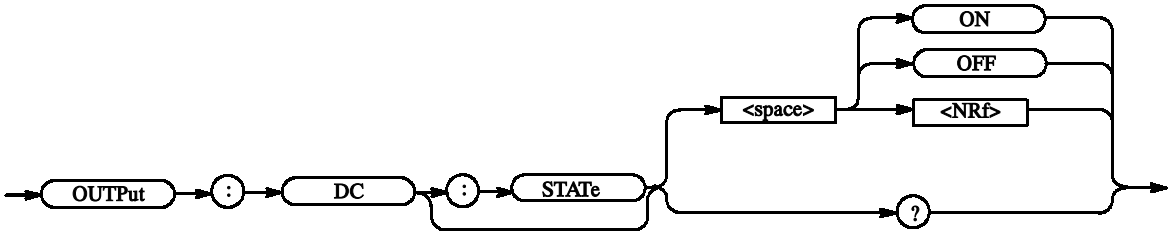
Sets the low limit of the DC output (0) to -1 V.

OUTPut:DC[:STATe] (?)

This command turns on or off the clock output.

Syntax `OUTPut:DC[:STATe] {ON | OFF | <NRf>}`

`OUTPut:DC[:STATe]?`



Arguments `OFF` or `<NRf>` = 0 – Turns off the DC output.

`ON` or `<NRf>` ≠ 0 – Turns on the DC output.

*RST returns the setting to 0 (Off).

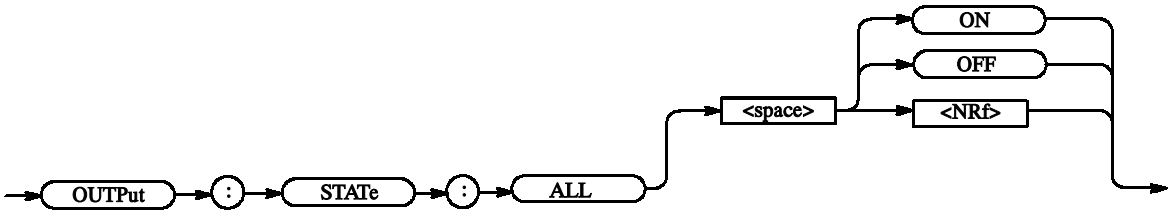
Returns `<NR1>`

Examples `OUTPut:DC:STATe ON`
Turns on the DC output.

OUTPut:STATe:ALL (No Query Form)

This command turns on or off all of the outputs (all assigned outputs, clock output, DC output).

Syntax `OUTPut:STATe:ALL {ON | OFF | <NRf>}`



Arguments `OFF` or `<NRf>` = 0 – Turns off the outputs.

ON or <NRf> ≠ 0 – Turns on the outputs.

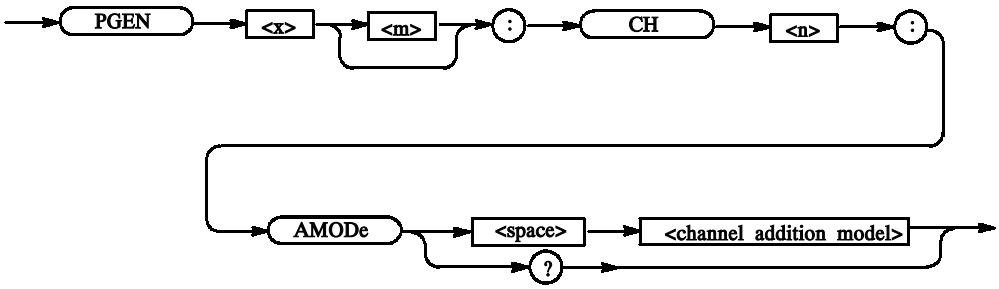
Examples OUTPut:STATe:ALL ON
Turns on the all outputs.

PGEN<x>[<m>]:CH<n>:AMODe (?)

This command sets the channel composition mode of the data output for the specified channel.

Syntax PGEN<x>[<m>]:CH<n>:AMODe <channel_addition_mode>

PGEN<x>[<m>]:CH<n>:AMODe?



Arguments <channel_addition_mode> ::= {NORMal | XOR | AND}

NORMal : Does not perform addition.

XOR : Uses exclusive OR for composition.
(You can select only a channel of an odd number.)

AND : Uses logical sum for composition.
(You can select only a channel of an even number.)

*RST returns the setting to NORMal.

Returns <channel_addition_mode>

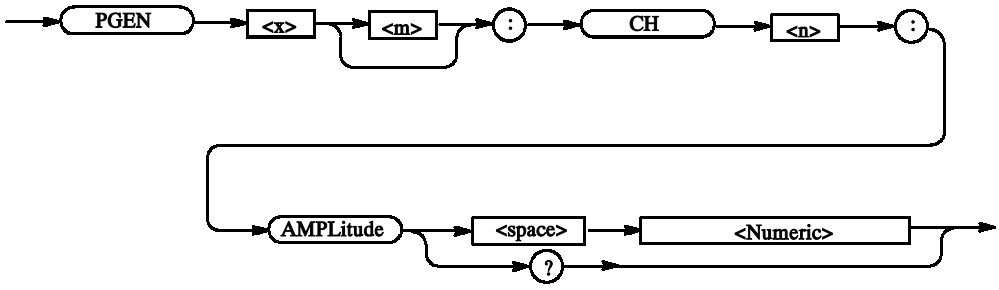
Examples PGENA:CH2:AMODE AND
Composes Mainframe 1, Slot A, Channel 1 and Channel 2 in the AND mode.

PGEN<x>[<m>]:CH<n>:AMPLitude(?)

This command sets the amplitude of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:AMPLitude <Numeric>
PGEN<x>[<m>]:CH<n>:AMPLitude?
```



Arguments

Range: 0.1 to 3.5V

Step: 5 mV

*RST returns the setting to 1 V.

Returns

<NR3>

Examples

```
PGENA:CH2:AMPLitude 1.2
```

Sets the amplitude of Mainframe 1, Slot A, Channel 2 to 1.2 V.

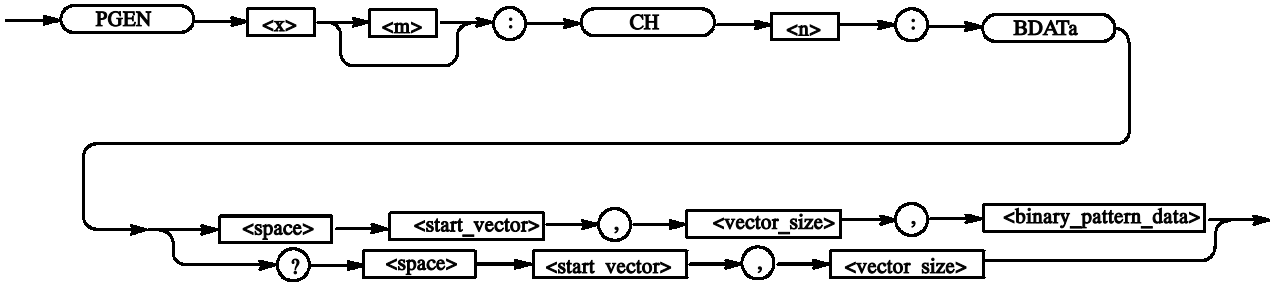
PGEN<x>[<m>]:CH<n>:BDATa(?)

This command transfers pattern data of the specified channel in binary format.

Syntax

```
PGEN<x>[<m>]:CH<n>:BDATa <start_vector>, <vector_size>,
<binary_pattern_data>
```

```
PGEN<x>[<m>]:CH<n>:BDATa? <start_vector>, <vector_size>
```



Arguments

<start_vector> ::= <NR1> - Start address of data

<vector_size> ::= <NR1> - Data size

<binary_pattern_data> ::= <block data> - Binary byte block

***NOTE**. Pattern data size is less than 1 MB (1024 x 1024).*

Returns

<binary_pattern_data>

Examples

PGENB1:CH2:BDATa 0,14,#12F9
This data contains the following:
#: Start character of the block
1: Indicates that the length in the length field is "1".
2: Indicates that the length of the data is "2".
F: 01000110
9: 00111001
Therefore, the data for 14 vectors is set to 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, and 1, beginning at the head of Mainframe 1, Slot B, Channel 2.

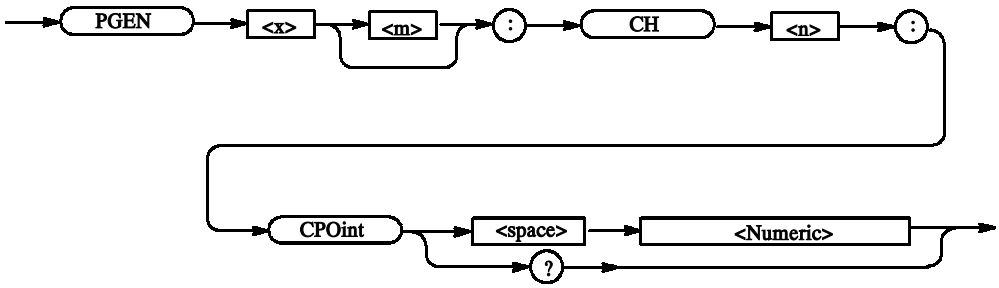
PGENB1:CH2:BDATa? 2,10
Reads the data for 10 vectors from Address 2 of Mainframe 1, Slot B, Channel 2.

PGEN<x>[<m>]:CH<n>:CPOint(?)

This command sets the Cross Point of the NRZ data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:CPOint <Numeric>
PGEN<x>[<m>]:CH<n>:CPOint?
```



Arguments

Range: 30% to 70%

Step: 2%

*RST returns the setting to 50%.

Returns

<NR3>

Examples

```
PGENA:CH1:CPOint 30
```

Sets the Cross Point of Mainframe 1, Slot A, Channel 1 to 30%.

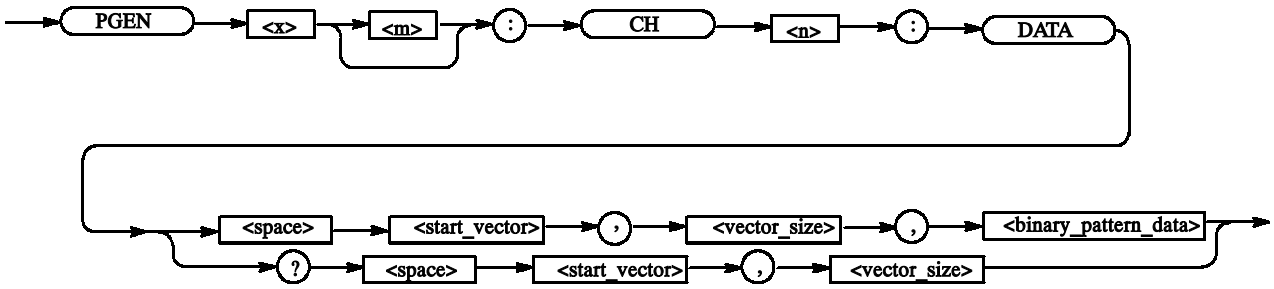
PGEN<x>[<m>]:CH<n>:DATA(?)

This command transfers pattern data of the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:DATA <start_vector>, <vector_size>,
<ascii_pattern_data>
```

```
PGEN<x>[<m>]:CH<n>:DATA? <start_vector>, <vector_size>
```



Arguments

<start_vector> ::= <NR1> - Start address of data

<vector_size> ::= <NR1> - Data size

<ascii_pattern_data> ::= <string> - Data string

NOTE. *Pattern data size is less than 1 MB (1024 x 1024).*

Returns

<binary_pattern_data>

Examples

```
PGENB:CH2:DATA 0,16,"0100011100111001"
```

Sets the data for 16 vectors, from Address 0, to the following in Mainframe 1, Slot B, Channel 2: 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, and 1.

```
PGENB:CH2:DATA? 2,10
```

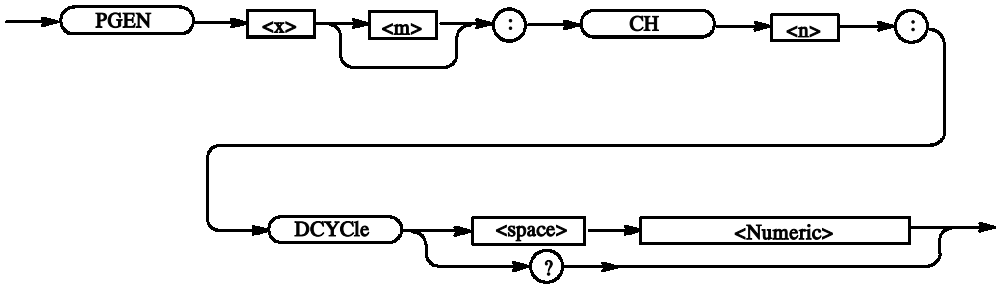
Reads the data for 10 vectors from Address 2 of Mainframe 1, Slot B, Channel 2.

PGEN<x>[<m>]:CH<n>:DCYCle(?)

This command sets the duty cycle of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:DCYCle <Numeric>
PGEN<x>[<m>]:CH<n>:DCYCle?
```



Arguments

Range: Greater than 0%, Less than 100%

For DG mode (Long Delay Off):
The pulse width must be from 290 ps to (period - 290 ps).

For DG mode (Long Delay On):
The pulse width must be from 290 ps to (period - 290 ps).

PG mode:
The pulse width must be from 290 ps to (period x pulse rate - 290 ps).

Step: 0.1%

*RST returns the setting to 50%.

Returns

<NR3>

Examples

```
PGENB:CH2:DCYCle 1
```

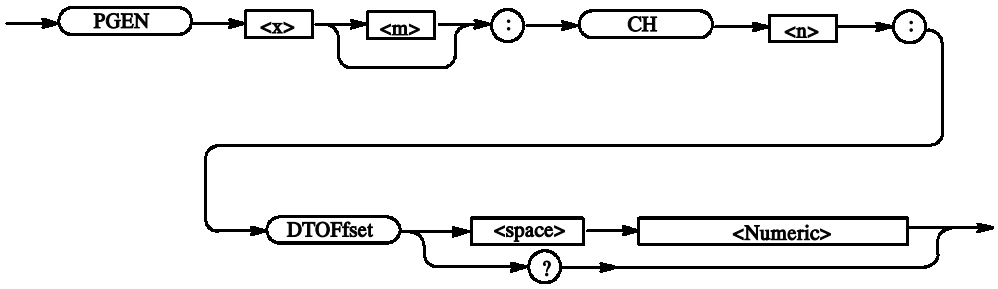
Sets the duty cycle of Mainframe 1, Slot B, Channel 2 to 1%.

PGEN<x>[<m>]:CH<n>:DTOFfset(?)

This command sets the differential timing offset of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:DTOFfset <Numeric>
PGEN<x>[<m>]:CH<n>:DTOFfset?
```



Arguments

Range: -1 to 1 ns

Step:

DTG 5078: 1 ps

DTG 5274: 0.2 ps

*RST returns the setting to 0.0 s.

Returns

<NR3>

Examples

```
PGENB:CH2:DTOFfset 1ps
```

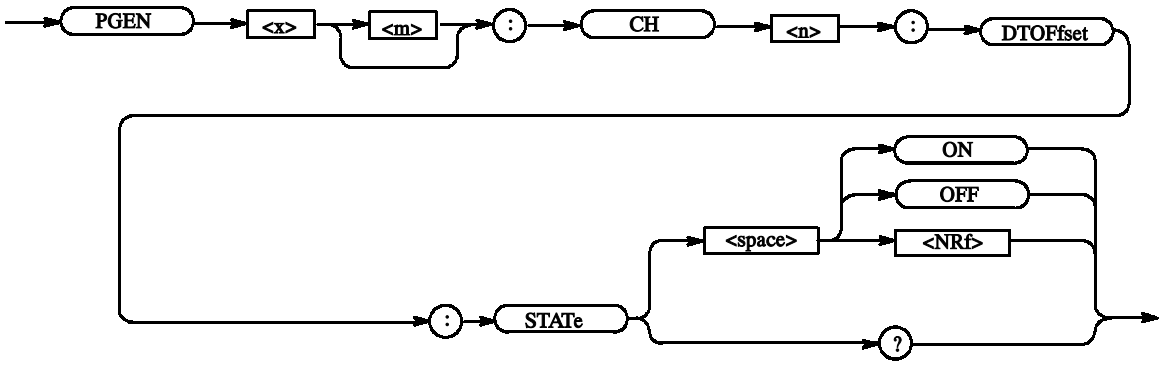
Sets 1 ps for the differential timing offset value of the data output of Mainframe 1, Slot B, Channel 2.

PGEN<x>[<m>]:CH<n>:DTOFfset:STATe(?)

This command turns on or off the differential timing offset for the data output of the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:DTOFfset:STATe { ON | OFF | <NRf> }
PGEN<x>[<m>]:CH<n>:DTOFfset:STATe?
```



Arguments

OFF or <NRf> = 0 – Turns off the differential timing offset.

ON or <NRf> ≠ 0 – Turns on the differential timing offset.

*RST returns the setting to 0 (Off).

Returns

<NR1>

Examples

```
PGENA:CH1:DTOFset:STATe ON
```

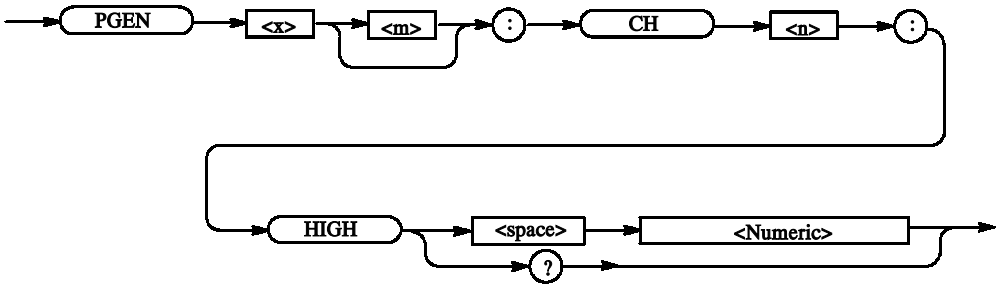
Turns on the differential timing offset of Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:HIGH(?)

This command sets the high level of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:HIGH <Numeric>
PGEN<x>[<m>]:CH<n>:HIGH?
```



Arguments

Step: 5 mV

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 1.0 V.

Returns

<NR3>

Examples

```
PGENB:CH2:HIGH 1.05
```
Sets 1.05 V for the high level of the data output of Mainframe 1, Slot B, Channel 2.

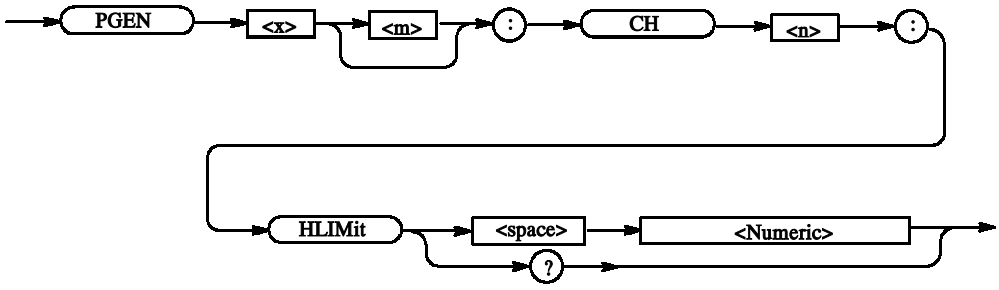
```
PGENB:CH2:HIGH? MAX
```
Query the maximum high level of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:HLIMit(?)

This command sets the high limit of the data output.

Syntax

```
PGEN<x>[<m>]:CH<n>:HLIMit <Numeric>
PGEN<x>[<m>]:CH<n>:HLIMit?
```



Arguments

Step: 5 mV

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 1.0 V.

Returns

<NR3>

Examples

```
PGENB:CH2:HLIMit 1.05
```
Sets 1.05 V for the high limit of the data output of Mainframe 1, Slot B, Channel 2.

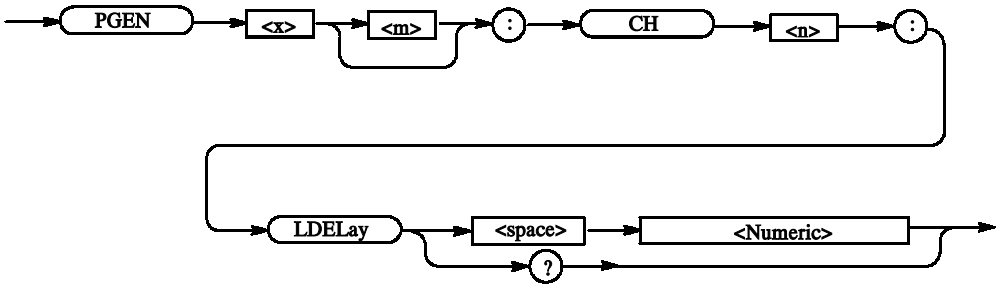
```
PGENB:CH2:HLIMit? MAX
```
Query the maximum high limit of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:LDELay(?)

This command sets the lead delay of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:LDELay <Numeric>
PGEN<x>[<m>]:CH<n>:LDELay?
```



Arguments

Step:

DTG 5078: 1 ps
DTG 5274: 0.2 ps

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.0 s.

Returns

<NR3>

Examples

```
PGENB:CH2:LDELay 1ps
```
Sets 1 ps for the lead delay of the data output of Mainframe 1, Slot B, Channel 2.

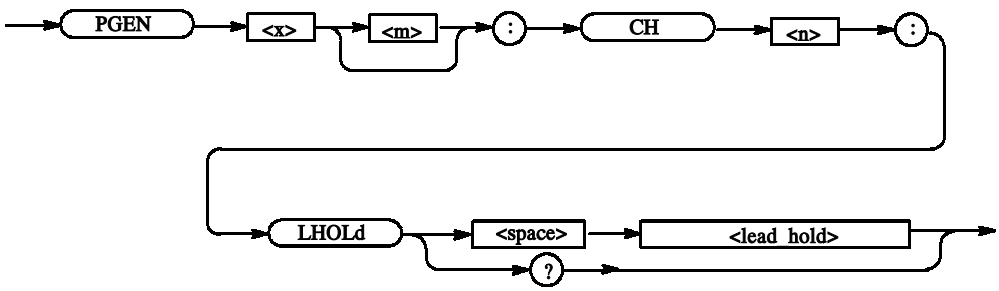
```
PGENB:CH2:LDELay? MAX
```
Query the maximum lead delay of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:LHOLd(?)

This command specifies how to hold the leading edge of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:LHOLd <lead_hold>
PGEN<x>[<m>]:CH<n>:LHOLd?
```



Arguments

<lead_hold> ::= { LDELay | PHASe }

LDELay: Lead delay
PHASe: Phase

Note: Phase = lead delay / period x 100 (%)

*RST returns the setting to LDELay.

Returns

<lead_hold>

Examples

```
PGENA:CH1:LHOLd PHAse
```

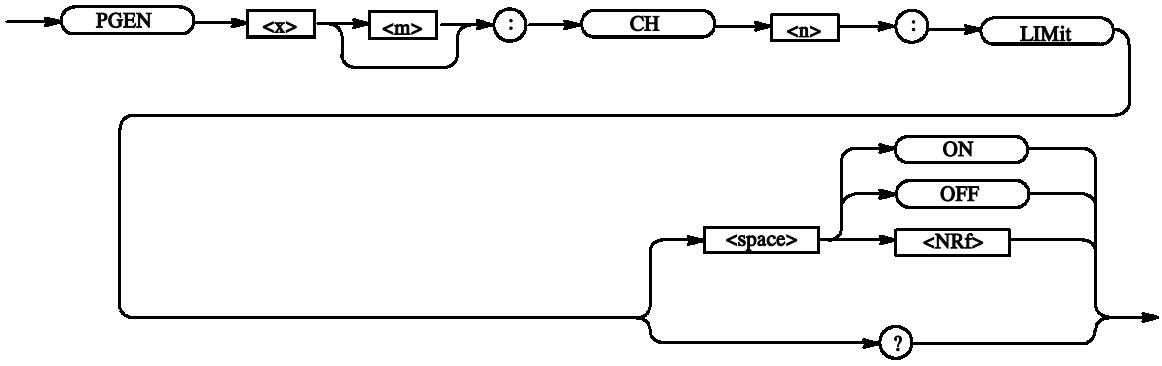
Sets phase for how to hold the leading edge of the data output of Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:LIMit(?)

This command sets whether the limit for the specified channel is applied.

Syntax `PGEN<x>[<m>]:CH<n>:LIMit { ON | OFF | <NRf> }`

`PGEN<x>[<m>]:CH<n>:LIMit?`



Arguments `OFF` or `<NRf>` = 0 – Turns off the limit.

`ON` or `<NRf>` ≠ 0 – Turns on the limit.

`*RST` returns the setting to 0.

Returns `<NR1>`

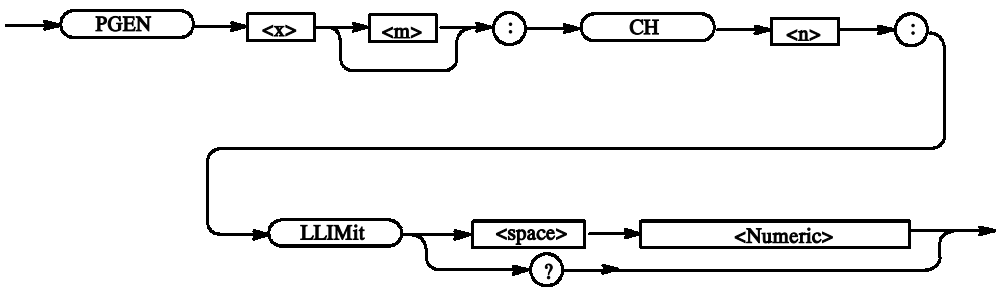
Examples `PGENA:CH1:LIMit ON`
Applies the limit to Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:LLIMit(?)

This command sets the low limit of the data output level for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:LLIMit <Numeric>
PGEN<x>[<m>]:CH<n>:LLIMit?
```



Arguments

Step: 5 mV

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.0 V.

Returns

<NR3>

Examples

```
PGENB:CH2:LLIMit? MAX
```

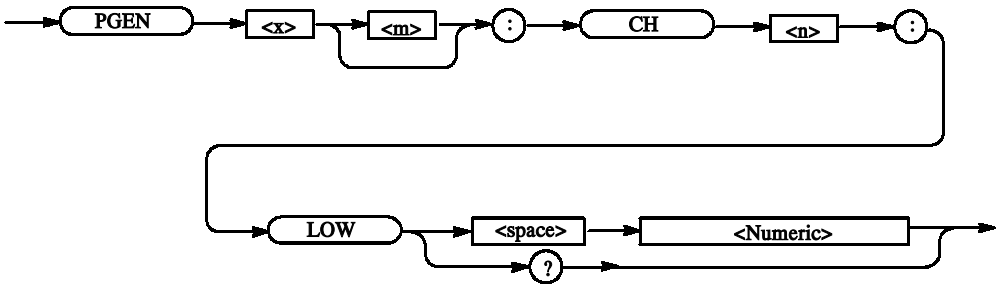
Query the maximum low limit of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:LOW(?)

This command specifies the low level of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:LOW <Numeric>
PGEN<x>[<m>]:CH<n>:LOW?
```



Arguments

Step: 5 mV

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.0 V.

Returns

<NR3>

Examples

```
PGENB:CH2:LOW MIN
```

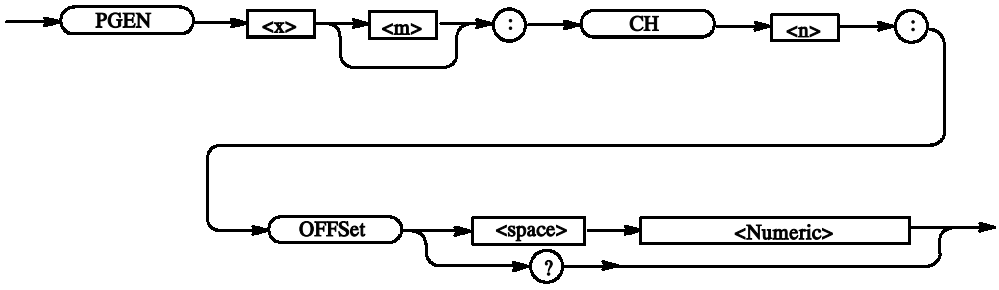
Set the minimum low level of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:OFFSet(?)

This command sets the offset level of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:OFFSet <Numeric>
PGEN<x>[<m>]:CH<n>:OFFSet?
```



Arguments

Step: 5 mV

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: “Output Level” Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.5 V.

Returns

<NR3>

Examples

```
PGENB:CH2:OFFSet 0.6
```
Sets 0.6 V for the offset level of the data output of Mainframe 1, Slot B, Channel 2.

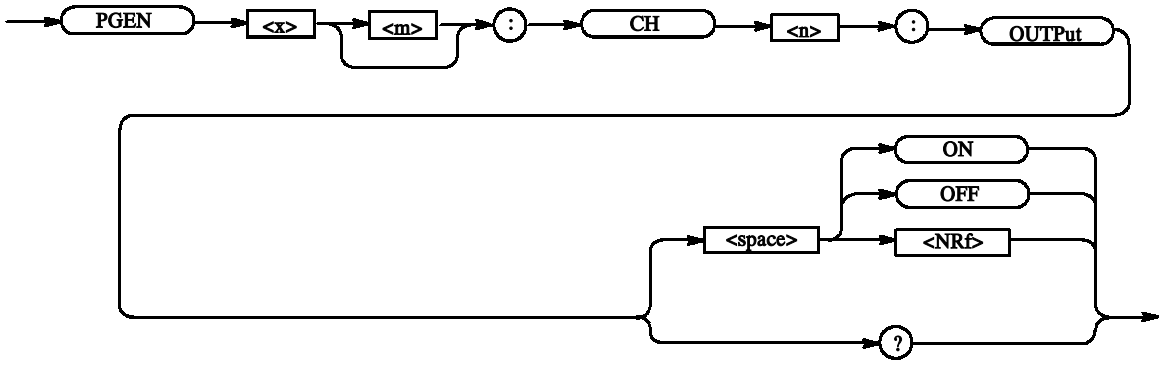
```
PGENB:CH2:OFFSet? MAX
```
Query the maximum offset level of the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:OUTPut(?)

This command turns on or off the data output of the specified channel.

Syntax PGEN<x>[<m>]:CH<n>:OUTPut { ON | OFF | <NRf> }

PGEN<x>[<m>]:CH<n>:OUTPut?



Arguments OFF or <NRf> = 0 – Turns off the data output.

ON or <NRf> ≠ 0 – Turns on the data output.

*RST returns the setting to 0.

Returns <NR1>

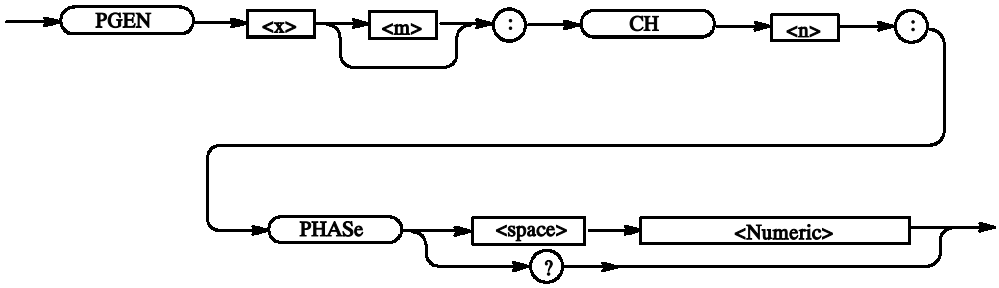
Examples PGENA:CH1:OUTPut ON
Sets ON the data output for Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:PHASe(?)

This command sets the phase for the data output of the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:PHASe <Numeric>
PGEN<x>[<m>]:CH<n>:PHASe?
```



Arguments

Both the lead delay and phase indicate the position of a pulse leading edge. They differ only in the manner of representation, and is identical in the "substantial" range of setting.

Step: 0.1%

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.0 %.

Returns

<NR3>

Examples

```
PGENB:CH2:PHASe 1
```
Sets 1% for the phase for the data output of Mainframe 1, Slot B, Channel 2.

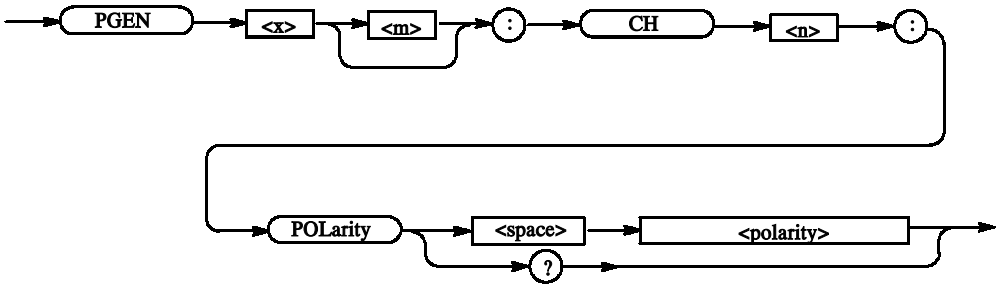
```
PGENB:CH2:PHASe? MAX
```
Query the maximum phase for the data output of Mainframe 1, Slot B, Channel 2 at the current.

PGEN<x>[<m>]:CH<n>:POLarity(?)

This command sets the polarity of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:POLarity <polarity>
PGEN<x>[<m>]:CH<n>:POLarity?
```



Arguments

<polarity> ::= { NORMal | INVert }

NORMal : Sets the polarity to positive.
INVert : Sets the polarity to negative.

*RST returns the setting to NORMal.

Returns

<polarity>

Examples

```
PGENA:CH1:POLarity INVert
```

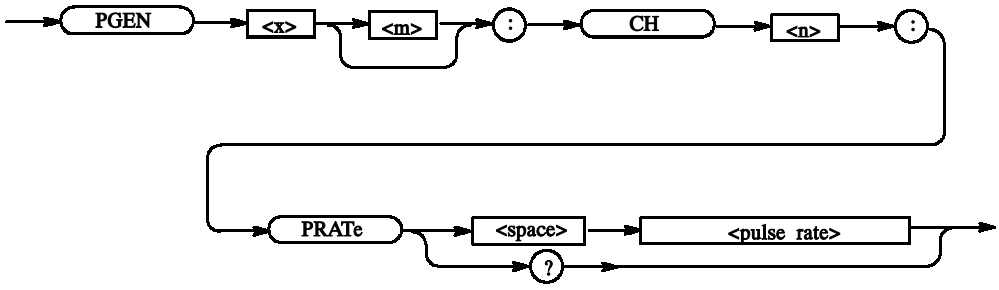
Sets the polarity of data output to negative for Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:PRATe(?)

This command sets the pulse rate of the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:PRATe <pulse_rate>
PGEN<x>[<m>]:CH<n>:PRATe?
```



Arguments

<pulse_rate> ::= { NORMal | HALF | QUARter | EIGHth | SIXTeenth | OFF }

NORMal	: Sets the pulse rate to normal.
HALF	: Sets the pulse rate to 1/2.
QUARter	: Sets the pulse rate to 1/4.
EIGHth	: Sets the pulse rate to 1/8.
SIXTeenth	: Sets the pulse rate to 1/16.
OFF	: Turns off the pulse rate.

*RST returns the setting to NORMal.

Returns

<pulse_rate>

Examples

```
PGENA:CH1:PRATe HALF
```

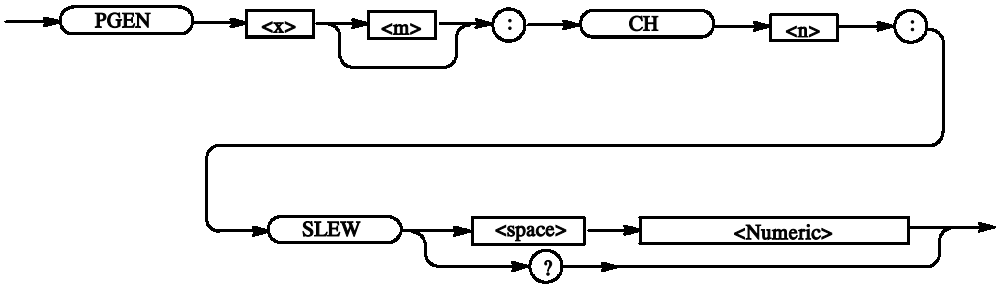
Sets the pulse rate for Mainframe 1, Slot A, Channel 1 to 1/2.

PGEN<x>[<m>]:CH<n>:SLEW(?)

This command sets the slew rate of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:SLEW <Numeric>
PGEN<x>[<m>]:CH<n>:SLEW?
```



Arguments

Step: 0.1 V/ns

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: “Output Level” Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 2.25 V/ns.

Returns

<NR3>

Examples

```
PGENA:CH1:SLEW 5.1
```

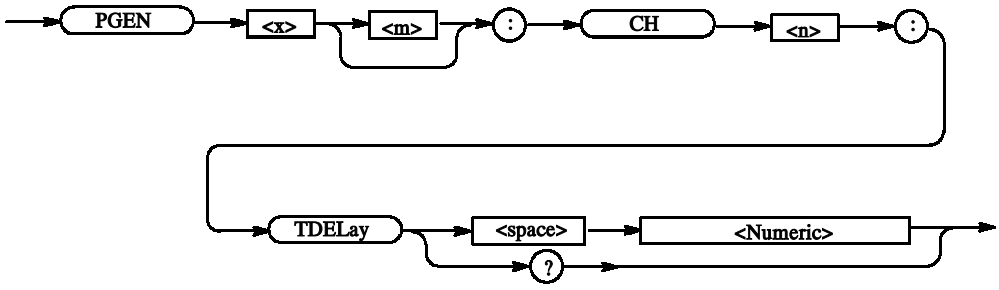
Sets the slew rate for Mainframe 1, Slot A, Channel 1 to 5.1 V/ns.

PGEN<x>[<m>]:CH<n>:TDELay(?)

This command sets the trail delay of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:TDELay <Numeric>
PGEN<x>[<m>]:CH<n>:TDELay?
```



Arguments

Step: 5 ps

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 5e-9 s.

Returns

<NR3>

Examples

```
PGENA:CH1:TDELay 0.5ns
```
Sets 0.5ns for the trail delay of the data output of Mainframe 1, Slot A, Channel 1.

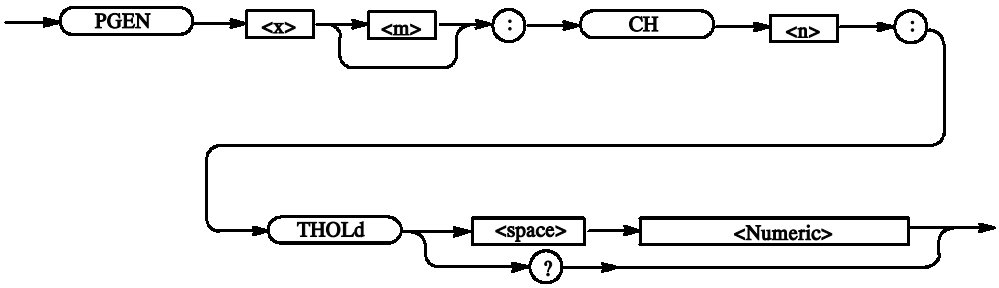
```
PGENA:CH1:TDELay? MAX
```
Query the maximum trail delay of the data output of Mainframe 1, Slot A, Channel 1 at the current.

PGEN<x>[<m>]:CH<n>:THOLd(?)

This command specifies how to hold the trailing edge of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:THOLd <trail_hold>
PGEN<x>[<m>]:CH<n>:THOLd?
```



Arguments

<trail_hold> ::= {TDELay | DCYCle | WIDTh}

TDELay: Sets TDELay for how to hold the trailing edge.
DCYCle: Sets DCYCle for how to hold the trailing edge.
WIDTh: Sets WIDTh for how to hold the trailing edge.

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to DCYCle.

Returns

<trail_hold>

Examples

```
PGENA:CH1:THOLd TDELay
```

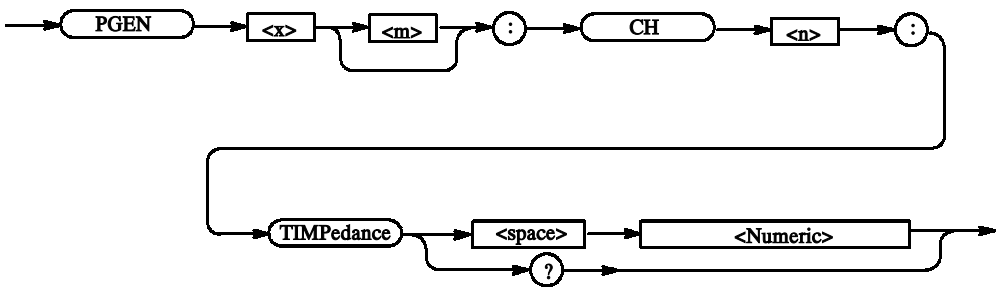
Sets TDELay for how to hold the trailing edge for Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:CH<n>:TIMPedance(?)

This command sets the termination impedance of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:TIMPedance <Numeric>
PGEN<x>[<m>]:CH<n>:TIMPedance?
```



Arguments

Range: 10 ohm to 1 Mohm,

≤ 0: Open

For Open, the response -1 will be returned.

Step: 3 significant digits. The minimum resolution is 1 ohm.

*RST returns the setting to 50 ohm.

Returns

<NR3>

Examples

```
PGENA:CH1:TIMPedance -1
```

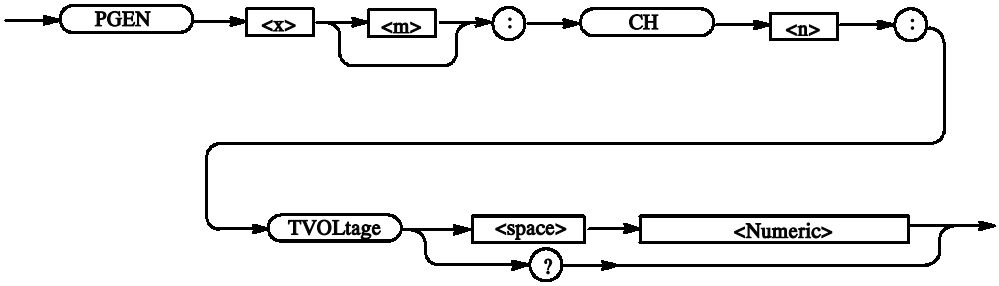
Sets the termination impedance for Mainframe 1, Slot A, Channel 1 to Open.

PGEN<x>[<m>]:CH<n>:TVOLtage(?)

This command sets the termination voltage of the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:TVOLtage <Numeric>
PGEN<x>[<m>]:CH<n>:TVOLtage?
```



Arguments

Range: -2 to +5V

Step: 0.1 V

*RST returns the setting to 0.0 V.

Returns

<NR3>

Examples

```
PGENA:CH1:TVOLtage 1
```

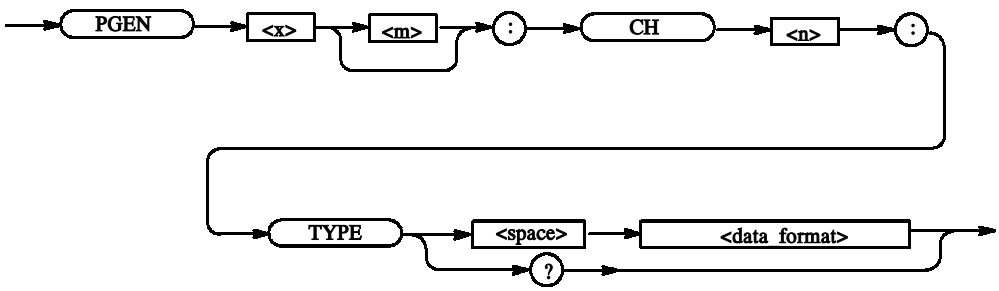
Sets the termination voltage for Mainframe 1, Slot A, Channel 1 to 1 V.

PGEN<x>[<m>]:CH<n>:TYPE(?)

This command sets the format of data output for the channel specified in DG mode.

Syntax

```
PGEN<x>[<m>]:CH<n>:TYPE <data_format>
PGEN<x>[<m>]:CH<n>:TYPE?
```



Arguments

```
<data_format> ::= {NRZ | RZ | R1}
```

NRZ : Sets the signal to “NRZ format”.
RZ : Sets the signal to “RZ format”.
R1 : Sets the signal to “R1 format”.

*RST returns the setting to NRZ.

Returns

```
<data_format>
```

Examples

```
PGENA:CH1:TYPE R1
```

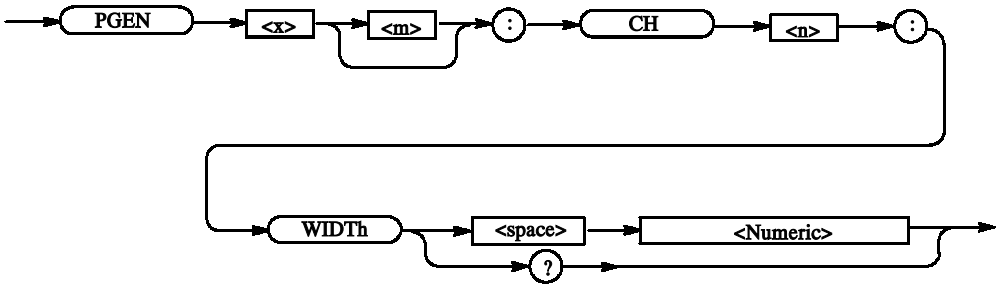
In DG mode, sets the format for Mainframe 1, Slot A, Channel 1 to “Return to 1”.

PGEN<x>[<m>]:CH<n>:WIDTh(?)

This command sets the pulse width for the data output for the specified channel.

Syntax

```
PGEN<x>[<m>]:CH<n>:WIDTh <Numeric>
PGEN<x>[<m>]:CH<n>:WIDTh?
```



Arguments

Step: 5 ps

Can be obtained using the following conversion expression from the range of trail delay or duty.

Pulse width = duty x (period x pulse rate) / 100

Or

Pulse width = trail delay - lead delay

*RST returns the setting to 5e-9 s.

Returns

<NR3>

Examples

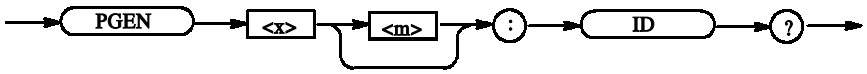
```
PGENA:CH1:WIDTh 6e-9
```

Sets 6e-9 for the pulse width for Mainframe 1, Slot A, Channel 1.

PGEN<x>[<m>]:ID? (Query Only)

This command examines what module the specified slot contains.

Syntax `PGEN<x>[<m>]:ID?`



Arguments None

Returns `<opt>`

-1: No module
1: DTGM10
2: DTGM20
3: DTGM30

Examples `PGENB:ID?`
Examines the module contained in Mainframe 1, Slot B.
If no module is contained, the following will be returned: -1

*RST (No Query Form)

This command resets the data timing generator to the default state. This command has no effect on the network and communication settings, such as GPIB or IP address. Refer to *Appendix C: Factory Initialization Settings*.

Syntax *RST

Arguments None

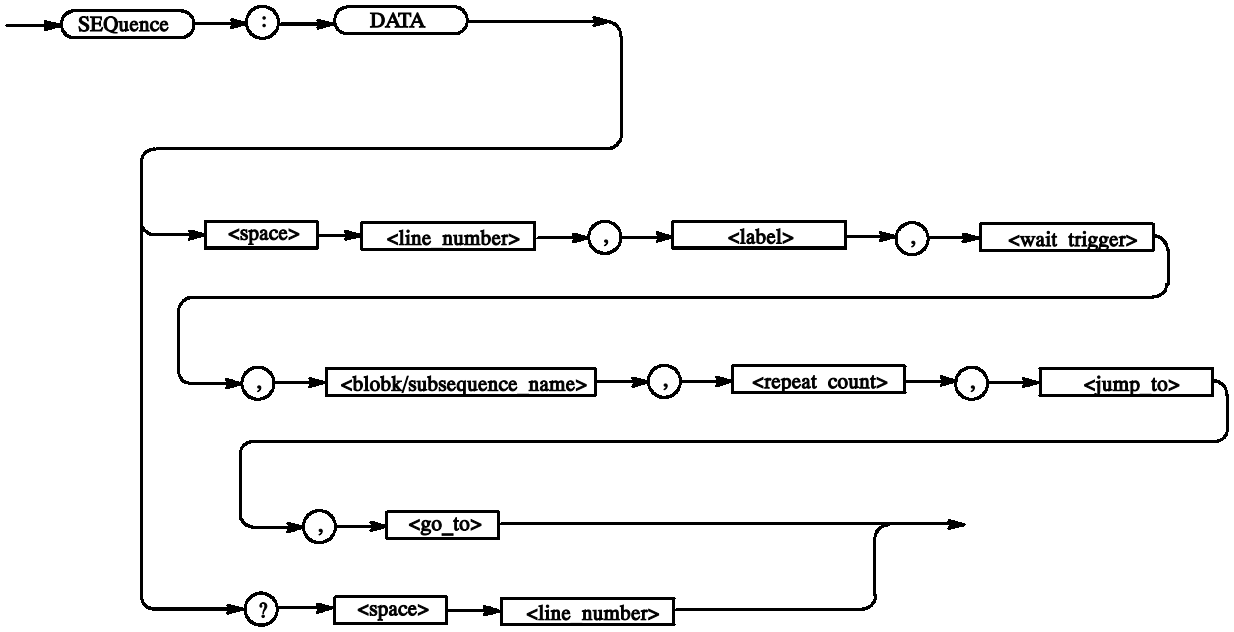
Examples *RST
resets the instrument.

SEQuence:DATA(?)

This command sets the data corresponding to one line of a sequence.

Syntax `SEQuence:DATA <line_number>, <label>, <wait_trigger>, <block/subsequence_name>, <repeat_count>, <jump_to>, <go_to>`

`SEQuence:DATA? <line_number>`



Arguments `<line_number>` ::= <NR1> - Begins at 0.

`<label>` ::= <string> - 16 characters or less

`<wait_trigger>` ::= { ON | OFF | <NRf> }

`<block/subsequence_name>` ::= <string> - 32 characters or less

`<repeat_count>` ::= <NR1> - 1 to 65536
Zero (0) causes an endless loop.

`<jump_to>` ::= <string> - Destination to which control jumps when an event occurs during output of this line.

`<go_to>` ::= <string> - Destination to which control jumps unconditionally after output of this line.

The response following *RST:
`"", 0, "Block1", 0, "", "".`

Returns `<label>, <wait_trigger>, <block/subsequence_name>, <repeat_count>, <jump_to>, <go_to>`

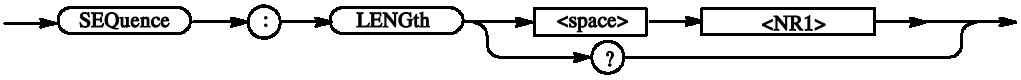
Examples `SEQuence:DATA 0, "", OFF, "", 1, "", "Label2"`
Indicates the following: line number 0, label "", weight trigger Off, block name "", number of repetitions 1, destination of jump "", and goto "Label2".

SEQuence:LENGth(?)

This command changes the sequence length.

Syntax `SEQuence:LENGth <NR1>`

`SEQuence:LENGth?`



Arguments Range: 1 to 8000
When the length is increase, the content is indefinite.

*RST returns the setting to -1.

Returns `<NR1>`

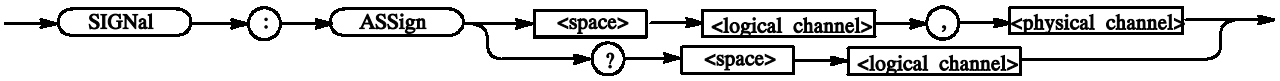
Examples `SEQuence:LENGth 1000`
Sets the sequence length to 1000 lines.

SIGNal:ASSign(?)

This command assigns a physical channel to a logical channel.

Syntax

```
SIGNal:ASSign <logical_channel>, <physical_channel>
SIGNal:ASSign? <logical_channel>
```



Arguments

<logical_channel>:

<group_name> For a group with a 1-bit width
<group_name>[<bit>] Specified bit number in the specified group
Example:
CLK
Addr[0]

<physical_channel>

Use the mainframe number, slot name, and channel number to specify the physical channel.
"1A4" indicates Mainframe 1, Slot A, Channel 4.
If you specify "", the assignment will be reset.

*RST resets the assignment.

Returns

<physical_channel>

Examples

```
SIGNal:ASSign "Addr1","1B4"
```

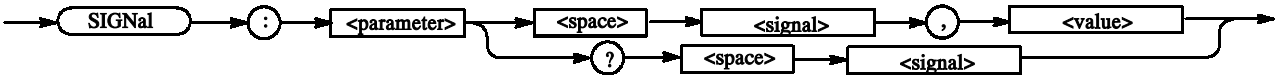
Assigns the physical channel of Mainframe 1, Slot B, Channel 4 to the logical channel of group name Addr, bit number 1.

SIGNal:<parameter>(?)

This command sets various data output parameters using a signal name.

Syntax

```
SIGNal:<parameter> <signal>, <value>
SIGNal:<parameter>? <signal>
```



Arguments

<parameter> ::= {AMODe | AMPLitude | CPOint | DCYCle | DTOFfset | DTOFfset:STATe | HIGH | HLIMit | LDELay | LHOLd | LIMit | LLIMit | LOW | OFFSet | OUTPut | PHASe | POLarity | PRATe | SLEW | TDELay | THOLd | TIMPedance | TVOLtage | TYPE | WIDTh}

<signal> ::= logical channel or bus

For example:

Addr[]
Addr[0:3]
Addr[0..3]
Addr[3..0]

NOTE. *If you omit the contents of the brackets, Addr[<msb>:<lsb>] will be assumed. (For example, Addr is 8 bit wide, Addr[] will be assumed to be Addr[7:0].)*
<value> varies with <parameter>.
For more specific information, see the PGEN<x>[<m>]:CH<n> command section.

If you query a number of channels, the first channel's value will be returned. For example, the SIGN:HIGH? "DATA[2..4]" command returns a value of DATA[2].

Returns

<value>

Examples

```
SIGNal:AMPLitude "Addr[1]",1.1
```
Sets 1.1 V for the amplitude for the channel specified with Addr[1].

```
SIGNal:TYPE "Addr2",R1
```
Sets R1 for the format of the data output for the channel specified with Addr2.

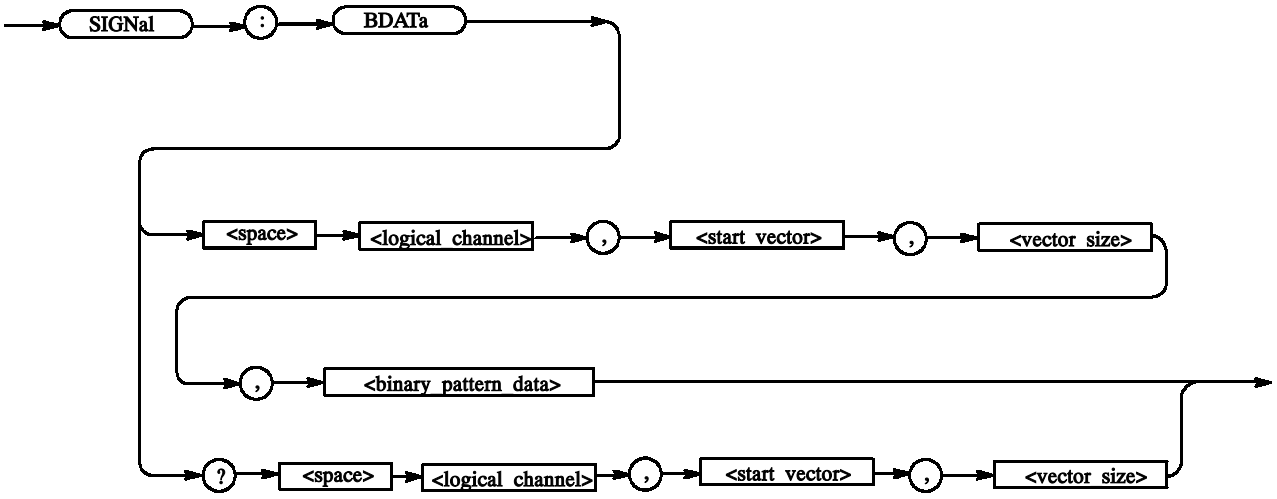
SIGNal:BDATa(?)

This command transfers pattern data in binary format.

Syntax

```
SIGNal:BDATa <logical_channel>, <start_vector>, <vector_size>,
<binary_pattern_data>

SIGNal:BDATa? <logical_channel>, <start_vector>, <vector_size>
```



Arguments

<logical_channel> ::= logical channel assigned with SIGNal:ASSign

<start_vector> ::= start address of data

<vector_size> ::= data size

<binary_pattern_data> ::= binary byte block

NOTE. *Pattern data size is less than 1 MB (1024 x 1024).*

Returns

<binary_pattern_data>

Examples

```
SIGNal:BDATa  "Addr[1]",0,14,#12F9
```

This data contains the following:

#: Start character of the block

1: Indicates that the length in the length field is “1”.

2: Indicates that the length of the data is “2”.

F: 01000110

9: 00111001

Therefore, the data for 14 vectors is set to 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, and 1, beginning at the head of the channel specified with Addr[1].

```
SIGNal:BDATa  "Addr[1]",2,10
```

Reads the data for 10 vectors from Address 2 of the channel specified with Addr[1].

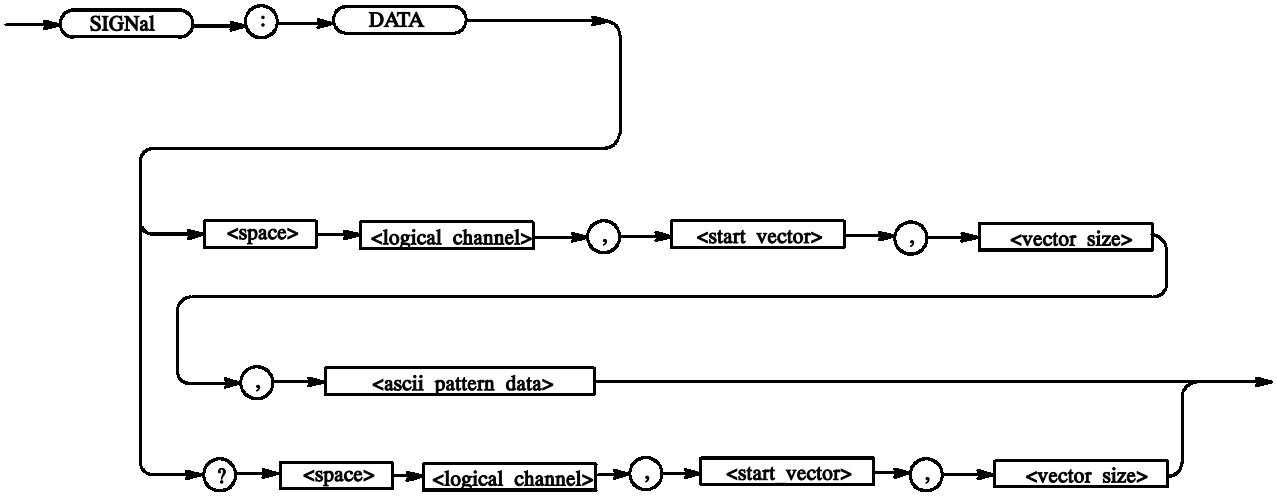
SIGNal:DATA(?)

This command transfers pattern data.

Syntax

```
SIGNal:DATA <logical_channel>, <start_vector>, <vector_size>,
<ascii_pattern_data>
```

```
SIGNal:DATA? <logical_channel>, <start_vector>, <vector_size>
```



Arguments

<logical_channel>::= logical channel assigned with SIGNal:ASSign

<start_vector> ::= start address of data

<vector_size> ::= data size

<ascii_pattern_data> ::= data string

NOTE. *Pattern data size is less than 1 MB (1024 x 1024).*

Returns

<ascii_pattern_data>

Examples

```
SIGNal:DATA "Test[2]",0,16, "0100011100111001"
```

Sets the data for 16 vectors, from Address 0 of the Test[2] channel, to the following: 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, and 1.

```
SIGNal:DATA "Test[2]",2,10
```

Reads the data for 10 vectors from Address 2 of the Test[2] channel.

*SRE (?)

This command sets and queries the bits in the Service Request Enable Register (SRER). For a complete discussion of the use of these registers, refer to the *Status and Events* section of this manual.

Syntax

```
*SRE <NR1>
```

```
*SRE?
```

Arguments

<NR1> is a value in the range from 0 to 255. The binary bits of the SRER are set according to this value. Using an out-of-range value causes an execution error. The power-on default for SRER is 0 if *PSC is 1. If *PSC is 0, the SRER maintains its value through a power cycle.

Examples

```
*SRE 48
```

sets the bits in the SRER to the binary value 00110000.

```
*SRE?
```

might return a value of 32, showing that the bits in the SRER have the binary value 00100000.

*STB? (Query Only)

This command returns the contents of the Status Byte Register (SBR) using the Master Summary Status (MSS) bit. For a complete discussion of the use of these registers, refer to the *Status and Events* section of this manual.

Syntax *STB?

Arguments None

Returns <NR1> indicates that the content of the SBR in a decimal number.

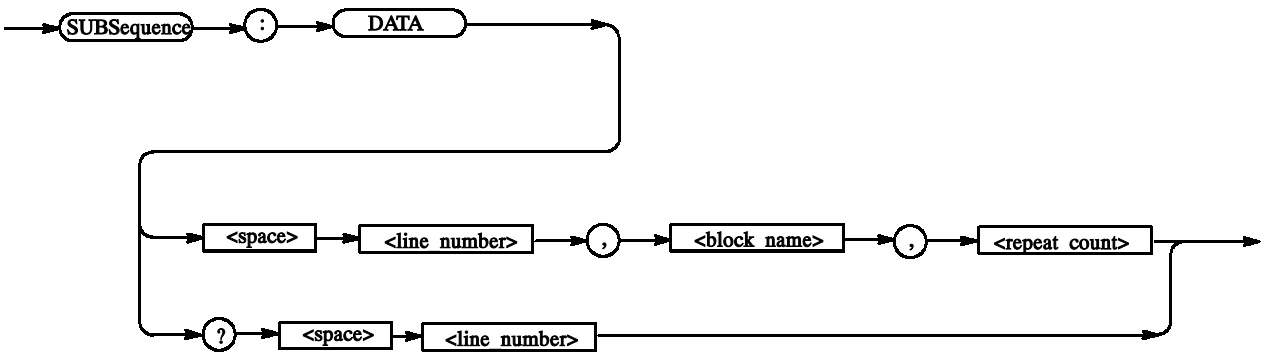
Examples *STB?
might return 96, which indicates that the SBR contains the binary number 0110 0000.

SUBSequence:DATA(?)

This command sets the data corresponding to one line of a subsequence.

Syntax SUBSequence:DATA <line_number>, <block_name>, <repeat_count>

SUBSequence:DATA? <line_number>



Arguments <line_number> ::= <NR1> - Begins at 0.

<block_name> ::= <string> - 32 characters or less

<repeat_count> ::= <NR1> - 1 to 65536

Returns <block_name>, <repeat_count>

Examples SUBSequence:DATA 0, "Block0",1
Sets block name “Block0” and the subsequence of repeat count 1 on line 0.

SUBSequence:DELete (No Query Form)

This command deletes a subsequence.

Syntax SUBSequence:DELete <subsequence_name>

Arguments <subsequence_name> ::= <string>

Examples SUBSequence:DELete "Block1"
Deletes a subsequence named “Block1”.

SUBSequence:DELete:ALL (No Query Form)

This command deletes all the subsequences.

Syntax SUBSequence:DELete:ALL

Arguments None

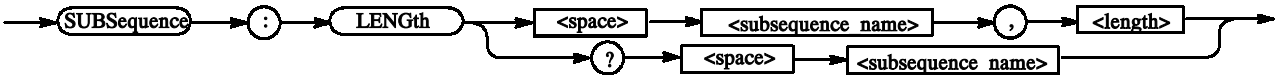
Examples SUBSequence:DELete:ALL
Deletes all the subsequences.

SUBSequence:LENGth(?)

This command changes the subsequence length.

Syntax `SUBSequence:LENGth <subsequence_name>, <length>`

`SUBSequence:LENGth? <subsequence_name>`



Arguments <subsequence_name> ::= <string>

<length> ::= <NR1> - The range is 1 to 256. If the specified subsequence is not found, -1 will be returned.
When the length is increase, the content is indefinite.

*RST returns the setting to -1.

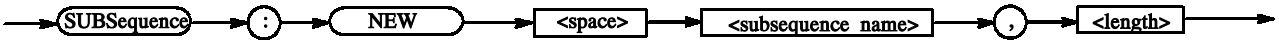
Returns <NR1>

Examples `SUBSequence:LENGth "Block1",128`
Sets the length of subsequence “Block1” to 128.

SUBSequence:NEW (No Query Form)

This command creates the subsequences.

Syntax `SUBSequence:NEW <subsequence_name>, <length>`



Arguments <subsequence_name> ::= <string> - 32 characters or less

<length> ::= <NR1> - The range is 1 to 256.

The content is indefinite.

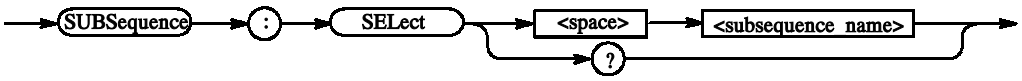
Examples `SUBSequence:NEW "Sub02",100`
Creates a subsequence named “Sub02” with a length of 100.

SUBSequence:SELect(?)

This command selects the subsequence to be set with SUBSequence:DATA.

Syntax SUBSequence:SELect <subsequence_name>

SUBSequence:SELect?



Arguments <subsequence_name> ::= <string>

*RST returns the setting to "".

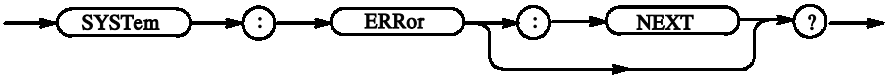
Returns <subsequence_name>

Examples SUBSequence:SELect "Sub03"
Sets "Sub03" for the subsequence to be used with SUBSequence:DATA.

SYSTem:ERRor[:NEXT]? (Query Only)

This command retrieves and returns error data from the Error and Event Queue. For more details, refer to the *Status and Event* section of this manual.

Syntax SYSTem:ERRor[:NEXT]?



Arguments None

Returns <error/event_number>,
"<error/event_description>"
where:
<error/event_number> is an integer between -32768 and 0.
0 indicates that no error or event has occurred.
Negative values are error/event numbers reserved in SCPI standards.

<error/event_description> is a message relating to the error/event number.

This error/event is same as a dialog of the display.

Examples

```
SYSTem:ERRor:NEXT?
```

might return the following response:

```
-141,"Invalid character data;Invalid enumeration - PGENA:CH1:TYPE
NR2"
```

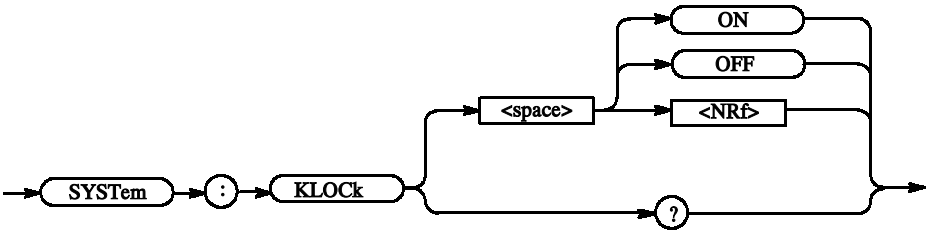
This response indicates that the unit is invalid.

SYSTem:KLOCk (?)

This command locks or unlocks the front panel. Use this command to disable manual operation while the data timing generator is being controlled externally. If the front panel is not explicitly locked out using this command, the data timing generator accepts input from both the external controller and the front panel.

Syntax

```
SYSTem:KLOCk { ON | OFF | <NRf> }
SYSTem:KLOCk?
```



Arguments

OFF or <NRf> = 0 unlocks controls of the front panel.

ON or <NRf> ≠ 0 locks controls of the front panel.

*RST has no effect on the parameter.

Returns

<NR1> = 0 indicates the front panel is unlocked.

<NR1> = 1 indicates the front panel is locked.

Examples

```
SYSTem:KLOCk ON
```

locks the front panel.

```
SYSTem:KLOCk?
```

might return 1, which indicates that the front panel is locked.

SYSTem:VERSion? (Query Only)

This command returns the SCPI version number with the data timing generator complies.

Syntax `SYSTem:VERSion?`

Returns <NR2>::=YYYY.V
where YYYY represents the year version and V represents an approved revision number for that year.

Examples `SYSTem:VERSion?`
might return 1999.0.

TBAS:COUNt(?)

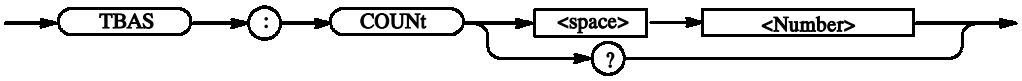
This command sets the burst count.

Syntax `TBAS:COUNt <Numeric>`

`TBAS:COUNt?`

Arguments Range: 1 to 65536.

*RST returns the setting to 1.



Returns <NR1>

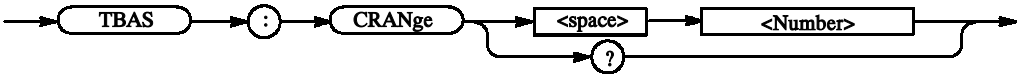
Examples `TBAS:COUNt 10`
Sets the burst count to 10.

TBAS:CRANge(?)

This command sets the clock range.

Syntax

```
TBAS:CRANge <Numeric>
TBAS:CRANge?
```



Arguments

0: 50 to 125 kHz
1: 100 to 250 kHz
2: 200 to 500 kHz
3: 250 to 625 kHz
4: 500 to 1.25 MHz
5: 1 to 2.5 MHz
6: 2 to 5 MHz
7: 2.5 to 6.25 MHz
8: 5 to 12.5 MHz
9: 10 to 25 MHz
10: 20 to 50 MHz
11: 25 to 62.5 MHz
12: 50 to 125 MHz
13: 100 to 250 MHz (100 MHz <, if RZ/R1 exists)
14: 200 M <

*RST returns the setting to 12.

Returns

<NR1>

Examples

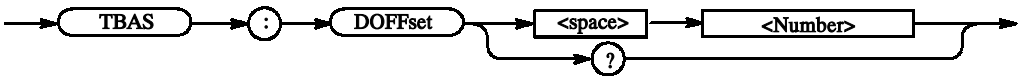
TBAS:CRANge 11
Sets the clock range to “25 to 62.5 MHz”.
TBAS:CRANge?
Queries the clock range.
A response example: 11

TBAS:DOFFset(?)

This command sets the delay offset.

Syntax `TBAS:DOFFset <Numeric>`

`TBAS:DOFFset?`



Arguments Step:
DTG 5078: 1 ps
DTG 5274: 0.2 ps

For the setting range, refer to the reference manual (the calculation is complicated).
The range is difficult to calculate, please refer to the reference manual (Chapter 2 Reference: "Output Level" Section). You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 0.0 s.

Returns <NR3>

Examples `TBAS:DOFFset 1ps`
Sets the delay offset to 1 ps.

TBAS:EIN:IMMediate (No Query Form)

This command generates an event.

Syntax `TBAS:EIN:IMMediate`

Examples `TBAS:EIN:IMMediate`
Generates an event.

TBAS:EIN:IMPedance(?)

This command sets the event input impedance.

Syntax `TBAS:EIN:IMPedance <Numeric>`

`TBAS:EIN:IMPedance?`

Arguments Range: 50 or 1e3

*RST returns the setting to 1e3 ohm.

Returns <NR3>

Examples `TBAS:EIN:IMPedance 50`
Sets the event input impedance to 50 ohm.

TBAS:EIN:LEVel(?)

This command sets the event input level.

Syntax `TBAS:EIN:LEVel <Numeric>`

`TBAS:EIN:LEVel?`

Arguments Range: -5 to +5V

Step: 0.1 V

*RST returns the setting to 1.4 V.

Returns <NR3>

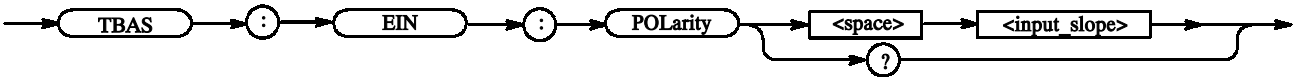
Examples `TBAS:EIN:LEVel 1.1`
Sets the event input level to 1.1 V.

TBAS:EIN:POLarity(?)

This command sets the polarity of the event input.

Syntax `TBAS:EIN:POLarity <input_slope>`

`TBAS:EIN:POLarity?`



Arguments `<input_slope> ::= { NORMal | INVert }`

`NORMal`: Sets the polarity of the event input to positive.
`INVert`: Sets the polarity of the event input to negative.

`*RST` returns the setting to `NORMal`.

Returns `<input_slope>`

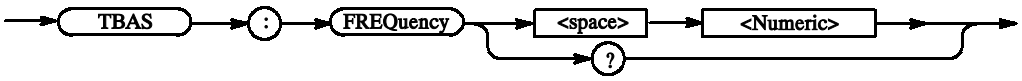
Examples `TBAS:EIN:POLarity INVert`
Sets the polarity of the event input to negative.

TBAS:FREQuency(?)

This command sets the frequency.

Syntax `TBAS:FREQuency <Numeric>`

`TBAS:FREQuency?`



Arguments Range:

DG mode (NRZ only)
DTG 5078: 50 kbps to 750 Mbps
DTG 5274: 50 kbps to 2.7 Gbps
DG mode (with RZ/R1)
DTG 5078: 50 kbps to 375 Mbps
DTG 5274: 50 kbps to 1.35 Gbps
PG mode

DTG 5078: 50 kHz to 375 MHz
DTG 5274: 50 kHz to 1.35 GHz

The resolutions of the frequency and cycle are eight-digit values.
By way of exception, they are four-digit values if the clock source is either the external clock input or external PLL input.

*RST returns the setting to 1e8(100MHz).

Returns <NR3>

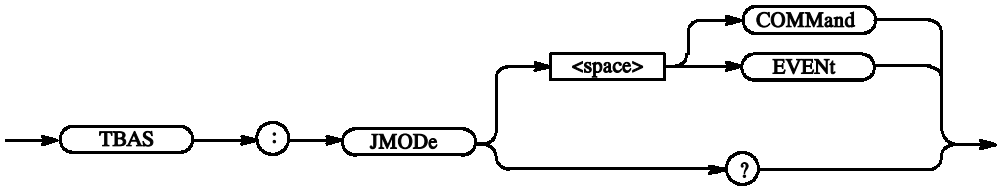
Examples TBAS:FREQuency 200MHZ
Sets the frequency to 200 MHz.

TBAS:JMODe(?)

This command sets the jump mode.

Syntax TBAS:JMODe { COMMand | EVENt }

TBAS:JMODe?



Arguments COMMand: Sets the jump mode to command.
EVENt: Sets the jump mode to event.

*RST returns the setting to EVENt.

Returns { COMMand | EVENt }

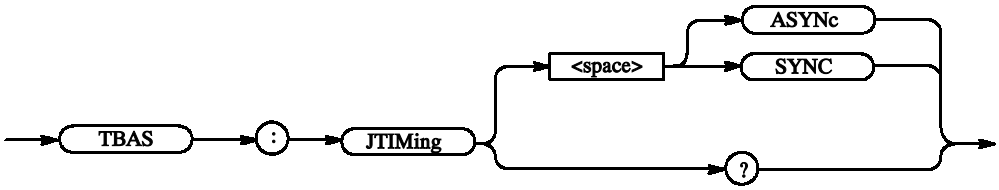
Examples TBAS:JMODe COMMand
Sets the jump mode to command.

TBAS:JTIMing(?)

This command sets the jump timing.

Syntax `TBAS:JTIMing {ASYNc | SYNC}`

`TBAS:JTIMing?`



Arguments ASYNc: Sets the jump timing to the asynchronous mode.
SYNC: Sets the jump timing to the synchronous mode.

*RST returns the setting to SYNC.

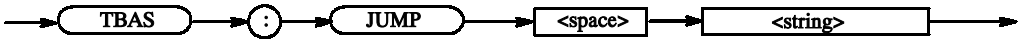
Returns { ASYNc | SYNC }

Examples TBAS:JTIMing ASYNc
Sets the jump timing to the asynchronous mode.

TBAS:JUMP (No Query Form)

This command causes a software jump.

Syntax `TBAS:JUMP <string>`



Arguments <string>: A label in the main sequence.

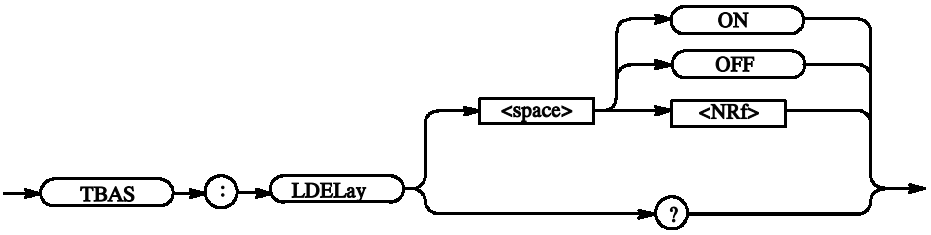
Examples TBAS:JUMP "test1"
Jumps to label “test1” in the sequence.

TBAS:LDELay (?)

Sets the long delay.
When long delay is on, you can't use a command jump and an event jump.

Syntax

```
TBAS:LDELay { ON | OFF | <NRf> }
TBAS:LDELay?
```



Arguments

OFF or <NRf> = 0 Turns off the long delay.

ON or <NRf> ≠ 0 Turns on the long delay.

*RST returns the setting to 0.

Returns

<NR1>

Examples

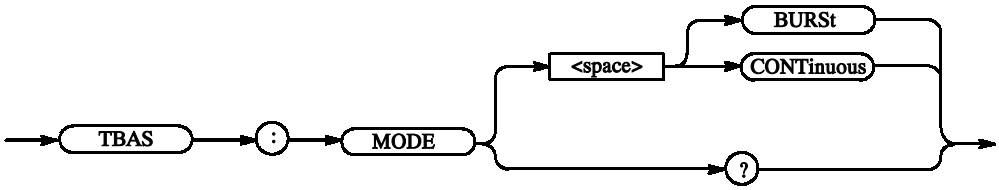
```
TBAS:LDELay ON
```
Turns on the long delay.

TBAS:MODE(?)

This command sets the PG run mode.

Syntax

```
TBAS:MODE {BURSt | CONTinuous}
TBAS:MODE?
```



Arguments BURSt: Sets the burst mode. (Waits for the trigger, and outputs the pulse the specified number of times.)

CONTinuous: Sets the mode to continuous. (Without waiting for the trigger, merely outputs the pulse consecutively.)

*RST returns the setting to CONTinuous.

Returns { BURSt | CONTinuous }

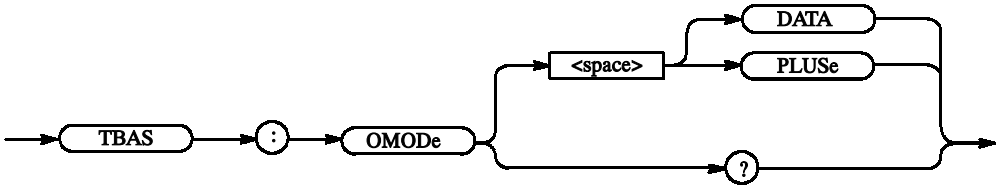
Examples TBAS:MODE BURSt
Sets the PG run mode to BURSt.

TBAS:OMODe(?)

This command sets the operating mode.

Syntax TBAS:OMODe {DATA | PULSe}

TBAS:OMODe?



Arguments DATA: Sets the mode to data generator (DG).

PULSe: Sets the mode to pulse generator (PG).

*RST returns the setting to DATA.

Returns { DATA | PULSe }

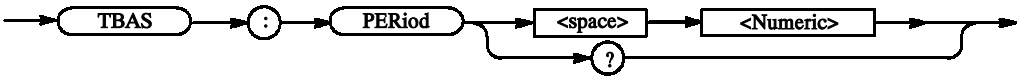
Examples TBAS:OMODe PULSe
Sets the mode to PG.

TBAS:PERiod(?)

This command sets the cycle.

Syntax TBAS:PERiod <Numeric>

TBAS:PERiod?



Arguments <Numeric> ::= PERiod

PERiod ::= 1/FREQuency

The range and other settings comply with the rules for FREQuency. You can query the minimum value and the maximum value by the use of MIN/MAX command.

*RST returns the setting to 1e-8 s.

Returns <NR3>

Examples TBAS:PERiod 2ns
Sets 2 ns for the cycle.
TBAS:PERiod? MIN
Query the minimum cycle at the current.

TBAS:PRATe? (Query Only)

This command queries the PLL multiplier rate.

Syntax TBAS:PRATe?

Returns <NR1>::= PLL Multiplier Rate

Examples TBAS:PRATe?
Queries the PLL multiplier rate.
For example, the following will be returned: 1000

TBAS:RSTate? (Query Only)

This command queries the sequencer status.

Syntax TBAS:RSTate?

Returns { RUN | STOP | WAIT | PUNLocked | ERMissing | EPMissing | ECMissing }

RUN - Now running
STOP - Now in stopped state
WAIT - Now waiting
PUNLocked - PLL remains unlocked.
ERMissing - The external reference was not found.
EPMissing - The external PLL input was not found.
ECMissing - The external clock was not found.

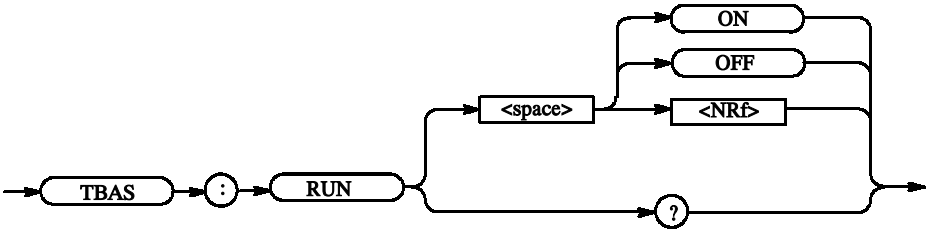
Examples TBAS:RSTate?
Queries the sequencer status.
The following is a response example, which returns when the sequencer is running:
RUN

TBAS:RUN (?)

Starts and stops the sequencer.

Syntax

```
TBAS:RUN { ON | OFF | <NRf> }
TBAS:RUN?
```



Arguments

OFF or <NRf> = 0 Stops the sequencer.

ON or <NRf> ≠ 0 Starts the sequencer.

Returns

<NR1>

Examples

```
TBAS:RUN ON
```

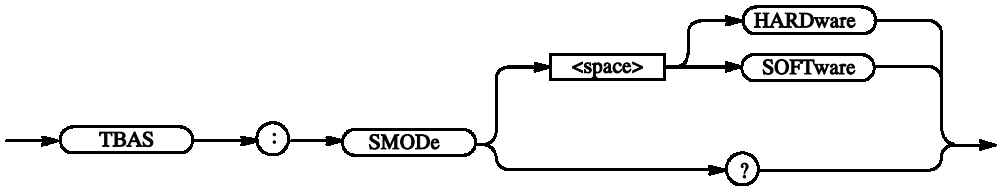
Starts the sequencer.

TBAS:SMODe(?)

Sets the sequencer mode.

Syntax

```
TBAS:SMODe {HARDware | SOFTware}
TBAS:SMODe?
```



Arguments

HARDware: Sets the sequencer mode to hardware.

SOFTware: Sets the sequencer mode to software.

*RST returns the setting to HARDware.

Returns {HARDware | SOFTware}

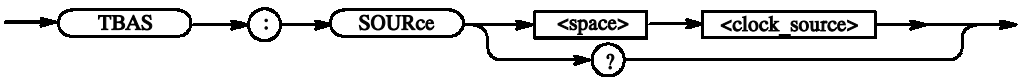
Examples TBAS:SMODe SOFTware
Sets the sequencer mode to software.

TBAS:SOURce(?)

Sets the clock source.
After changes are made to the clock source, the clock stops and does not restart automatically.

Syntax TBAS:SOURce <clock_source>

TBAS:SOURce?



Arguments <clock_source> ::= {INTernal | EXTReference | EXTPll | EXTernal }

INTernal - Internal 10 MHz reference
EXTReference - External 10MHz reference
EXTPll - External PLL input
EXTernal - External clock input

*RST returns the setting to INTernal.

Returns <clock_source>

Examples TBAS:SOURce EXTPll
Sets the clock source to external PLL input.

TBAS:TIN:IMPedance(?)

This command sets the trigger input impedance.

Syntax `TBAS:TIN:IMPedance <Numeric>`

`TBAS:TIN:IMPedance?`

Arguments Range: 50 or 1e3

*RST returns the setting to 1e3 ohm.

Returns <NR3>

Examples `TBAS:TIN:IMPedance 50`
Sets the trigger input impedance to 50 ohm.

TBAS:TIN:LEVel(?)

This command sets the trigger input level.

Syntax `TBAS:TIN:LEVel <Numeric>`

`TBAS:TIN:LEVel?`

Arguments Range: -5 to +5V

Step: 0.1 V

*RST returns the setting to 1.4 V.

Returns <NR3>

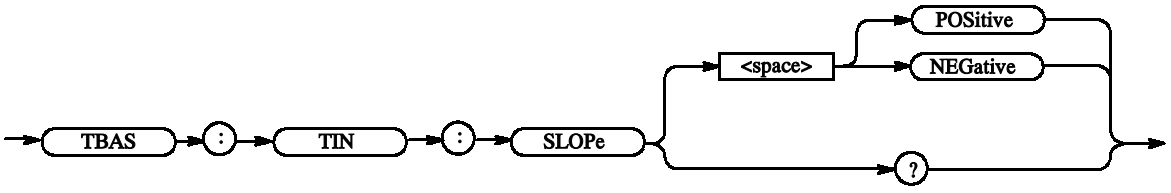
Examples `TBAS:TIN:LEVel 1.1`
Sets the trigger input level to 1.1 V.

TBAS:TIN:SLOPe(?)

This command sets the polarity of the trigger input.

Syntax `TBAS:TIN:SLOPe {POSitive | NEGative}`

`TBAS:TIN:SLOPe?`



Arguments `POSitive`: Positive

`NEGative`: Negative

*RST returns the setting to `POSitive`.

Returns `{POSitive | NEGative}`

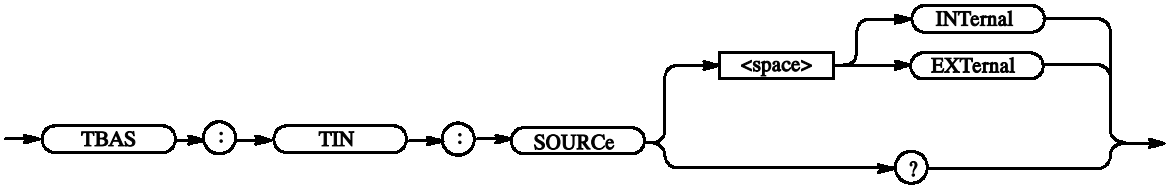
Examples `TBAS:TIN:SLOPe NEGative`
Sets the polarity of the trigger input to negative.

TBAS:TIN:SOURce(?)

This command sets the trigger input source.

Syntax `TBAS:TIN:SOURce {INTernal | EXTernal}`

`TBAS:TIN:SOURce?`



Arguments `INTernal`: Internal input

`EXTernal`: External input

*RST returns the setting to EXTernal.

Returns {INTernal | EXTernal}

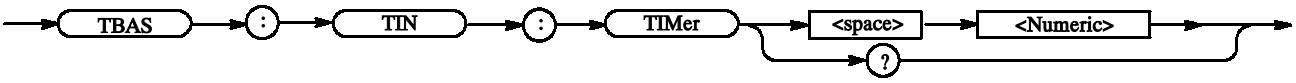
Examples TBAS:TIN:SOURce INTernal
Sets trigger input source to internal.

TBAS:TIN:TIMer(?)

This command sets the cycle of the internal trigger.

Syntax TBAS:TIN:TIMer <Numeric>

TBAS:TIN:TIMer?



Arguments Range: 1.0μs to 10.0 s

Step: 0.1μs (3 significant digits)

*RST returns the setting to 1e-3 s.

Returns <NR3>

Examples TBAS:TIN:TIMer 1.01
Sets the cycle of the internal trigger to 1.01 s.

TBAS:TIN:TRIGger (No Query Form)

This command generates a trigger.
Produces the same effect as *TRG.

Syntax TBAS:TIN:TRIGger

Arguments None

Examples `TBAS:TIN:TRIGger`
Generates a trigger signal.

TBAS:VRATe? (Query Only)

This command queries the vector rate.

Syntax `TBAS:VRATe?`

Returns <NR1> ::= Sends the response indicating the vector rate, i.e., the ratio between the hardware clock and user frequencies.

Examples `TBAS:VRATe?`
Queries the vector rate.
For example, the following will be returned: 8

*TRG (No Query Form)

This command generates a trigger. Produces the same effect as the Force Trigger key on the front panel.

Syntax `*TRG`

Arguments None

Examples `*TRG`
generates a trigger event.

*TST? (Query Only)

This command performs the selftest and returns the results. If an error is detected during selftest, execution is stopped immediately.

***NOTE**. This command takes several minutes to complete the self test, the data timing generator will not respond to any commands and queries during this time.*

Syntax

```
*TST?
```

Arguments None

Returns <NR1>

0 Terminated without error.
-330 Selftest failed.

Examples

```
*TST?
```

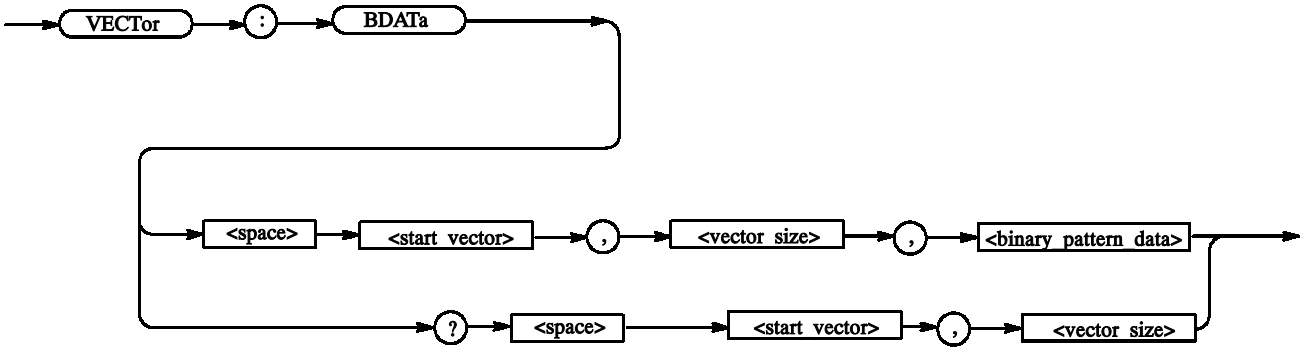
might return -330 indicating the selftest failed.

VECTor:BDATa(?)

This command transfers pattern data for several channels in binary format.

Syntax

```
VECTor:BDATa <start_vector>, <vector_size>, <binary_pattern_data>
VECTor:BDATa? <start_vector>, <vector_size>
```



Arguments <start_vector> ::= start address of data (First address is 0)

<vector_size> ::= data size

<binary_pattern_data> ::= binary byte block

NOTE. *Pattern data size is less than 1 MB (1024 x 1024).*

Returns <binary_pattern_data>

Examples

```
VECTor:BIOFormat "G1[2:10]","G2[1]"
VECTor:BDATa 1,2,#16abCDEF
```

The VECTor:BIOFormat command specifies the bus to which the data is to be transferred, and VECTor:BDATa transfers all the data for the channels together. The following steps are taken at this time:

- First, the necessary number of bits are obtained for each <signal> and the necessary number of bytes are obtained from the number of bits.
- The data is fetched by that number of bytes.
- The unnecessary MSBs are discarded. Then, the data is written into the specified logical channels in order, beginning at the MSB side.

This data contains the following:

#: Start character of the block

1: Indicates that the length in the length field is “1”.

6: Indicates that the length of the data is “6”.

abCDEF: Represents 01100001 01100010 01000011 01000100 01000101 01000110, and the data is therefore set as follows:

	Vector Add1			Vector Add2		
group	G1		G2	G1		G2
<binary pattern data>	a	b	C	D	E	F
Binary	0110000 1	01100010	0100001 1	01000100	01000101	01000110

Vector Add1

0110000 : not used

1 : G1[2]

01100010 : G1[3] - G1[10]

0100001 : not used

1 : G2[1]

Vector Add2

Same as “Vector Add1”

In the above example, G1[2:10] is specified; therefore, the MSBs and LSBs in the data for the six bits are placed into G1[2] and G1[10], respectively. This will be reversed if you specify G1[10:2].

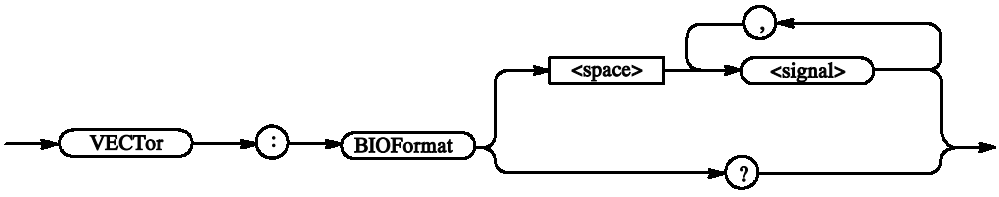
VECTor:BDATa? 2,10
Reads 10-bit data from the specified address, i.e., Address 2.

VECTor:BIOFormat(?)

This command sets the data items to be transferred with VECTor:BDATa.

Syntax

```
VECTor:BIOFormat <signal> [, <signal>...]
VECTor:BIOFormat?
```



Arguments

<signal> ::= <string> – Specifies the bus or logical channel for the data to be transferred with the BDATa command.

For example,

Addr[]
Addr[0:3]
Addr[0..3]
Addr[3..0]

NOTE. *If you omit the contents of the brackets, Addr[<msb>:<lsb>] will be assumed. (For example, Addr is 8 bit wide, Addr[] will be assumed to be Addr[7:0].)*

If you query a number of channels, the first channel's value will be returned. For example, the SIGN:HIGH? "DATA[2..4]" command returns a value of DATA[2].

Returns

<signal> [, <signal>...]

Examples

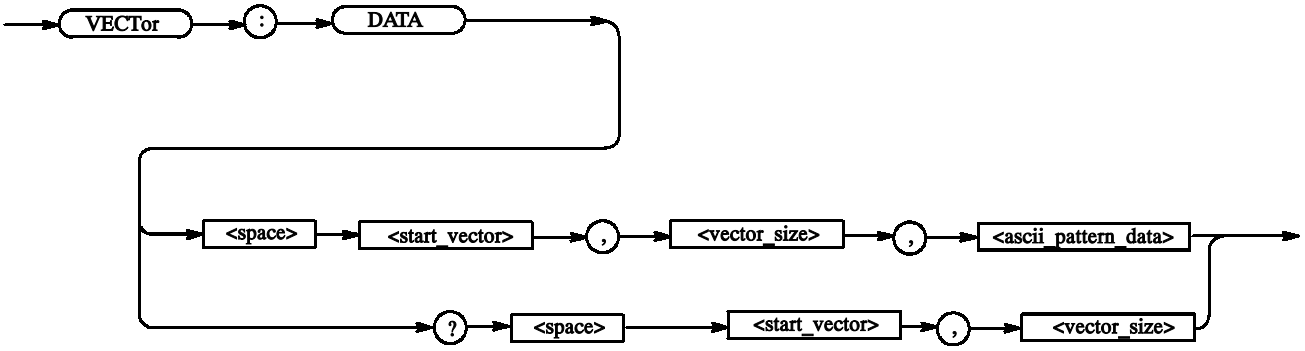
VECTor:BIOFormat "Addr[1:3]"
Specifies "Addr[1:3]" for the bus for the data to be transferred with the BDATa command.

VECTor:DATA(?)

This command transfers pattern data in ASCII format.

Syntax

```
VECTor:DATA <start_vector>, <vector_size>, <ascii_pattern_data>
VECTor:DATA? <start_vector>, <vector_size>
```



Arguments

<start_vector> ::= start address of data

<vector_size> ::= data size

<ascii_pattern_data> ::= data string

NOTE. *Pattern data size is less than 1 MB (1024 x 1024).*

Returns

<ascii_pattern_data>

Examples

```
VECTor:IOFormat "G1[2:7]",HEX,"G2[1]",BIN
VECTor:DATA 1,2,"AB0CD1"
```

When pattern data is received, the following processes are performed:

- The received <ascii_pattern_data> is fetched by each set of the number of characters.
- The characters are converted into the binary numbers, with the unnecessary MSBs discarded.
- The data is written into the specified logical channels in order, beginning at the MSB side.

	Vector Add1			Vector Add2		
group	G1		G2	G1		G2
<ascii pattern data>	A	B	0	C	D	1
Binary	1010	1011	0	1100	1101	1

Vector Add1

10 : not used
1 : G1[2]
0 : G1[3]
1 : G1[4]
0 : G1[5]
1 : G1[6]
1 : G1[7]
0 : G2[1]

Vector Add2

11 : not used
0 : G1[2]
0 : G1[3]
1 : G1[4]
1 : G1[5]
0 : G1[6]
1 : G1[7]
1 : G2[1]

Vector	Group G1									Group G2		
	8	7	6	5	4	3	2	1	0	2	1	0
0	x	x	x	x	x	x	x	x	x	x	x	x
1	x	1	1	0	1	0	1	x	x	x	0	x
2	x	1	0	1	1	0	0	x	x	x	1	x
3	x	x	x	x	x	x	x	x	x	x	x	x

In the above example, G1[2:7] is specified; therefore, the MSBs and LSBs in the data for the six bits are placed into G1[2] and G1[7], respectively. This will be reversed if you specify G1[7:2].

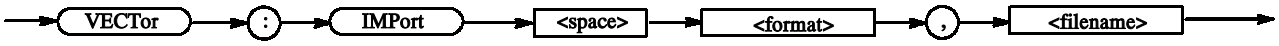
```
VECTor:IOFormat "DT",OCT
VECTor:DATA? 2,6
```

Reads 6-bit data from Address 2 of DT in OCT form.

VECTor:IMPort (No Query Form)

This command imports pattern data from a file to the block selected with BLOCk:SELect.

Syntax VECTor:IMPort <format>, <filename>



Arguments <format> ::= { TLA | VCA | VCB }

TLA: Tektronix TLA Data Exchange Format (*.txt)
VCA: HFS Vector Files (*.vca)
VCB: HFS Vector Files (*.vcb)

<filename> ::= <string> - File name (absolute path)

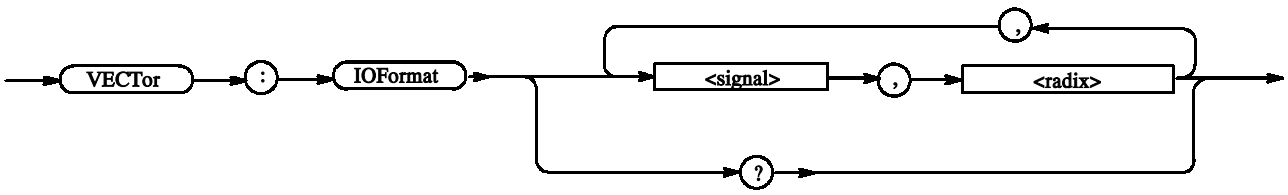
Examples VECTor:IMPort TLA,"C:\tmp\tla2.txt"
Imports a file named "C:\tmp\tla2.txt" that was created with TLA format in the DTG5000.

VECTor:IOFormat(?)

This command sets the data items to be transferred with VECTor:DATA and their format.

Syntax VECTor:IOFormat <signal>, <radix> [, <signal>, <radix> ...]

VECTor:IOFormat?



Arguments

<signal> ::= <string> - Specifies the bus or logical channel for the data to be transferred with the DATA command.

For example:

Addr[]
Addr[0:3]
Addr[0..3]
Addr[3..0]

NOTE. *If you omit the contents of the brackets, Addr[<msb>:<lsb>] will be assumed. (For example, Addr is 8 bit wide, Addr[] will be assumed to be Addr[7:0].)*

If you query a number of channels, the first channel's value will be returned. For example, the SIGN:HIGH? "DATA[2..4]" command returns a value of DATA[2].

<radix> ::= { BINary | OCTal | HEXadecimal }

BINary: Sets the base to binary.
OCTal: Sets the base to octal.
HEXadecimal: Sets the base to hexadecimal.

Returns

<signal>, <radix> [, <signal>, <radix> ...]

Examples

VECTor:IOFormat "Adr[0:3]",HEX
Makes the settings required to transfer data in HEX form from logical channel Adr0 to Adr3.

VECTor:IOFormat?
For example, the following will be returned:
"DT",OCT
DT is the bus or logical channel, and OCT is the base of data.

*WAI (No Query Form)

This command prevents the data timing generator from executing further commands or queries until all pending operations finish.

In DTG5000 series and in this application, all commands are designed to be executed in the order in which they are sent from the external controller.

Syntax `*WAI`

Arguments None

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

Status and Event Reporting

This section provides details about the status information and events the data timing generator reports.

Status Reporting Structure

The data timing generator status reporting functions conform to IEEE-488.2 and SCPI standards. Use the status reporting function to check for instrument errors and to identify the types of events that have occurred on the instrument.

Figure 3-1 is a diagram of the instrument's status reporting function.

- Standard/Event Status

The operations processed in this block are summarized in status bytes, which provide the error and event data.

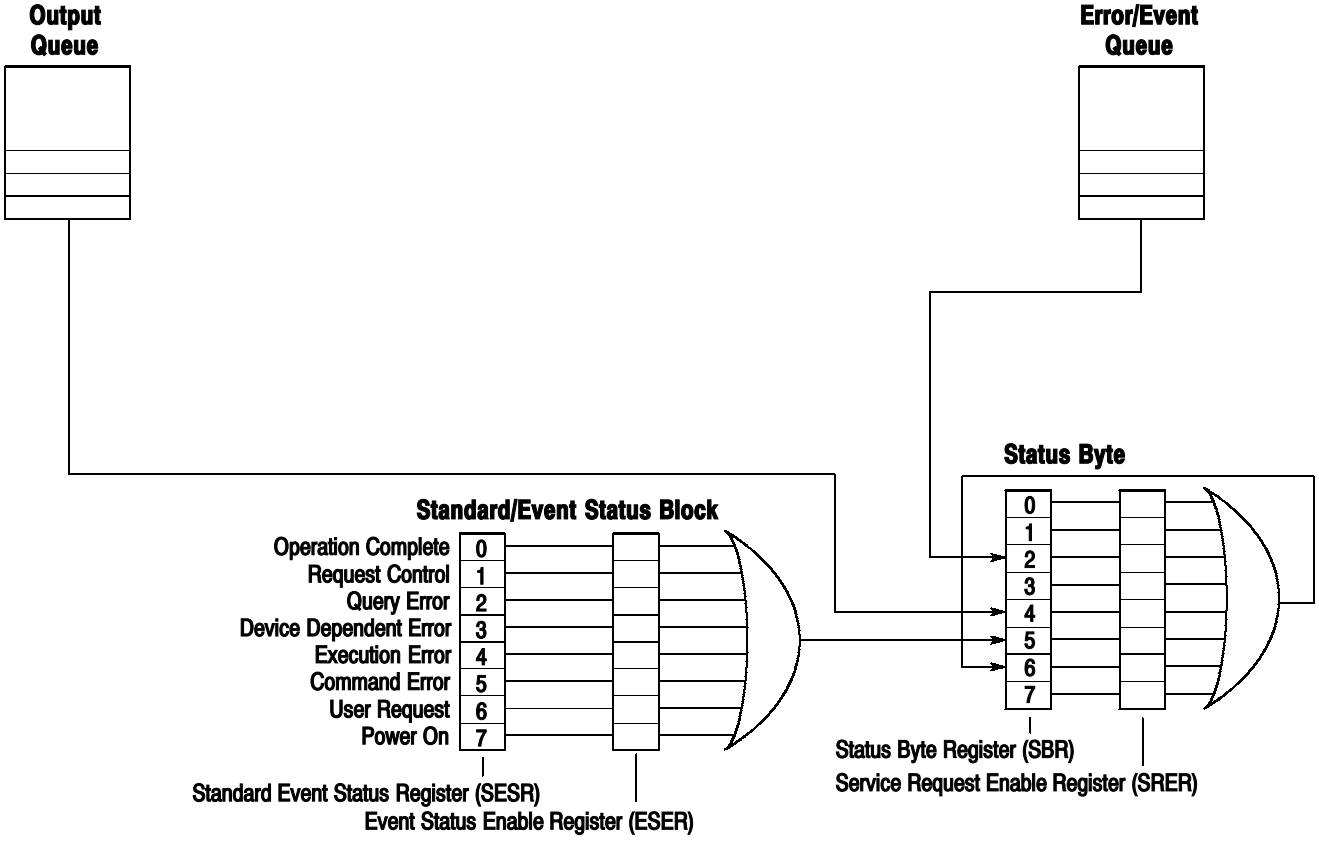


Figure 3-1: Error and Event handling process overview

Standard/Event Status Block

This block is used to report power on/off, command error, and command execution status.

The block has two registers: the Standard Event Status Register (SESR) and the Event Status Enable Register (ESER). Refer to the Standard/Event Status Block shown at the bottom of Figure 3-1 on page 3-2.

The SESR is an eight-bit status register. When an error or other type of event occurs on the instrument, the corresponding bit is set. You cannot write to this register. The ESER is an eight-bit enable register that masks the SESR. You can set this mask, and take AND with the SESR to determine whether or not the ESB bit in the Status Byte Register (SBR) should be set. Refer to *Event Status Enable Register (ESER)* on page 3-6, and *Standard Event Status Register (SESR)* on page 3-5, for the contents of these registers.

Registers

There are two main types of registers:

- **Status Registers**: store data relating to instrument status. These registers are set by the data timing generator.
- **Enable Registers**: determine whether to set events that occur in the instrument to the appropriate bits in the status registers and event queues. You can set this register.

Status Registers

There are two types of status registers:

- Status Byte Register (SBR)
- Standard Event Status Register (SESR)

Read the contents of these registers to determine errors and conditions.

Status Byte Register (SBR)

The SBR is made up of 8 bits. Bits 4, 5 and 6 are defined in accordance with IEEE Std 488.2-1992 (see Figure 3–2 and Table 3–1). These bits are used to monitor the output queue, SESR, and service requests, respectively. The contents of this register are returned when the *STB? query is used.

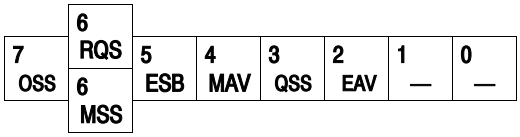


Figure 3–2: The Status Byte Register (SBR)

Table 3–1: SBR bit functions

Bit	Function
7	Operation Summary Status (OSS).
6	RQS (Request Service)/MSS (Master Summary Status). When the instrument is accessed using the GPIB serial poll command, this bit is called the Request Service (RQS) bit and indicates to the controller that a service request has occurred (in other words, that the GPIB bus SRQ line is LOW). The RQS bit is cleared when serial poll ends. When the instrument is accessed using the *STB? query, this bit is called the Master Summary Status (MSS) bit and indicates that the instrument has issued a service request for one or more reasons. The MSS bit is never cleared to 0 by the *STB? query.
5	Event Status Bit (ESB). This bit indicates whether or not a new event has occurred after the previous Standard Event Status Register (SESR) has been cleared or after an event readout has been performed.
4	Message Available Bit (MAV). This bit indicates that a message has been placed in the output queue and can be retrieved.
3	Questionable Summary Status (QSS).
2	Event Queue Available (EAV).
1–0	Not used

Standard Event Status Register (SESR)

The SESR is made up of 8 bits. Each bit records the occurrence of a different type of event, shown in Figure 3–3 and Table 3–2. The contents of this register are returned when the *ESR? query is used.

7	6	5	4	3	2	1	0
PON	—	CME	EXE	DDE	QYE	—	OPC

Figure 3–3: The Standard Event Status Register (SESR)

Table 3–2: SESR bit functions

Bit	Function
7	Power On (PON). Indicates that the power to the instrument is on.
6	Not used.
5	Command Error (CME). Indicates that a command error has occurred while parsing by the command parser was in progress.
4	Execution Error (EXE). Indicates that an error occurred during the execution of a command. Execution errors occur for one of the following reasons: ■ A value designated in the argument is outside the allowable range of the instrument, or is in conflict with the capabilities of the instrument ■ The command could not be executed properly because the conditions for execution differed from those essentially required
3	Device-Specific Error (DDE). An instrument error has been detected.
2	Query Error (QYE). Indicates that a query error has been detected by the output queue controller. Query errors occur for one of the following reasons: ■ An attempt was made to retrieve messages from the output queue, despite the fact that the output queue is empty or in pending status. ■ The output queue messages have been cleared despite the fact that they have not been retrieved.
1	Not used.
0	Operation Complete (OPC). This bit is set with the results of the execution of the *OPC command. It indicates that all pending operations have been completed.

Enable Registers

There are two types of enable registers:

- Event Status Enable Register (ESER)
- Service Request Enable Register (SRER)

Each bit in the enable registers corresponds to a bit in the controlling status register. By setting and resetting the bits in the enable register, you can determine whether or not events that occur will be registered to the status register and queue.

Event Status Enable Register (ESER)

The ESER is made up of bits defined exactly the same as bits 0 through 7 in the SESR register (see Figure 3-4). You can use this register to designate whether or not the SBR ESB bit should be set when an event has occurred, and to determine if the corresponding SESR bit is set.

To set the SBR ESB bit (when the SESR bit has been set), set the ESER bit corresponding to that event. To prevent the ESB bit from being set, reset the ESER bit corresponding to that event.

Use the *ESE command to set the bits of the ESER. Use the *ESE? query to read the contents of the ESER.

7	6	5	4	3	2	1	0
PON	—	CME	EXE	DDE	QYE	—	OPC

Figure 3-4: The Event Status Enable Register (ESER)

Service Request Enable Register (SRER)

The SRER is made up of bits defined exactly the same as bits 0 through 7 in the SBR (see Figure 3-5). You can use this register to define which events will generate service requests.

The SRER bit 6 cannot be set. Also, the RQS is not maskable.

The generation of a service request with the GPIB interface involves changing the SRQ line to LOW, and making a service request to the controller. The result is that a status byte for which an RQS has been set is returned in response to serial polling by the controller.

Use the *SRE command to set the bits of the SRER. Use the *SRE? query to read the contents of the SRER. Bit 6 must be set to 0.

7	6	5	4	3	2	1	0
OSS	—	ESB	MAV	QSS	EAV	—	—

Figure 3-5: The Service Request Enable Register (SRER)

Queues

There are two types of queues in the status reporting system: output queues and error/event queues.

Output Queue

The output queue is a FIFO (first-in, first-out) queue that holds response messages to queries awaiting retrieval. When there are messages in the queue, the SBR MAV bit is set.

The output queue is emptied each time a command or query is received, so the controller must read the output queue before the next command or query is issued. If this is not done, an error occurs and the output queue is emptied; however, the operation proceeds even if an error occurs.

Error/Event Queue

The event queue is a FIFO queue, which stores events as they occur in the instrument. Maximum event count in the que is 100.

- `SYSTem:ERRor[:NEXT]?`

First, issue the `*ESR?` query to read the contents of the SESR. The contents of the SESR are cleared after they are read. If an SESR bit is set, events are stacked in the Error/Event Queue. Retrieve the event code with the following command sequence:

```
*ESR?
SYSTem:ERRor[:NEXT]?
```

If you omit the `*ESR?` query, the SESR bit will remain set, even if the event disappears from the Error/Event Queue.

Status and Event Processing Sequence

Standard/Event Status Block

As illustrated in Figure 3-6, when an event occurs, a signal is sent to the SESR and the event is recorded in the Event Queue (1). If the corresponding bit in the ESER is also enabled (2), the ESB bit in the SBR is set to one (3).

When output is sent to the Output Queue, the MAV bit in the SBR is set to one (4).

When a bit in the SBR is set to one and the corresponding bit in the SRER is enabled (5), the MSS bit in the SBR is set to one and a service request is generated (6).

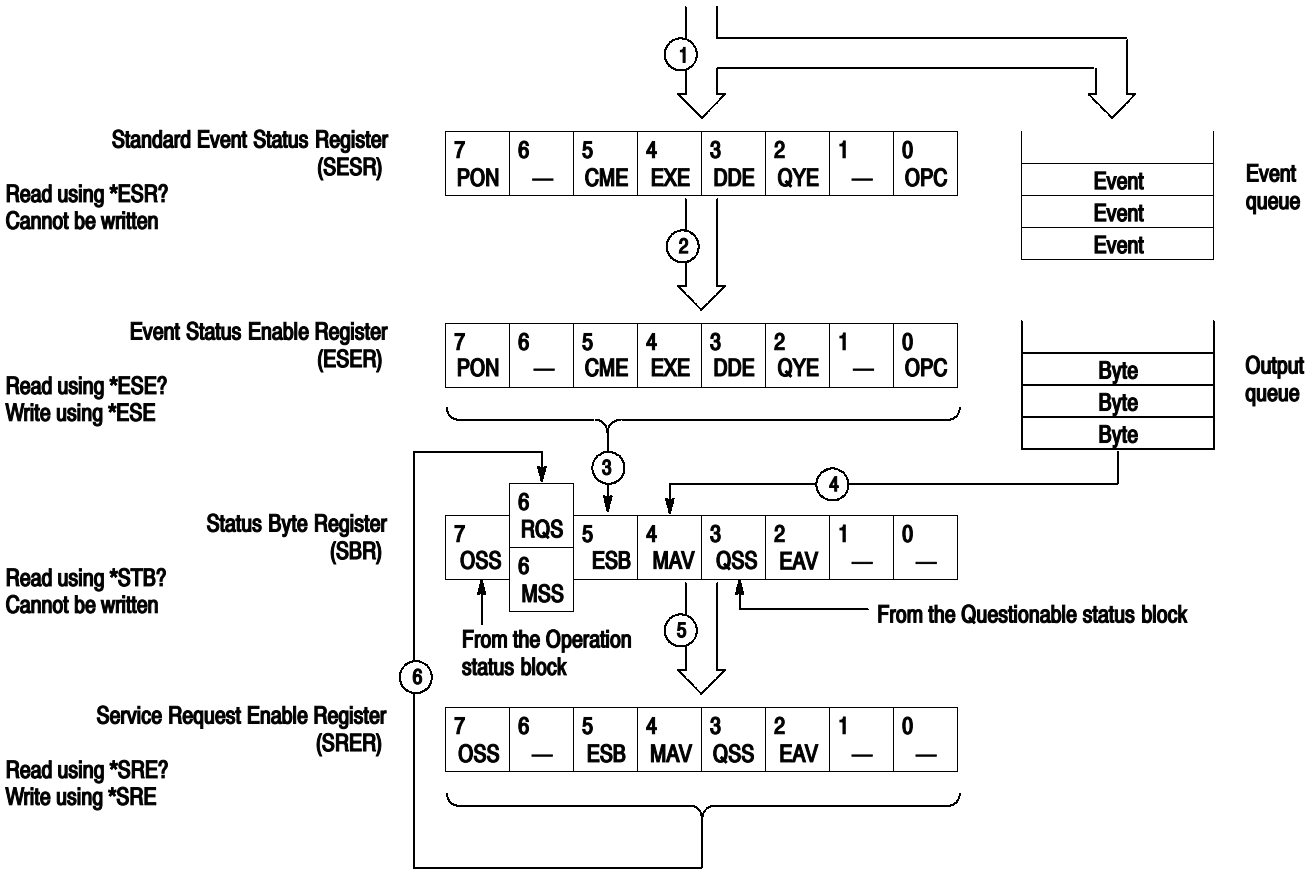


Figure 3-6: Status and Event processing sequence — Standard/Event status block

Synchronizing Execution

All commands used in the data timing generator are designed to be executed in the order in which they are sent from the external controller. But it cannot be checked by the following methods whether execution of a command has been completely performed to hardware.

```
PGENA:CH1:HIGH 2.0
PGENA:CH1:HIGH?
```

Even if the reply of HIGH? comes by the above-mentioned example, a setup to the hardware of the previous command may not be ended at the time. The setting end to hardware can be checked by the following methods.

```
PGENA1:CH1:HIGH 2.0;*OPC
```

--> An end can be checked when the event of Operation Complete occurs.

```
PGENA1:CH1:HIGH 2.0;*OPC?
```

--> An end can be checked when 1 should be returned.

```
PGENA:CH1:HIGH 2.0;*WAI
PGENA:CH1:HIGH?
```

--> An end can be checked when the reply of HIGH? should be returned.

Messages

Tables 3-4 through 3-11 show the codes and messages used in the status and event reporting system.

Event codes and messages can be obtained by using the queries SYSTem:ERRor[:NEXT]?. Responses are returned in the following format:

<event code>,"<event message>"

Messages and Codes

Error and event codes with negative values are SCPI standard codes.

Table 3–3 lists event code definitions. When an error occurs, you can find its error class by checking for its the code range in Tables 3–4 through 3–11. Events in these tables are organized by event class.

Table 3–3: Definition of event codes

Event class	Code range	Description
No error	0	No event or status
Command errors	–100 to –199	Command syntax errors
Execution errors	–200 to –299	Command execution errors
Device-specific errors	–300 to –399	Internal device errors
Query errors	–400 to –499	System event and query errors
Power-on events	–500 to –599	Power-on events
User request events	–600 to –699	User request events
Request control events	–700 to –799	Request control events
Operation complete events	–800 to –899	Operation complete events

Command Errors

Command errors are returned when there is a syntax error in the command.

Table 3-4: Command errors

Error code	Error message
-100	Command error
-101	Invalid character
-102	Syntax error
-103	Invalid separator
-104	Data type error
-105	GET not allowed
-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
-115	Unexpected number of parameters
-120	Numeric data error
-121	Invalid character in number
-123	Exponent too large
-124	Too many digits
-128	Numeric data not allowed

Table 3-4: Command errors (cont.)

Error code	Error message
-130	Suffix error
-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
-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

Execution Errors

These error codes are returned when an error is detected during command execution.

Table 3-5: Execution errors

Error code	Error message
-200	Execution error
-201	Invalid while in local
-202	Settings lost due to RTL
-203	Command protected
-210	Trigger error
-211	Trigger ignored
-212	Arm 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	Out of memory
-226	Lists not same length
-230	Data corrupt or stale
-231	Data questionable
-232	Invalid format
-233	Invalid version
-240	Hardware error
-241	Hardware missing
-250	Mass storage error
-251	Missing mass storage
-252	Missing media

Table 3-5: Execution errors (cont.)

Error code	Error message
-253	Corrupt media
-254	Media full
-255	Directory full
-256	File name not found
-257	File name error
-258	Media protected
-260	Expression error
-261	Math error in expression
-270	Macro error
-271	Macro syntax error
-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 runtime error
-290	Memory use error
-291	Out of memory
-292	Referenced name does not exist
-293	Referenced name already exists
-294	Incompatible type

Device Specific Errors

These error codes are returned when an internal instrument error is detected. This type of error can indicate a hardware problem.

Table 3-6: Device specific errors

Error code	Error message
-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
-320	Storage fault
-321	Out of memory
-330	Self-test failed
-340	Calibration failed
-350	Queue overflow
-360	Communication error
-361	Parity error in program message
-362	Framing error in program message
-363	Input buffer overrun
-365	Time out error

Query Errors

These error codes are returned in response to an unanswered query.

Table 3-7: Query errors

Error code	Error message
-400	query error
-410	query INTERRUPTED
-420	query UNTERMINATED
-430	query DEADLOCKED
-440	query UNTERMINATED after indefinite response

Power-On Events

These events occur when the instrument detects an off to on transition in its power supply.

Table 3-8: Power-on events

Event code	Event message
-500	Power on

User Request Events

These events are unused in DTG5000 series.

Table 3-9: User request events

Event code	Event message
-600	User request

Request Control Events

This event is unused in DTG5000 series.

Table 3-10: Request control events

Event code	Event message
-700	Request control

Operation Complete Events

This event occurs when the instrument's synchronization protocol, having been enabled by an *OPC command, completes all selected pending operations.

Table 3-11: Operation complete events

Event code	Event message
-800	Operation complete

Programming Examples

This section includes a sample program to use the GPIB interfaces. This program is written in Microsoft Visual BASIC V6.0.

GPIB programs run on a PC-compatible system. To use the GPIB interface, your PC-compatible system must be equipped with a National Instruments GPIB board and associated drivers. GPIB programs are also compatible with National Instruments LabVIEW.

Sample program

This program outputs 4 bits binary counter repeatedly.

```
Option Explicit

Private Sub Command1_Click()
    'Following is the example to
    Dim dev%, i&, count&
    Dim s$, ss$, blockSize&

    blockSize = 1024

    'select GPIB Board 0
    'Primary Address 1
    'Secondary Address None
    'Timeout 3s
    'Assert EOI at the end of ibwrt
    'Do not handle EOS character automatically
    ibdev 0, 1, 0, T10s, 1, 0, dev

    ibwrt dev, "*CLS"
    ibwrt dev, "*RST"
    ibwrt dev, "GROUP:DELETE:ALL"
    ibwrt dev, "GROUP:NEW ""GRP1"",4"
    ibwrt dev, "BLOCK:DELETE:ALL"
    ibwrt dev, "BLOCK:NEW ""BLK1"", " & blockSize
    ibwrt dev, "BLOCK:SELECT ""BLK1"""
    ibwrt dev, "VECTOR:IOFORMAT ""GRP1"", HEX"

    'Define 4bit counter pattern
    count = blockSize / 16
    s = "0123456789ABCDEF"
```

```
    For i = 0 To count - 1
        ss = ss & s
    Next
    'Send block data
    ibwrt dev, "VECTOR:DATA 0, " & blockSize & ", """ & ss & """"

    ibwrt dev, "SEQUENCE:LENGTH 1"
'Define Sequence
    ibwrt dev, "SEQUENCE:DATA 0, """", 0, ""BLK1"",0,"""","""""
'Infinite Loop of BLK1
    ibwrt dev, "SIGNAL:ASSIGN ""GRP1[3]"", ""A1"""
'Channel Assign
    ibwrt dev, "SIGNAL:ASSIGN ""GRP1[2]"", ""A2"""
    ibwrt dev, "SIGNAL:ASSIGN ""GRP1[1]"", ""B1"""
    ibwrt dev, "SIGNAL:ASSIGN ""GRP1[0]"", ""B2"""
    ibwrt dev, "TBAS:FREQ 100e6"
'Set Frequency
    ibwrt dev, "SIGNAL:HIGH ""GRP1[]"", 0.5"
'Set High Level
    ibwrt dev, "SIGNAL:LOW ""GRP1[]"", -0.0"
'Set Low Level
    For i = 0 To 3
        ibwrt dev, "SIGNAL:OUTPUT ""GRP1[" & i & "]"", 1"
'Output On
    Next
    ibwrt dev, "TBAS:RUN 1"
'Start Sequencer

End Sub
```

Appendix A: Character Charts

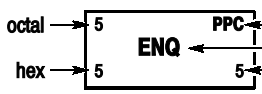
Table A-1: The DTG character set

	0	1	2	3	4	5	6	7
0	NUL 0	16	space 32	0 48	@ 64	P 80	‘ 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	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	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 SP LA0 20 32	60 0 LA16 30 48	100 @ TA0 40 64	120 P TA16 50 80	140 ` SA0 60 96	160 p SA16 70 112
0 0 0 1	1 SOH GTL 1 1	21 DC1 LL0 11 17	41 ! LA1 21 33	61 1 LA17 31 49	101 A TA1 41 65	121 Q TA17 51 81	141 a SA1 61 97	161 q SA17 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 " LA2 22 34	62 2 LA18 32 50	102 B TA2 42 66	122 R TA18 52 82	142 b SA2 62 98	162 r SA18 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 # LA3 23 35	63 3 LA19 33 51	103 C TA3 43 67	123 S TA19 53 83	143 c SA3 63 99	163 s SA19 73 115
0 1 0 0	4 EOT SDC 4 4	24 DC4 DCL 14 20	44 $ LA4 24 36	64 4 LA20 34 52	104 D TA4 44 68	124 T TA20 54 84	144 d SA4 64 100	164 t SA20 74 116
0 1 0 1	5 ENQ PPC 5 5	25 NAK PPU 15 21	45 % LA5 25 37	65 5 LA21 35 53	105 E TA5 45 69	125 U TA21 55 85	145 e SA5 65 101	165 u SA21 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 & LA6 26 38	66 6 LA22 36 54	106 F TA6 46 70	126 V TA22 56 86	146 f SA6 66 102	166 v SA22 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' LA7 27 39	67 7 LA23 37 55	107 G TA7 47 71	127 W TA23 57 87	147 g SA7 67 103	167 w SA23 77 119
1 0 0 0	10 BS GET 8 8	30 CAN SPE 18 24	50 (LA8 28 40	70 8 LA24 38 56	110 H TA8 48 72	130 X TA24 58 88	150 h SA8 68 104	170 x SA24 78 120
1 0 0 1	11 HT TCT 9 9	31 EM SPD 19 25	51) LA9 29 41	71 9 LA25 39 57	111 I TA9 49 73	131 Y TA25 59 89	151 i SA9 69 105	171 y SA25 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 * LA10 2A 42	72 : LA26 3A 58	112 J TA10 4A 74	132 Z TA26 5A 90	152 j SA10 6A 106	172 z SA26 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 + LA11 2B 43	73 ; LA27 3B 59	113 K TA11 4B 75	133 [TA27 5B 91	153 k SA11 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 L TA12 4C 76	134 \ TA28 5C 92	154 l SA12 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 M TA13 4D 77	135] TA29 5D 93	155 m SA13 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 N TA14 4E 78	136 ^ TA30 5E 94	156 n SA14 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 O TA15 4F 79	137 _ UNT 5F 95	157 o SA15 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY



Tektronix
REF: ANSI STD X3.4-1977
IEEE STD 488.2-1992
ISO STD 646-2973

Appendix B: GPIB Interface Specification

This appendix lists and describes the GPIB functions and messages the data timing generator implements.

Interface Functions

Table B–1 lists the GPIB interface functions this instrument implements. Each function is briefly described on the next page.

Table B–1: GPIB interface function implementation

Interface function	Implemented subset	Capability
Acceptor Handshake (AH)	AH1	Complete
Source Handshake (SH)	SH1	Complete
Talker (T)	T6	Basic Talker, Serial Poll Unaddress if my-listen-address (MLA) No Talk Only mode
Listener (L)	L4	Basic Listener Unaddress if my talk address (MTA) No Listen Only mode
Service Request (SR)	SR1	Complete
Remote/Local (RL)	RL1	Complete
Parallel Poll (PP)	PP0	None
Device Clear (DC)	DC1	Complete
Device Trigger (DT)	DT1	Complete
Controller (C)	C0	None
Electrical Interface	E2	Three-state driver

- Acceptor Handshake (AH). Enables a listening device to coordinate data reception. The AH function delays data transfer initiation or termination until the listening device is ready to receive the next data byte.
- Source Handshake (SH). Enables a talking device to support the coordination of data transfer. The SH function controls the initiation and termination of data byte transfers.
- Talker (T). Enables a device to send device-dependent data over the interface. This capability is available only when the device is addressed to talk, and uses a one-byte address.
- Listener (L). Enables a device to receive device-dependent data over the interface. This capability is available only when the device is addressed to listen, and uses a one-byte address.
- Service Request (SR). Enables a device to request service from the controller.
- Remote/Local (RL). Enables a device to select between one of two sources for data timing generator control. It determines whether input information is controlled from the front panel (local control) or by GPIB commands (remote control).
- Device Clear (DC). Enables a device to be cleared or initialized, either individually, or as part of a group of devices.
- Controller (C). Enables a device that has this capability to send its address, universal commands, and addressed commands to other devices over the interface.
- Electrical Interface (E). Identifies the electrical interface driver type. The notation E1 means the electrical interface uses open collector drivers, E2 means the electrical interface uses three-state drivers.

Interface Messages

Table B–2 shows the standard interface messages that the data timing generator supports. Brief function descriptions are provided on the next page.

Table B–2: DTG standard interface message

Message	GPIB
DCL	Yes
GET	Yes
GTL	Yes
LLO	Yes
PPC	No
PPD	No
PPE	No
PPU	No
SDC	Yes
SPD	Yes
SPE	Yes
TCT	No
UNL	Yes
UNT	Yes
Listen Addresses	Yes
Talk Addresses	Yes

* **For detailed information see page B–4.**

- Device Clear (DCL). Will clear (initialize) all devices on the bus that have a device clear function, whether or not the controller has addressed them.
- Group Execute Trigger (GET). Triggers all applicable devices and causes them to initiate their programmed actions.
- Go To Local (GTL). Causes the listen-addressed device to switch from remote to local (front-panel) control.
- Local Lockout (LLO). Disables the return to local function.
- 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.
- Select Device Clear (SDC). Clears or initializes all listen-addressed devices.
- Serial Poll Disable (SPD). Changes all devices on the bus from the serial poll state to the normal operating state.
- Serial Poll Enable (SPE). Puts all bus devices 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 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.
- Take Control (TCT). Allows the controller in charge to pass control of the bus to another controller on the bus.

Appendix C: Factory Initialization Settings

The following tables lists the commands affected by factory initialization. Table C–1 on page C–1 lists commands for the DTG5000 series.

The *RST command has no effect on the Status commands.

Table C–1: Factory initialization settings

Header	Default settings
BLOCk:SELect	“”
DIAGnostic:SELect	ALL
JGENeration:AMPLitude	0
JGENeration:AMPLitude:UNIT	SPP
JGENeration:EDGE	BOTH
JGENeration:FREQuency	1e6
JGENeration:GSOurce	“”
JGENeration:MODE	ALL
JGENeration:PROFile	SINusoid
JGENeration[:STATe]	0
OUTPut:CLOCk:AMPLitude	1.0
OUTPut:CLOCk:OFFSet	0.48
OUTPut:CLOCk[:STATe]	0
OUTPut:CLOCk:TIMPedance	50
OUTPut:CLOCk:TVOLtage	0
OUTPut:DC:HLIMit	1.0
OUTPut:DC:LEVel	1.0
OUTPut:DC:LIMit	0
OUTPut:DC:LLIMit	0
OUTPut:DC[:STATe]	0
PGEN<x>[m]:CH<n>:AMODe	NORMal
PGEN<x>[m]:CH<n>:AMPLitude	1
PGEN<x>[m]:CH<n>:CPOint	50
PGEN<x>[m]:CH<n>:DCYCle	50
PGEN<x>[m]:CH<n>:DTOFfset	0
PGEN<x>[m]:CH<n>:DTOFfset:STATe	0

Table C-1: Factory initialization settings (cont.)

Header	Default settings
PGEN<x>[m]:CH<n>:HIGH	1.0
PGEN<x>[m]:CH<n>:HLIMit	1.0
PGEN<x>[m]:CH<n>:LDELay	0
PGEN<x>[m]:CH<n>:LHOLd	LDELay
PGEN<x>[m]:CH<n>:LIMit	0
PGEN<x>[m]:CH<n>:LLIMit	0
PGEN<x>[m]:CH<n>:LOW	0
PGEN<x>[m]:CH<n>:OFFSet	0.5
PGEN<x>[m]:CH<n>:OUTPut	0
PGEN<x>[m]:CH<n>:PHASe	0
PGEN<x>[m]:CH<n>:POLarity	NORMal
PGEN<x>[m]:CH<n>:PRATe	NORMal
PGEN<x>[m]:CH<n>:SLEW	2.25
PGEN<x>[m]:CH<n>:TDELay	5e-9
PGEN<x>[m]:CH<n>:THOLd	DCYCle
PGEN<x>[m]:CH<n>:TIMPedance	50
PGEN<x>[m]:CH<n>:TVOLtage	0
PGEN<x>[m]:CH<n>:TYPE	NRZ
PGEN<x>[m]:CH<n>:WIDTh	5e-9
SEQuence:LENGth	1
SIGNal:ASSign	Resets the assignment
SUBSequence:LENGth	-1
SUBSequence:SELect	“”
SYSTem:KLOCk	0
TBAS:COUNt	1
TBAS:CRANge	12
TBAS:DOFFset	0
TBAS:EIN:IMPedance	1e3
TBAS:EIN:LEVel	1.4
TBAS:EIN:POLarity	NORMal
TBAS:FREQuency	1e8
TBAS:JMODe	EVENt
TBAS:JTIMing	SYNC

Table C-1: Factory initialization settings (cont.)

Header	Default settings
TBAS:LDELay	0
TBAS:MODE	CONTinuous
TBAS:OMODe	DATA
TBAS:PERiod	1e-8
TBAS:SMODe	HARDware
TBAS:SOURce	INTernal
TBAS:TIN:IMPedance	1e3
TBAS:TIN:LEVel	1.4
TBAS:TIN:SLOPe	POSitive
TBAS:TIN:SOURce	EXTernal
TBAS:TIN:TIMer	1e-3

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.

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.

SCPI
Acronym for Standard Commands for Programmable Instruments.

Index

J

L

M

N

O

P

Q

R

S

T

U

V

W