Programmer Manual

SONY Tektronix

AWG2000 Series
Arbitrary Waveform Generators

070-8657-06

Printed in Japan.

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

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

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

Examples

Appendices

Glossary & Index

List of Figures

List of Tables

Preface

This is the Programmer Manual for the AWG2000 Series Arbitrary Waveform Generators. This manual provides information on operating these instruments using General Purpose Interface Bus (GPIB) interface and RS-232-C interface.

Related Manuals

Other documentation for the waveform generators includes:

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

Getting Started

Getting Started

Overview

The Arbitrary Waveform Generator has two interfaces for remote operation — the GPIB interface and the RS-232-C interface. All menu controlled and front-panel controlled functions, except the ON/STBY function, the edit function, and the GPIB and RS-232-C parameter setup functions, can be controlled through the GPIB or the RS-232-C interface using the programming command set (see Section 3).

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

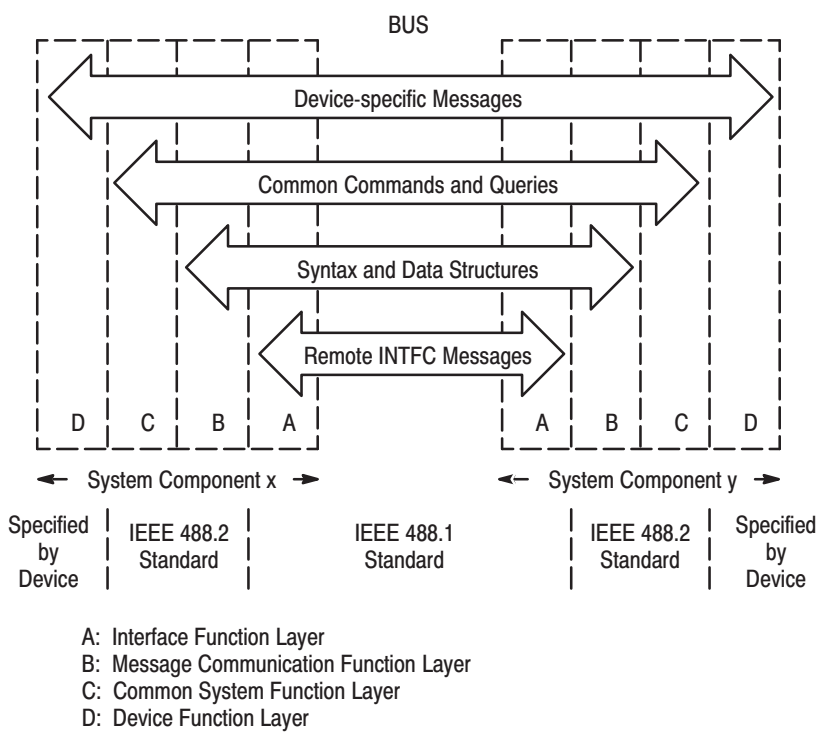


Figure 1-1: Functional Layers in GPIB System

The RS-232-C interface, which was established by the Electronic Industries Association (EIA), provides a common basis of communication between devices

that exchange data. This interface has long been used on terminals, modems, printers, and other devices. The RS-232-C interface that the waveform generator provides also uses most of the same Codes, Formats, Protocols, and Common Commands as are used with the GPIB interface (ANSI/IEEE Std 488.2-1987).

Programmer Manual Contents

- *Getting Started* describes how to connect and set up for remote operation.
- *Syntax and Commands* define the command syntax and processing conventions and describes each command in the waveform generator command set.
- *Status and Events* explain the status information and event messages reported by the waveform generator.
- *Examples* describe how to compile, link, and use the example programs provided on the floppy disk included with this manual. These programs also serve as examples of how you can program the waveform generator to do certain tasks (waveform transmission, for example).
- *Appendices* collect various topics of use to the programmer.
- *Glossary and Index* contains a glossary of common terms and an index to this manual.

Choosing an Interface

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

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

Operating Attribute	GPIB	RS-232-C
Cable	ANSI/IEEE Std 488	9-wire (DCE)
Data flow control	Hardware, 3-wire handshake	Flagging: soft (XON/XOFF), hard (DTR/CTS)
Data format	8-bit parallel	8-bit serial
Interface control	Operator low-level control message	None
Interface messages	Most ANSI/IEEE Std 488	Device clear via ASCII break signal

Table 1-1: GPIB and RS-232-C Comparison (Cont.)

Operating Attribute	GPIB	RS-232-C
Interrupts reported	Service requests status and event code	Status and event code (no service requests)
Message termination (Receive)	Hardware EOI, software LF, or both	Software CR, LF, or CR and LF
Message termination (Transmit)	Hardware EOI, and software LF	Software LF
Timing	Asynchronous	Asynchronous
Transmission path length	≤2 meters between devices; ≤20 meters total cabling for GPIB system	≤15 meters
Speed	200 Kbytes/sec	19,200 bits/sec
System environment	Multiple devices (≤15)	Single terminal (point to point connection)

Installing for GPIB Communication

With the power off, connect a GPIB cable from the GPIB controller to the ANSI/IEEE Std 488 port (GPIB) connector on the rear panel of the waveform generator (see Figure 1-2).

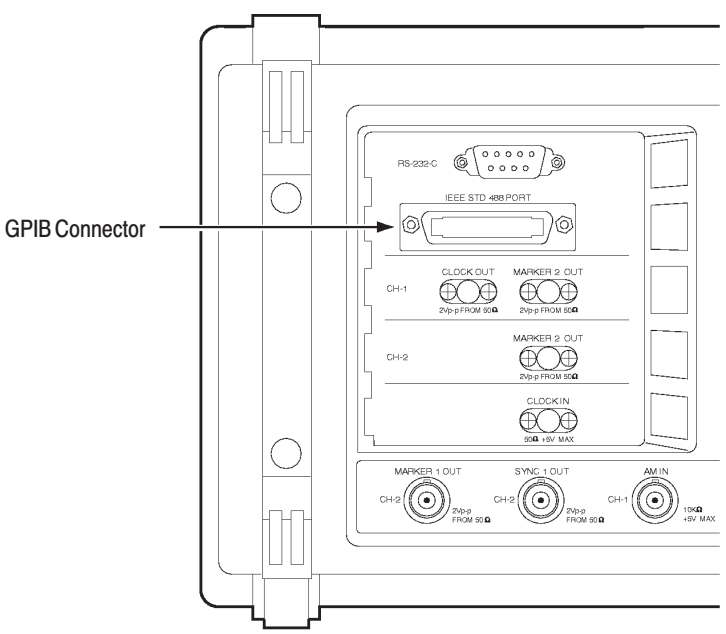


Figure 1-2: GPIB Connector

For example, when using an MS-DOS compatible controller, connect the GPIB cable between the National Instrument PC2A GPIB board and the waveform generator GPIB connector.

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

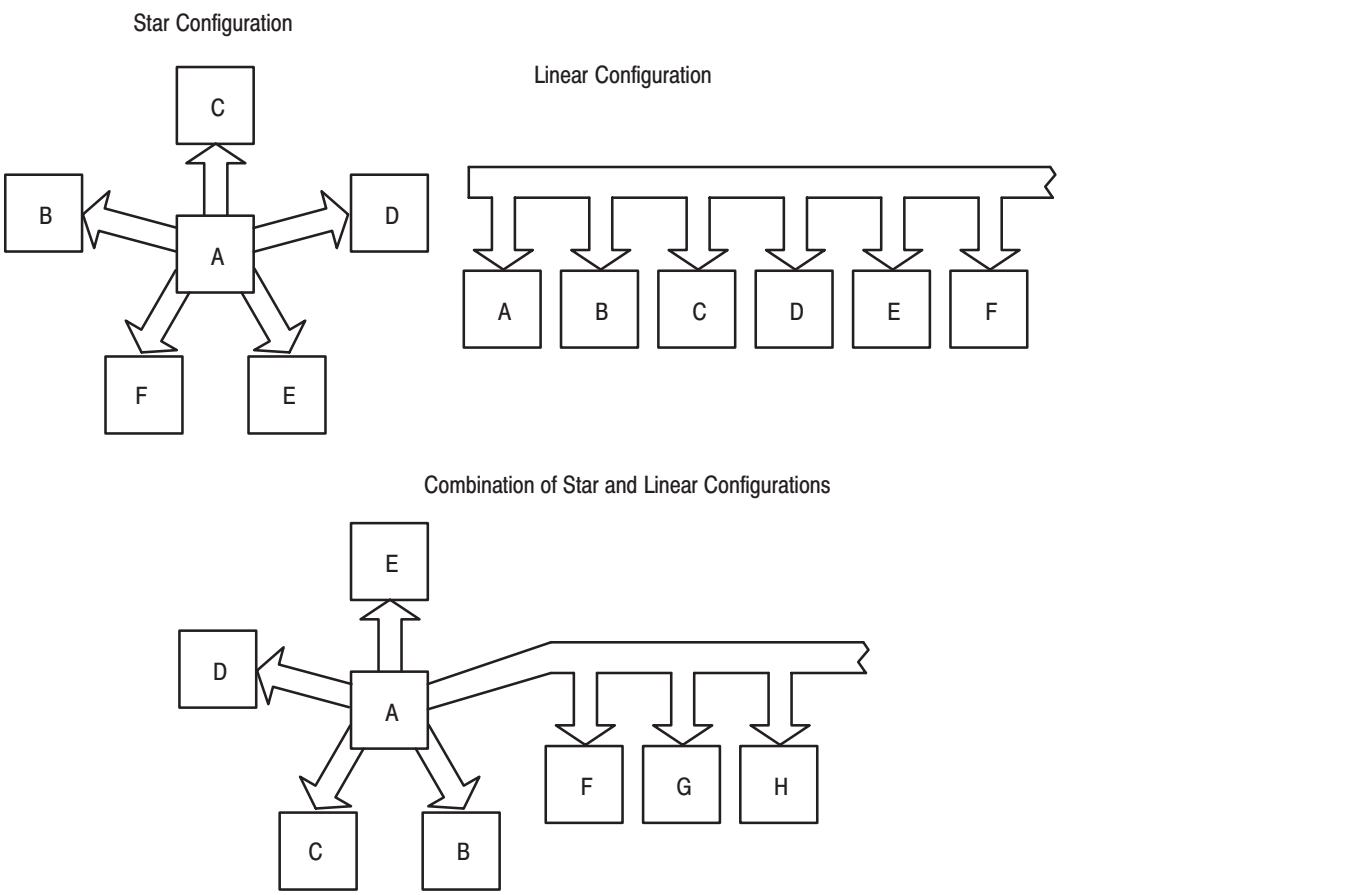


Figure 1-3: GPIB System Configurations

Restrictions

Consider the following rules when distributing instruments on the GPIB:

1. No more than 15 total devices (including the controller) can be included on a signal bus.

2. In order to maintain the electrical characteristics of the bus, one device load must be connected for every two meters of cable (most often, each device represents one device load to the bus).
3. The total cable length (cumulative) must not exceed 20 meters.
4. At least two-thirds of the device loads must be powered on.

Setting the GPIB Parameters

To access the GPIB parameters, proceed as follows:

1. Press the UTILITY button in the MENU column to the right of the screen. The UTILITY menu appears above the bottom menu buttons.
2. Press the GPIB bottom menu button to display the GPIB side menu (see Figure 1-4). The GPIB side menu displays the following items:
   - Talk/Listen, Address. Sets the communication mode to Talk/Listen, and sets the primary communication address of the waveform generator. The address range is 0 to 30.
   - Off Bus. Logically disconnects the waveform generator from GPIB system.
   - Talk only. Sets the communication mode to Talk Only to output hardcopy.

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

3. Press the Talk/Listen, Address side menu button to set the communication mode to Talk/Listen and also to assign the rotary knob to select an address. Turn the rotary knob clockwise or counterclockwise to change the address.
4. Press Misc bottom menu button, and press Config... side menu button to display Config submenu.
5. Press the Remote Port side menu button one or two times until the GPIB item highlights to select the GPIB interface as a remote interface port.

After these parameters are set, the GPIB interface is ready to operate and the GPIB indicator is highlighted in the status line on the screen (see Figure 1-11 on page 1-12).

To take the waveform generator off bus without disconnecting from the GPIB system, display the GPIB side menu as just described, but press the Off Bus side menu button to take the waveform generator off bus.

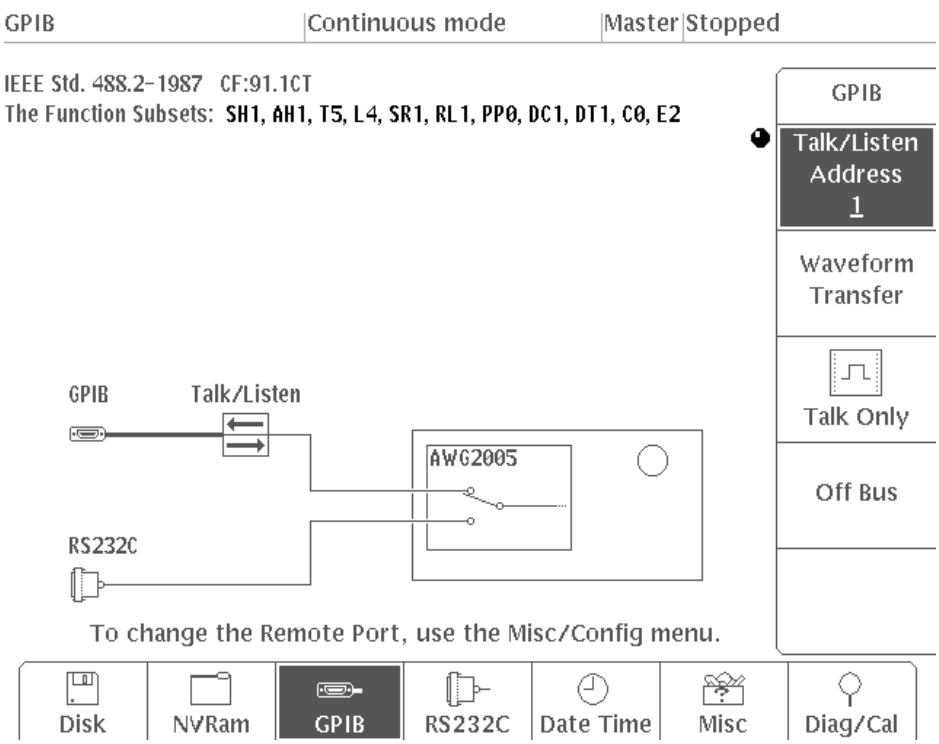


Figure 1-4: GPIB Parameter Settings

Installing for RS-232-C Communication

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

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

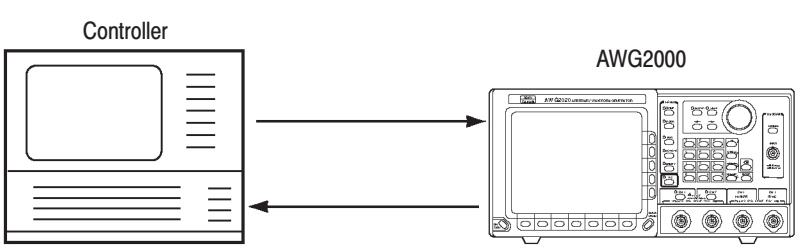


Figure 1-5: RS-232-C Point-to-Point Connection

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

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

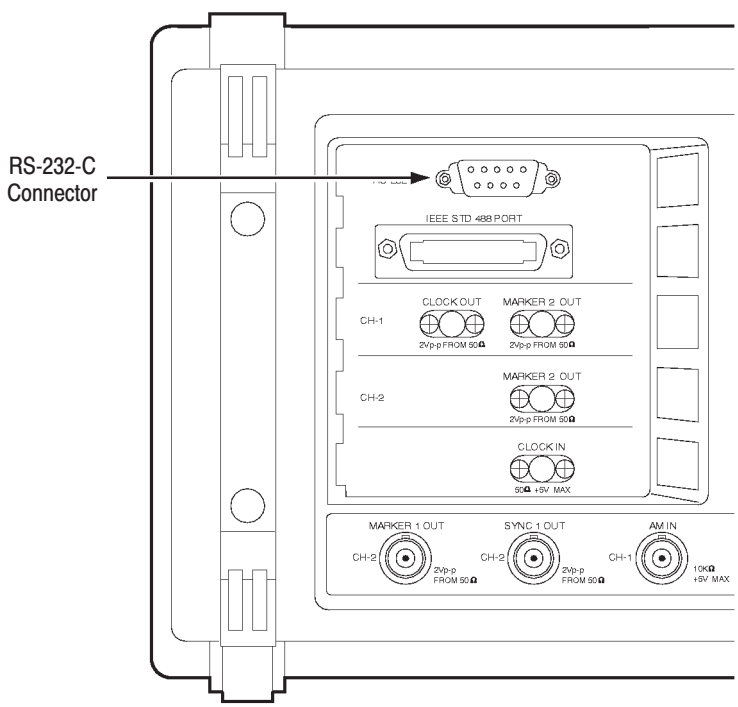
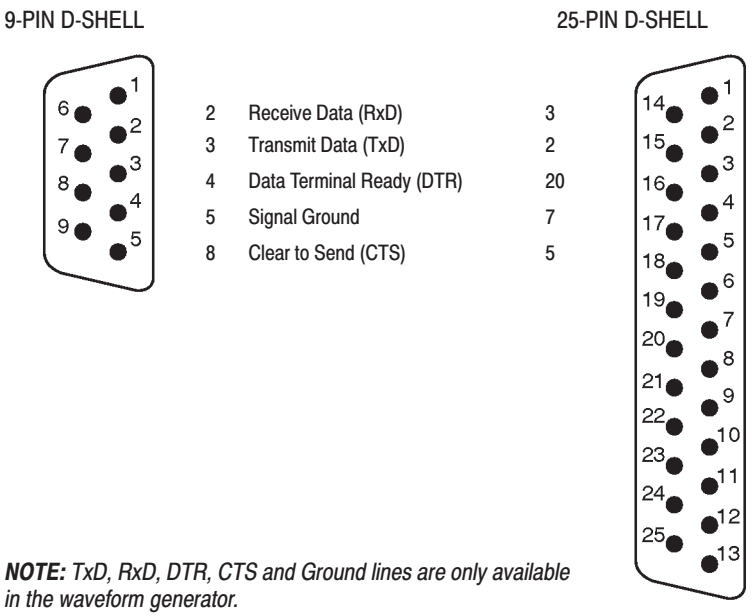


Figure 1-6: RS-232-C Port

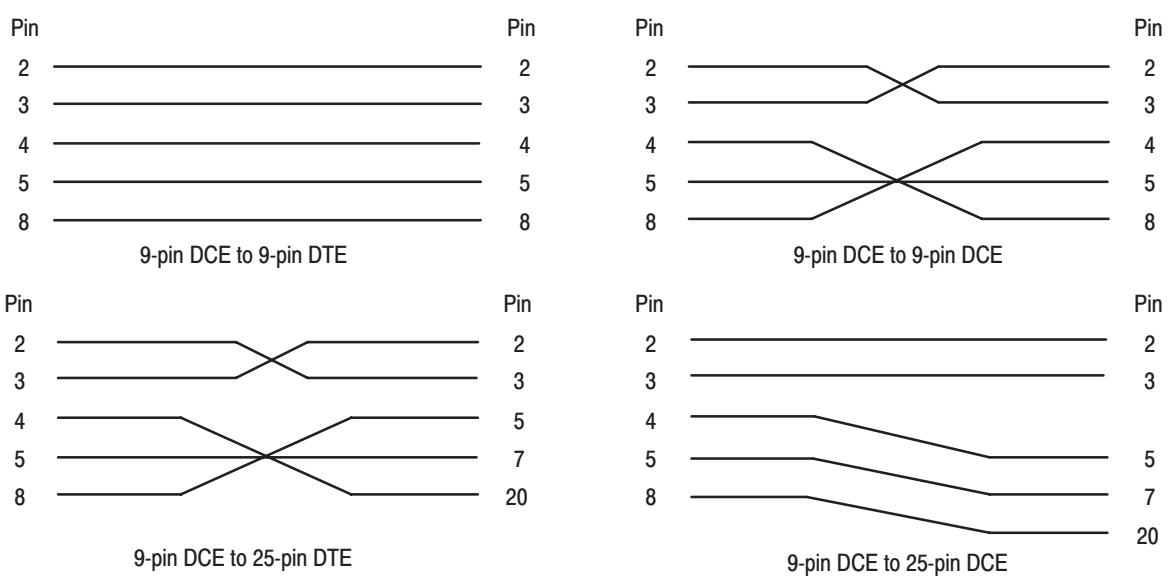
This waveform generator is designed as DCE device. You may connect it up to 15 meters (50 feet) from a DTE device using a straight-through male-to-female cable. However, if the other device is instead configured as a DCE device, you will need a special adapter or null-modem cable for local DCE-to-DCE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

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



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

Figure 1-7: Pin Assignments of 9-Pin and 25-Pin D-Type Shell Connector



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

Figure 1-8: Typical RS-232-C Cable Wiring Requirements

Setting the RS-232 Parameters

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

1. Press the UTILITY button in the MENU column to the right of the screen. The Utility menu appears above the bottom menu buttons.
2. Press the RS-232-C bottom menu button to display the RS-232-C side menu (see Figure 1-9). You may set the following parameters:
   - Baud Rate. Sets the data transmission rate. You can set rates of 300, 600, 1200, 2400, 4800, 9600, or 19200 baud.
   - Data Bits. Sets the data bit length for each character. You can set lengths of either 7 or 8 bits.
   - Parity. Sets the error check bit for each character. You can set the error bit for either None, Even, or Odd parity.
   - Stop Bits. Sets the number of stop bits sent after each character. You can set 1 or 2 stop bits.
   - Flagging. Sets the method of controlling the flow of data between devices. You can set the data flow methods Hard (DTR/CTS), Soft (XON/XOFF), or None.
3. Press, in turn, each parameter-labeled button in the menu. While any individual parameter is selected, turn the rotary knob in either direction to change the setting for the selected parameter.
4. Press Misc bottom menu button, and press Config... side menu button to display Config submenu.
5. Press the Remote Port side menu button one or two times until the RS-232-C item highlights to select the RS-232-C as a remote interface port.

After these parameters are set, the RS-232-C interface is ready to operate and the RS-232-C indicator is highlighted in the status line on the screen. (The status line is shown in Figure 1-11 on page 1-12.)

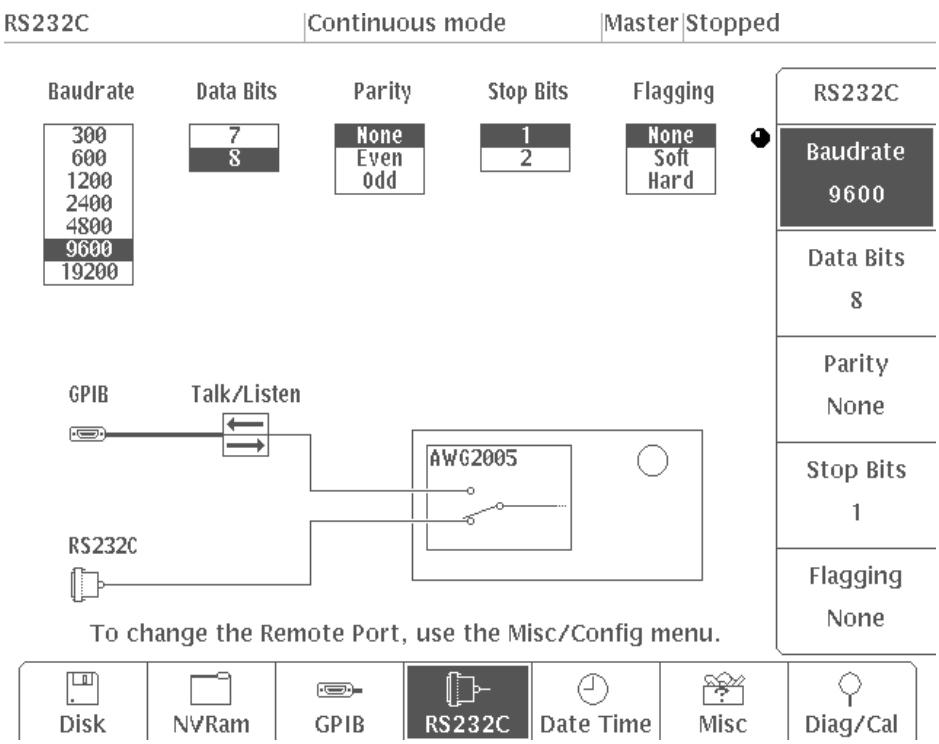


Figure 1-9: RS-232-C Parameter Settings

Confirmation of GPIB Settings

Settings for the GPIB interface can be confirmed by displaying the Status menu (see Figure 1-10). To display the System GPIB/RS-232-C Status menu, perform the following steps.

1. Press the UTILITY button in the MENU column to the right of the screen. The Utility menu appears above at the bottom of the screen.
2. Press the Misc bottom menu button to display the Misc side menu.
3. Press Status... side menu button to display Status side menu.
4. Press System side menu button to display System submenu (See Figure 1-10).

   The status of the following parameters can be confirmed in this screen:

   - Address: the current setting of the GPIB primary address
   - Configure: the current setting of the communication mode

- PSC: the current setting of PSC (Power-on Status Clear). For more details, refer to the description of *PSC common command on page 2-151 in *Section 2, Syntax and Commands*.
- Header: the current setting for header response, where 1 indicates response enabled and 0 indicates response disabled. For more details, refer to *Query Responses* on page 2-9 in *Section 2, Syntax and Commands.*
- Verbose: the current setting for header response length, where 1 indicates a long response is set and 0 a short response. For more details, refer to *Query Responses* on page 2-9 or the VERBose command, both in *Section 2, Syntax and Commands.*
- Data: the current settings of parameters related to waveform transfers. For more details, refer to the DATA:SOURce, DATA:DESTination, and DATA:ENCDG descriptions in *Section 2, Syntax and Commands.*
- Debug: the current setting for debugging parameters. For more details, refer to the DEBug description in *Section 2.*

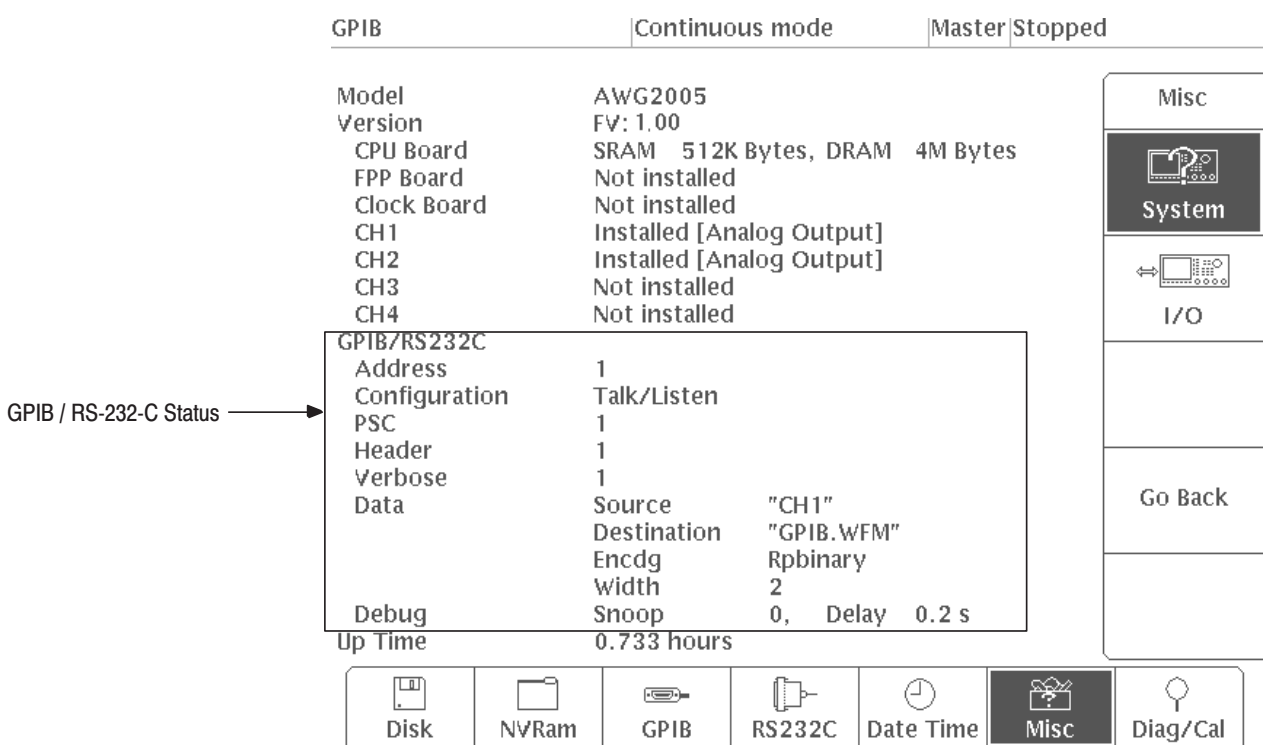


Figure 1-10: Confirmation of GPIB Settings

Operation

With the waveform generator rear-panel principal power switch turned on, turn on front-panel ON/STBY switch to obtain a screen display.

At power up, you can use either the front-panel controls or the remote interfaces as you require without any local or remote control switching required.

Figure 1-11 shows the status line on the screen. The indicators in the GPIB and RS232C status area are highlighted when the following events occur:

- GPIB. Highlights when you select the GPIB interface as a remote interface.
- RS232C. Highlights when you select the RS-232-C interface as a remote interface.
- SRQ. Highlights when the waveform generator issues an SRQ to a external controller over the GPIB.
- LOCK. Highlights when the waveform generator locks its front-panel controls in response to a command.

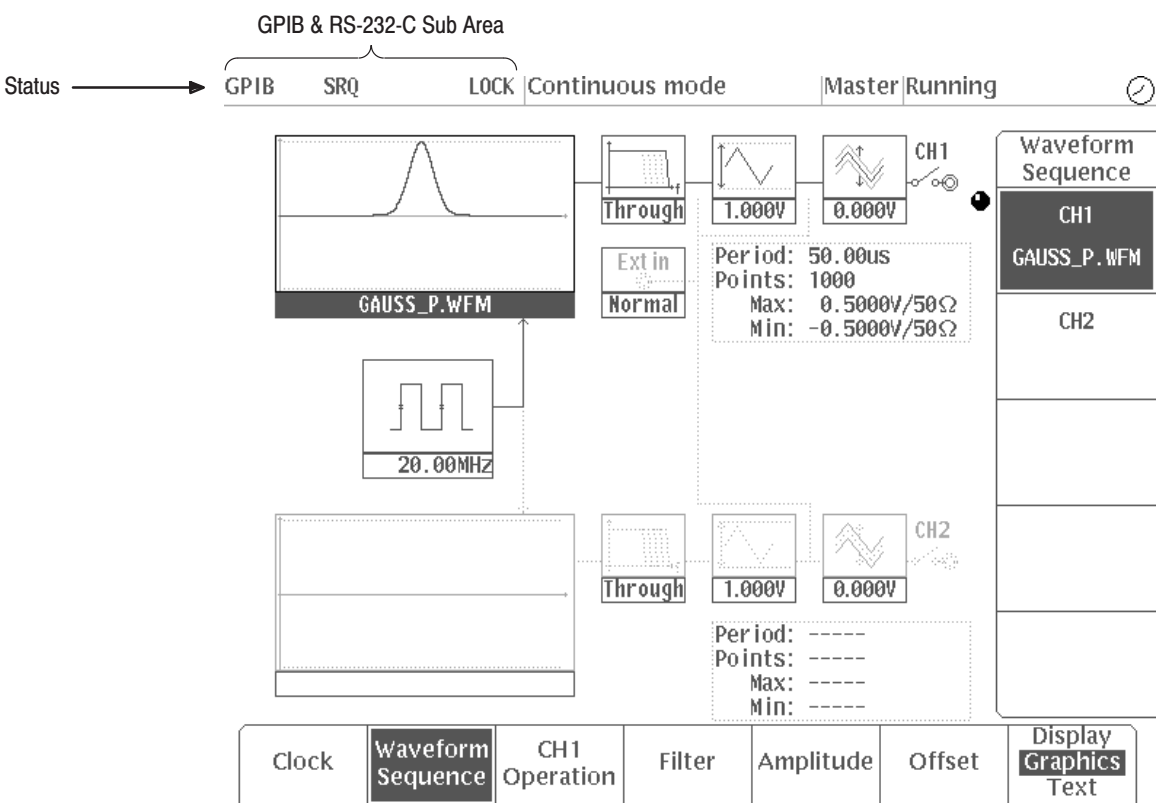


Figure 1-11: GPIB and RS-232-C Status Line

Syntax and Commands

Command Syntax

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

Command Notation

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

Table 2-1: BNF Symbols and Meanings

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

Program and Response Messages

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

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

Command and Query Structure

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

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

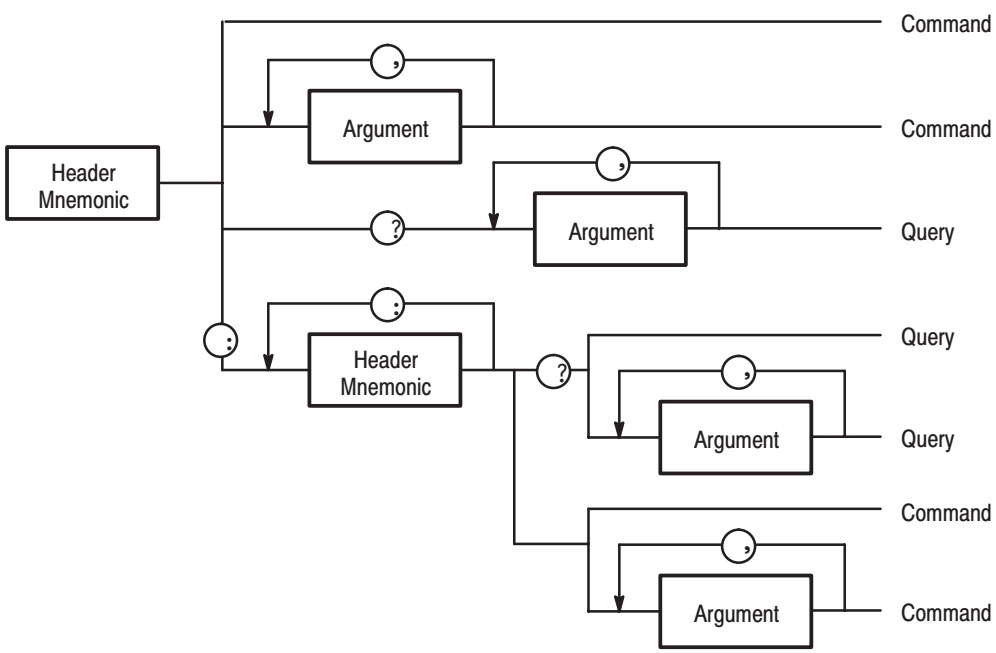


Figure 2-1: Command and Query Structure Flowchart

Character Encoding

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

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

Syntactic Delimiters

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

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

```
MMEMORY:ALOAD:MSIS, OUTPUT:CH1:STATE
```

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

```
DIAG:SELECT ALL
MODE BURST,4000
```

`DIAG:SELECT` and `MODE` are the command headers, and `ALL` and `BURST,4000` are the arguments.

Comma (,). Typically delimits between multiple arguments. In the above example, a comma delimits the multiple arguments `BURST` and `4000`.

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

White Space

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

Special Characters

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

Arguments

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

- Decimal Numeric
- String
- Arbitrary Block

Decimal Numeric

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

Table 2-2: Decimal Numeric Notation

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

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

Unit and SI Prefix

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

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

V or v for voltage

Hz, HZ, or hz for frequency

PCT, PCt, PcT, Pct, pct, pCT, or pcT, for % (percentage)

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

SI Prefix[1]	m/M	k/K	m/M	g/G
Corresponding Power	10^{-3}	10^{3}	10^{6}	10^{9}

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

String

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

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

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

```
serial number "B010000"
```

would be defined as:

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

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

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

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

Arbitrary Block

An arbitrary block argument is defined as:

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

or:

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

where:

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

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

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

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

```
#16AB4ZLT<LF><&EOI>
#0EHTGNILEDOM<LF><&EOI>
```

Header

Header Mnemonic

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

Channel and Marker Representation

In a command or query, a channel and a marker can be specified with the header mnemonics CH<x> and MARKER<x>, respectively. CH<x> can be either CH1, for channel 1, or CH2, for channel 2. Similarly, MARKER<x> can be either MARKER1 or MARKER2. The CH2 and MARKER2 header mnemonics can be used only when the channel 2 option is installed.

Header Structure

Commands and queries can be structured into six basic forms.

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

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

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

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

such as:

```
START
```

or

```
STOP
```

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

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

such as:

```
MEMORY?
```

or

```
TRIGGER?
```

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

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

such as:

```
OUTPUT:CH1:STATE ON
```

or

```
DISK:FORMAT:TYPE HD1
```

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

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

such as:

```
DISK:DIRECTORY?
```

or

```
MEMORY:CATALOG:ALL?
```

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

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

such as:

```
*RST
```

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

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

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

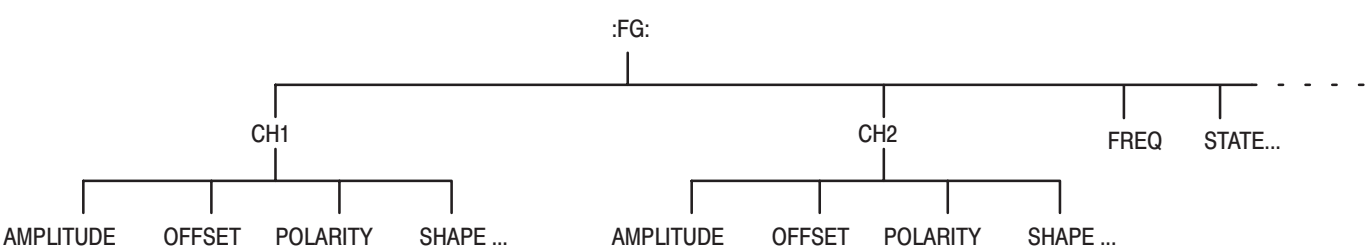
such as:

```
*IDN?
```

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

Concatenating Commands

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



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

```
:FG:CH1:AMPLITUDE 3.5; :FG:CH1:OFFSET 1.5;
:FG:CH1:POLARITY INVERTED; :FG:CH1:SHAPE SQUARE
```

You must include the complete path in each header when there is no common complete path to the start of the tree structure (the colon). However, note that part of each header in the above example has a common path `:FG:CH1`. You may

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

```
:FG:CH1:AMPLITUDE 3.5; OFFSET 1.5; POLARITY INVERT; SHAPE
SQUARE
```

Note that the mnemonics :FG and :CH1 are assumed from the first header by the headers that follow. The following command descriptions are valid examples of commands shortened using the principle just described. (Note that the insertion of common command (*SRE) between headers does not prevent the headers that follow from assuming the earlier header mnemonics.)

```
:FG:CH1:AMPLITUDE 3.5; OFFSET 1.5; :FG:CH2:AMPLITUDE 3.5;
OFFSET 1.5

:FG:STATE ON; CH1:SHAPE SQUARE; POLARITY INVERTED

:FG:CH1:AMPLITUDE 3.5; *SRE; OFFSET 1.5;
POLARITY INVERTED; SHAPE SQUARE
```

The following examples have been shortened incorrectly and cause errors.

```
:FG:CH1:AMPLITUDE 5.0; FG:CH2:AMPLITUDE 5.0

:FG:CH1:SHAPE SQU; CH2:SHAPE SQUARE

:FG:CH1:AMPLITUDE 5.0; STATE ON
```

Query Responses

The query causes the waveform generator to return information about its status or settings. A few queries also initiate an operation action before returning information; for instance, the *CAL? query runs a calibration.

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

Table 2-3: Header in Query Responses

Query	Header On	Header Off
FG:CH1:AMPLITUDE?	:FG:CH1:AMPLITUDE 5.000 V	5.000 V
DIAG:SELECT?	:DIAG:SELECT WMEMORY	WMEMORY

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

Other General Command Conventions

Upper and Lower Case

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

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

Abbreviation

Any header, argument, or reserved word that is sent to the waveform generator can be abbreviated. The minimum required spelling is shown in upper case throughout the subsection *Command Groups* beginning on page 2-13. The command CLOCk:SOURce INTernal can be rewritten in either of the following forms.

```
CLOCK:SOURCE INTERNAL
```
or
```
CLOC:SOUR INT
```

Syntax Diagrams

The syntax of each command and query is explained by syntax diagrams as well as the BNF notation. Figure 2-2 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. Command name, query name, and nonquoted string argument are abbreviated.
- Circle symbols contain separators or special symbols such as (:), (,), and (?).
- Box symbols contain the defined element.
- 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 one and only one of the paths must be taken in the command. (See the top diagram of Figure 2-2.)

- A loop around an element(s) shows the element can be repeated. (See the middle diagram.)
- A path around a group of elements shows that those elements are optional. (See bottom diagram.)

***NOTE**. The unit and SI prefix that can be added to decimal numeric arguments are not described in the syntax diagram. See* Units and SI Prefix *on page 2-4.*

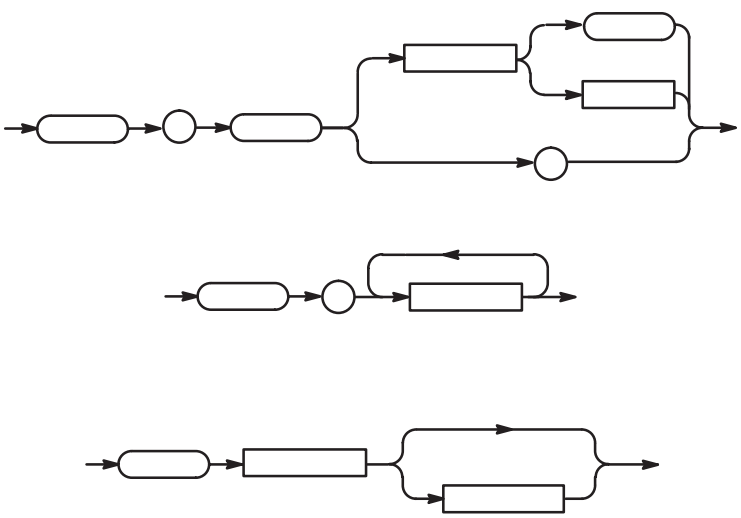


Figure 2-2: Typical Syntax Diagrams

Command Groups

This subsection describes the organization of the AWG2000 Series Arbitrary Waveform Generator command set into functional groups. (See subsection *Command Descriptions* on page 2-27 for a complete description of each command in alphabetical order.)

Throughout this section, the parenthesized question symbol (?) follows the command header to indicate that both a command and query form are included for the command.

Commands Grouped by Function

Table 2-4 lists the 12 functional groups into which the waveform generator commands are classified.

Table 2-4: Function Groups in the Command Set

Group	Functions Controlled
Calibration and Diagnostic	Control calibration and self-test diagnostics according to selected routines
Display	Control functions assigned to keys and knob, including adjusting intensity
FG	Select standard waveform functions for output and set parameters related to such waveforms. (Function generator mode.)
Hardcopy	Control start/stop of hardcopy operation, select port and its output format
Memory	Control floppy disk, internal memory, and mass memory operations
Mode	Control operating mode and set trigger parameters
Output	Turn output waveform on and off and select the sync signal position
Setup	Select clock source and its parameters
Status and Event	Set and query the registers and queues of the reporting system
Synchronization	Control operation complete and pending command execution
System	Control miscellaneous instrument functions such as data and time, local lockout, query response forms, and instrument ID
Waveform	Control transfer of waveforms

Command "Quick Reference"

The following two pages list all the commands in each functional group and can be copied for use as a quick reference. The minimum accepted character string for each command is in upper case.

Calibration and Diagnostic Commands

- SELFcal:SELect (?)
- SELFcal:STATe
- SELFcal:RESUlt?
- SELFcal?
- *CAL?
- DIAG:SELect (?)
- DIAG:STATe
- DIAG:RESUlt?
- DIAG?
- *TST?

Display Commands

- ABSTouch
- DISPlay:BRIGhtness (?)
- DISPlay:CATalog?
- DISPlay:CATalog:ORDer (?)
- DISPlay:CLOCk (?)
- DISPlay:MENU:SETUp:FORMat (?)
- DISPlay:MENU:SETUp?
- DISPlay:MENU?
- DISPlay:MESSage (?)
- DISPlay:MESSage:SHOW (?)
- DISPlay?

FG Commands

- FG:CH<x>:AMPLitude (?)
- FG:CH<x>:OFFSet (?)
- FG:CH<x>:POLarity (?)
- FG:CH<x>:SHAPe (?)
- FG:CH<x>?
- FG:FREQuency (?)
- FG:STATe (?)
- FG?

Hardcopy

- HCOPy (?)
- HCOPy:DATA?
- HCOPy:FORMat (?)
- HCOPy:PORT (?)

Memory Commands

- DISK:FORMat:TYPE (?)
- DISK:FORMat?
- DISK:FORMat:STATe
- DISK:CDIRectory
- DISK:DIRectory?
- DISK:MDIRectory
- DISK?
- MEMory:COPY
- MEMory:DELete
- MEMory:COMMent (?)
- MEMory:LOCk (?)
- MEMory:CATalog:ALL?
- MEMory:CATalog:AST?
- MEMory:CATalog:CLK?
- MEMory:CATalog:EQU?
- MEMory:CATalog:SEQ?
- MEMory:CATalog:WFM?
- MEMory:CATalog?
- MEMory:FREE:ALL?
- MEMory:FREE?
- MEMory?
- MMEMory:LOAD
- MMEMory:LOAD
- MMEMory:DELete
- MMEMory:MSIS (?)
- MMEMory:REName
- MMEMory:SAVE
- MMEMory:LOCk (?)
- MMEMory:ALOad:MSIS (?)
- MMEMory:ALOad:STATe (?)
- MMEMory:ALOad?
- MMEMory:CATalog:ALL?
- MMEMory:CATalog:AST?
- MMEMory:CATalog:CLK?
- MMEMory:CATalog:EQU?
- MMEMory:CATalog:SEQ?
- MMEMory:CATalog:WFM?
- MMEMory:CATalog?
- MMEMory:FRE:ALL?
- MMEMory:FREE?
- MMEMory?

Mode Commands

- CONFigure (?)
- MODE (?)
- RUNNing?
- STARt
- STOP
- TRIGger:IMPedance (?)
- TRIGger:LEVel (?)
- TRIGger:SLOPe (?)
- TRIGger?
- *TRG

Output Commands

- OUTPut:CH<x>:STATe (?)
- OUTPut:CH<x>?
- OUTPut:CH1:NORMal:STATe (?)
- OUTPut:CH1:INVerted?STATe (?)
- OUTPut:CH1:INVerted?
- OUTPut:CH1:NORMal?
- OUTPut:SYNC
- OUTPut?

Setup Commands

- CLOCK:FREQuency (?)
- CLOCK:SOURce (?)
- CLOCk:SWEep:DEFine (?)
- CLOCk:SWEep:DWELl (?)
- CLOCk:SWEep:FREQuency:STARt (?)
- CLOCk:SWEep:FREQuency:STOP (?)
- CLOCk:SWEep:FREQuency?
- CLOCk:SWEep:MODE (?)
- CLOCk:SWEep:STATe (?)
- CLOCk:SWEep:TIME (?)
- CLOCk:SWEep:TYPE (?)
- CLOCk:SWEep (?)
- CLOCK:CH2:DIVider (?)
- CLOCK:CH2?
- CLOCK?
- CH1:OPERation (?)
- CH<x>:AMPLitude (?)
- CH<x>:FILTer (?)
- [CH1]MARKERLEVEL1:LOW (?)
- [CH1]MARKERLEVEL1:HIGH (?)
- [CH1]MARKERLEVEL2:LOW (?)
- [CH1]MARKERLEVEL2:HIGH (?)
- [CH1]MARKERLEVEL1?
- [CH1]MARKERLEVEL2?
- CH<x>:OFFSet (?)
- CH<x>:TRACk:AMPLitude (?)
- CH<x>:TRACk:OFFSet (?)
- CH<x>:TRACk?
- CH<x>:WAVeform (?)
- CH<x>?

Status and Event Commands

- ALLEv?
- *CLS
- DESE (?)
- *ESE (?)
- *ESR?
- EVENT?
- EVMsg?
- EVQty?
- *PSC (?)
- *SRE (?)
- *STB?

Synchronization Commands

- *OPC (?)
- *WAI

System Commands

- DATE (?)
- DEBug:SNOop:STATe (?)
- DEBug:SNOop:DELAy:TIME (?)
- DEBug:SNOop:DELAy?
- DEBug:SNOop?
- DEBug:
- FACTory
- HEADer (?)
- HWSequencer?
- HWSequencer:INSTalled?
- HWSequencer:MODE (?)
- ID?
- *IDN?
- LOCk (?)
- *LRN?
- *OPT?
- *RST
- SECUre
- TIME (?)
- UPTime?
- UNLock
- VERBose (?)

Waveform Commands

- AUTOStep:DEFine (?)
- CURVe (?)
- DATA:DESTination (?)
- DATA:ENCDG (?)
- DATA:SOURce (?)
- DATA:WIDTh (?)
- EQUAtion:COMPile:STATe (?)
- DATA (?)
- EQUAtion:COMPile
- EQUAtion:DEFine (?)
- EQUAtion:WPOints (?)
- MARKER<x>:AOFF
- MARKer<x>:POINt (?)
- MARKer:AOFF
- MARKER:DATA (?)
- MARKer:POINt (?)
- SEQUence:DEFine (?)
- SEQUence:EXPAnd
- WAVFrm?
- WFMPre:ENCDG (?)
- WFMPre:BN_FMT (?)
- WFMPre:BYT_NR (?)
- WFMPre:BIT_NR (?)
- WFMPre:BYT_OR (?)
- WFMPre:CRVCHK (?)
- WFMPre:WFID (?)
- WFMPre:NR_PT (?)
- WFMPre:PT_FMT (?)
- WFMPre:XUNIT (?)
- WFMPre:XINCR (?)
- WFMPre:PT_OFF (?)
- WFMPre:XZERO (?)
- WFMPre:YUNIT (?)
- WFMPre:XMULT (?)
- WFMPre:YZERO (?)
- WFMPre:YOFF (?)
- WFMPre?

Command Summaries

Tables 2-5 through 2-16 describe each command in each of the 11 functional groups.

Calibration and Diagnostic Commands

The Calibration and Diagnostic commands perform calibration and self-test diagnostic routines.

Table 2-5: Calibration and Diagnostic Commands

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

Display Commands

The Display commands mimic manipulation of front-panel controls and set screen intensity.

Table 2-6: Display Commands

Header	Description
ABSTouch	Perform the function corresponding to the front-panel control selected
DISPlay:BRIGhtness(?)	Set brightness of screen
DISPlay?	Query settings made with display group commands
DISPlay:CATalog?	Query the condition of displaying catalog
DISPlay:CATalog:ORDer(?)	Select the order of displaying files for catalog
DISPlay:CLOCk(?)	Set date and time
DISPlay:MENU:SETUp:FORMat(?)	Set displaying format
DISPlay:MENU:SETUp?	Query displaying format
DISPlay:MENU?	Query displaying format

Table 2-6: Display Commands (Cont.)

Header	Description
DISPlay:MESSage(?)	Select on and off of displaying the message
DISPlay:MESSage:SHOW(?)	Display message

FG Commands

The FG (Function Generator) commands set the parameters, such as peak-to-peak voltage range, offset, polarity, shape, and frequency, for the waveform functions that the waveform generator outputs. They also turn the FG mode on or off. (FG mode outputs standard function waveforms, such as sine, square, and triangle waves.)

Table 2-7: FG Commands

Header	Description
FG?	Query all current settings related to the FG mode
FG:CH<x>?	Query all current settings related to FG mode for the specified channel
FG:CH<x>:AMPLitude(?)	Set the peak-to-peak voltage of the function waveform
FG:CH<x>:OFFSet(?)	Set the offset voltage of the function waveform
FG:CH<x>:POLarity(?)	Select the polarity of the function waveform
FG:CH<x>:SHAPe(?)	Select the function or type of waveform (square wave, sine wave, etc.)
FG:FREQuency(?)	Set the frequency of the function waveform
FG:STATe(?)	Turn the function generator mode on or off

Hardcopy Commands

Hardcopy commands control start and stop for hardcopy operation, and select port and its outputting format.

Table 2-8: Hardcopy Commands

Header	Description
HCOPy(?)	Control start/stop of hardcopy
HCOPy:DATA?	Query the image data of hardcopy
HCOPy:FORMat(?)	Select output format of hardcopy
HCOPy:PORT(?)	Select output port of hardcopy

Memory Commands

The Memory commands perform operations on the storage media within the waveform generator, such as formatting its floppy disk, renaming a file in its internal memory, or returning information about a file in its mass memory. Keep in mind the following points when reading about those commands in Table 2-9.

- The memory commands operate on the waveform generator floppy disk, internal memory, and mass memory storage media using the root mnemonics DISK, MEMory, and MMEMory respectively.
- Mass memory is either the waveform generator floppy disk (DISK) or its nonvolatile memory (NVRAM), according to which of these media you select. The MMEMory commands listed in the table operate on whichever of these storage media you select using MMEMory:MSIS.

Table 2-9: Memory Commands

Header	Description
DISK?	Query all the current settings related to floppy disk
DISK:CDIRectory	Change the current working directory
DISK:DIRectory?	Query the current working directory
DISK:FORMat?	Query the selected format type
DISK:FORMat:TYPE(?)	Select the type of floppy disk format
DISK:FORMat:STATe	Start formatting
DISK:MDIRectory	Create a new directory
MEMory?	Query information on all files and the size of the used and unused memory
MEMory:CATalog?	Query information on all files
MEMory:CATalog:ALL?	Query information on all files
MEMory:CATalog:AST?	Query information on all auto step files
MEMory:CATalog:CLK? (AWG2005)	Query information on all clock sweep files
MEMory:CATalog:EQU?	Query information on all equation files
MEMory:CATalog:SEQ?	Query information on all sequence files
MEMory:CATalog:WFM?	Query information on all waveform files
MEMory:COMMent(?)	Write a comment into a file in internal memory
MEMory:COPY	Copy a file in internal memory
MEMory:DELete	Delete a file
MEMory:FREE?	Query the size of the used and unused memory
MEMory:FREE:ALL?	Query the size of the used and unused memory
MEMory:LOCk(?)	Lock a file

Table 2-9: Memory Commands (Cont.)

Header	Description
MEMory:REName	Rename a file
MMEMory?	Query information on all files, used size and unused size, and status of auto-load settings
MMEMory:ALOad?	Query all current settings related to auto-load
MMEMory:ALOad:MSIS(?)	Select mass memory for auto-load
MMEMory:ALOad:STATe(?)	Define whether auto-load is enabled
MMEMory:CATalog?	Query information on all files
MMEMory:CATalog:ALL?	Query information on all files
MMEMory:CATalog:AST?	Query information on all auto-step files
MMEMory:CATalog:CLK? (AWG2005)	Query information on all clock sweep files
MMEMory:CATalog:EQU?	Query information on all equation files
MMEMory:CATalog:SEQ?	Query information on all sequence files
MMEMory:CATalog:WFM?	Query information on all waveform files
MMEMory:DELete	Delete file in mass memory
MMEMory:FREE?	Query used size and unused size
MMEMory:FREE:ALL?	Query used size and unused size
MMEMory:LOAD	Load files in mass memory to the internal memory
MMEMory:LOCk(?)	Set the lock attribute of a file
MMEMory:MSIS(?)	Select the current mass memory, DISK or NVRAM
MMEMory:REName	Rename file
MMEMory:SAVE	Save a file(s) in internal memory into current mass memory

Mode Commands

The Mode commands select the manner in which waveforms are output, such as continuously or in bursts of a certain number of waveform cycles. These commands also generate triggering events for waveforms and set trigger parameters, such as impedance, level, polarity and slope.

Table 2-10: Mode Commands

Header	Description
CONFigure(?) (AWG2005)	Select system configuration
MODE(?)	Select waveform output mode

Table 2-10: Mode Commands (Cont.)

Header	Description
RUNNing?	Query whether a waveform is currently being generated
STARt	Start the waveform output by generating a triggering event
STOP	Stop waveform from being output and initialize for output of another waveform
*TRG	Generate the triggering event (equivalent to STARt)
TRIGger?	Query all current trigger-related settings
TRIGger:IMPedance(?) (AWG2020/21/40/41)	Select the impedance presented to the the external trigger signal
TRIGger:LEVel(?)	Set the level on the external trigger signal that generates the triggering event
TRIGger:POLarity(?)	Set the polarity of external signal that generates a triggering event
TRIGger:SLOPe(?)	Select the slope of external signal that generates a triggering event

Output Commands

The Output commands turn the output waveform on or off, select the waveform output channel, and select the position on the waveform at which an external sync signal is generated. In Table 2-11, CH<x> refers to the waveform output channel, where <x> represents related channel number.

Table 2-11: Output Commands

Header	Description
OUTPut:CH<x>:STATe(?)	Turn the output on or off
OUTPut:CH<x>?	Query whether the waveform is turned on or not
OUTPut:CH1:NORMal:STATe(?) (AWG2040/41)	Turn the output on or off
OUTPut:CH1: INVerted:STATe(?) (AWG2040/41)	Turn the output on or off
OUTPut:CH1:INVerted ? (AWG2040/41)	Turn the output on or off
OUTPut:CH1:NORMal ? (AWG2040/41)	Turn the output on or off

Table 2-11: Output Commands (Cont.)

Header	Description
OUTPut:SYNC(?) (AWG2020/21)	Select position where the sync signal is generated
OUTPut?	Query all the current settings related to output

Setup Commands

The Setup commands are used to set parameters for the clock, such as clock source and frequency, and for the waveform output channel, such as the waveform amplitude or its cutoff frequency. In Table 2-12, CH<x> refers to the waveform output channel, where <x> represents related channel number.

Table 2-12: Setup Commands

Header	Description
CH1:OPERation(?) (AWG2005/20/21)	Set the mathematical operation between channels 1 and 2
CH<x>?	Query all current settings for the CH<x> waveform
CH<x>:AMPLitude(?)	Set full scale voltage for the CH<x> waveform
CH<x>:FILTer(?)	Select frequency cut-off filter for the CH<x>waveform
[CH1]MARKERLEVEL1:LOW(?) (AWG2040/41)	Set low level for marker 1
[CH1]MARKERLVEL1:HIGH(?) (AWG2040/41)	Set high level for marker 1
[CH1]MARKERLEVEL2:LOW(?) (AWG2040/41)	Set low level for marker 2
[CH1]MARKERLVEL2:HIGH(?) (AWG2040/41)	Set high level for marker 2
[CH1]MARKERLEVEL1? (AWG2040/41)	Query level setting for marker 1
[CH1]MARKERLEVEL2? (AWG2040/41)	Query level setting for marker 2
CH<x>:OFFSet(?)	Set offset voltage for the CH<x>waveform
CH<x>:TRACk:AMPLitude(?) (AWG2005/20/21)	Set tracking for voltage range
CH<x>:TRACk:OFFSet(?) (AWG2005/20/21)	Set tracking for offset voltage
CH<x>:TRACk? (AWG2005/20/21)	Query all settings for all tracking
CH<x>:WAVeform(?)	Specify the CH<x> waveform or sequence

Table 2-12: Setup Commands (Cont.)

Header	Description
CLOCk?	Query all current settings related to clock
CLOCk:CH2? (AWG2020/21)	Query all current settings related to clock for channel 2
CLOCk:CH2:DIVider(?) (AWG2020/21)	Set divide ratio to divider
CLOCk:FREQuency(?)	Set source clock frequency
CLOCk:SOURce(?)	Select clock source
CLOCk:SWEep:DEFine(?) (AWG2005)	Data transfer and writing files for clock sweep
CLOCk:SWEep:DWELl(?) (AWG2005)	Set dwell value for clock sweep
CLOCk:SWEep: FREQuency:STARt(?) (AWG2005)	Set start frequency for clock sweep
CLOCk:SWEep: FREQuency:STOP(?) (AWG2005)	Set stop frequency for clock sweep
CLOCk:SWEep:FREQuency? (AWG2005)	Query start/stop frequency for clock sweep
CLOCk:SWEep:MODE(?) (AWG2005)	Set mode for clock sweep
CLOCk:SWEep:STATe(?) (AWG2005)	Turn on or off for clock sweep
CLOCk:SWEep:TIME(?) (AWG2005)	Set time for clock sweep
CLOCk:SWEep:TYPE(?) (AWG2005)	Select type for clock sweep
CLOCk:SWEep(?) (AWG2005)	Query all settings for clock sweep

Status and Event Commands

The Status and Event commands are used by the external controller to set and query the registers and queues of the waveform generator event and status reporting system. These commands let the external controller coordinate operation between the waveform generator and other devices on the bus. For the registers and queues described in Table 2-13, refer to the status and event reporting system described in Section 4.

Table 2-13: Status and Event Commands

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

Synchronization Commands

The Synchronization commands are used by the external controller to prevent communication to the waveform generator from interfering with commands or other operations that the waveform generator is currently executing.

Table 2-14: Synchronization Commands

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

System Commands

The System commands control elements are related to the operating system of the waveform generator, such as setting date and time and locking or unlocking the front-panel controls. They also reset the system and return system-related information.

Table 2-15: System Commands

Header	Description
DATE(?)	Set date
DEBug:SNOop:STATe(?)	Turn on or off for debugging
DEBug:SNOop:DELAy:TIME(?)	Set delay time for debugging
DEBug:SNOop:DELAy?	Query delay time for debugging

Table 2-15: System Commands (Cont.)

Header	Description
DEBug:SNOop?	Query all settings for debugging
DEBug?	Query all settings for debugging
FACTory	Reset all settings to defaults
HEADer(?)	Allow or suppress the return of the control header in response messages
HWSequencer? (AWG2041)	Query the installation state and on/off state of the hardware sequencer.
HWSequencer:INSTalled? (AWG2041)	Query the state of the hardware sequencer installation.
HWSequencer:MODE(?) (AWG2041)	Set the hardware sequencer mode.
ID?	Query ID information about the waveform generator
*IDN?	Query ID information about the waveform generator
LOCk(?)	Lock or unlock local control using the front-panel controls
*LRN?	Query all settings of the waveform generator
*OPT?	Query which options are implemented for this waveform generator
*RST	Reset this waveform generator
SECUre	Clear memory to reset it to factory shipping settings
TIME(?)	Set the waveform generator time
UPTime?	Query the elapsed time since power on
UNLock	Unlock (allow) local control using the front-panel controls
VERBose(?)	Select short or long response headers

Waveform Commands

The Waveform commands control the transfer of, and parameters related to the transfer of, waveform-related information between the waveform generator and an external controller. This information includes unscaled waveform data, the waveform preamble that specifies how to reconstruct the waveform data, equations defining waveforms, and formats for transferring waveforms. Consider the following points when using waveform commands.

- Waveform data transferred includes only raw, binary-formatted data. The preamble contains the data-encoding format, waveform scale, etc., that allow a scaled waveform to be obtained.

- The CURVe command or query transfers the unscaled waveform, marker, and sequence data.
- The WAVFrm command or query transfers both the waveform and the preamble.
- The WFMPre commands and queries set up the waveform preamble.
- The DATA commands and queries specify the format and location of the waveform and marker data.
- EQUAtion commands define, compile, and otherwise control the conversion of an equation expression into a waveform.

Table 2-16: Waveform Commands

Header	Description
AUTOStep:DEFine(?)	Send the auto step data associated with the specified channel to a file in the waveform generator
CURVe(?)	Transmit waveform between the external controller and the waveform generator
DATA(?)	Query all current settings related to the waveform or marker data to be transferred
DATA:DESTination(?)	Define the destination to which the waveform is to be transferred
DATA:ENCDG(?)	Select the waveform data transfer format
DATA:SOURce(?)	Designate the source from which waveform is transferred
DATA:WIDTh(?)	Set the number of bytes per wavefrom point
EQUAtion:COMPile(?)	Compile the equation expression
EQUAtion:COMPile:STATe(?)	Compile the equation files
EQUAtion:DEFine(?)	Write the equation expression into a file
EQUAtion:WPOints(?)	Write a specified number of waveform points
MARKer:AOFF	Reset all marker data
MARKer:DATA(?)	Transmit marker data between the external controller and the waveform generator
MARKer:POINt(?)	Set marker data for specified point
MARKER<x>:AOFF	Set all markers to off
MARKER<x>:POINt(?)	Set the marker to the specified point
SEQUence:DEFine(?)	Write a sequence to a file
SEQUence:EXPAnd	Break the sequence into waveform data to generate waveform files

Table 2-16: Waveform Commands (Cont.)

Header	Description
WAVFrm?	Transmit the waveform preamble and waveform from the waveform generator to external controller
WFMPre:BIT_NR(?)	Specify the bits of precision per byte
WFMPre:BN_FMT(?)	Specify the binary data format
WFMPre:BYT_OR(?)	Specify the byte order
WFMPre:BYT_NR(?)	Specify the data field width for each binary data point
WFMPre:CRVCHK(?)	Specify the error check method
WFMPre:ENCDG(?)	Set waveform data encoding
WFMPre:NR_PT(?)	Set the number of waveform data points
WFMPre:PT_FMT(?)	Define format of data
WFMPre:PT_OFF(?)	Define the X-axis point offset value
WFMPre:WFID(?)	Set comment and additional information
WFMPre:XINCR(?)	Define the X-axis increment value
WFMPre:XUNIT(?)	Define the X-axis data unit type
WFMPre:XZERO(?)	Define the X-axis origin offset value
WFMPre:YMULT(?)	Define the Y-axis data multiplier value
WFMPre:YOFF(?)	Define the Y-axis offset value
WFMPre:YUNIT(?)	Define the Y-axis data unit type
WFMPre:YZERO(?)	Define the Y-axis origin offset value
WFMPre?	Query all the current preamble settings

Command Descriptions

This subsection lists each command and query in the AWG2000 Series Arbitrary Waveform Generators command set alphabetically. Each command entry includes its command description and command group, its related commands (if any), its syntax, and its arguments. Each entry also includes one or more usage examples.

This subsection fully spells out headers, mnemonics, and arguments with the minimal spelling shown in upper case. For example, to use the abbreviated version of the `AUTOStep:DEFine` command, just type `AUTOS:DEF`.

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

ABSTouch

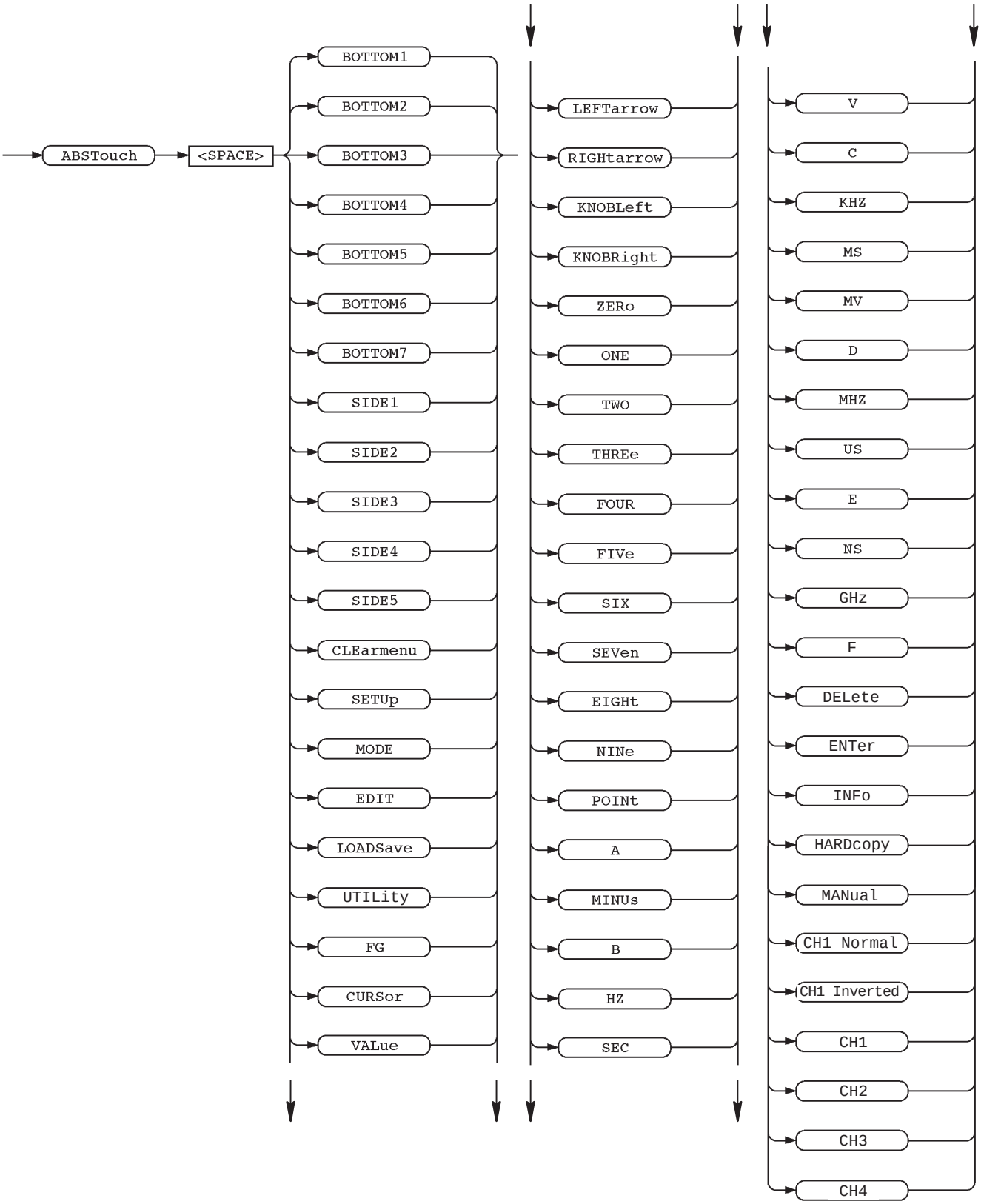
The `ABSTouch` command performs the same action that actuating the corresponding front-panel key, button, or knob would do.

Group DISPLAY

Related Commands `DISPlay? DISPlay:BRIGhtness`

Syntax

```
ABSTouch {BOTTOM1 | BOTTOM2 | BOTTOM3 | BOTTOM4 | BOTTOM5 |
BOTTOM6 | BOTTOM7 | SIDE1 | SIDE2 | SIDE3 | SIDE4 | SIDE5 |
CLEarmenu | SETUp | MODE | EDIT | LOADSave | UTILity | FG |
CURSor | VALue | LEFTarrow | RIGHtarrow | KNOBLeft | KNOBRight |
ZERo | ONE | TWO | THREe | FOUR | | FIVe | SIX | SEVen | EIGHt |
NINe | POINt | A | MINUs | B | HZ | SEC | V | C | KHZ | MS | MV |
D | MHZ | US | E | NS | GHz(AWG2040/41) | F | DELete | ENTer |
HARDcopy(AWG2005/21/40/41) | INFo | MANual | CH1(AWG2005/20/21) |
CH1Normal(AWG2040/41) | CH1Inverted(AWG2040/41) |
CH2(AWG2005/20/21) | CH3(AWG2005) | CH4(AWG2005)}
```


ABSTouch
<SPACE>
BOTTOM1
BOTTOM2
BOTTOM3
BOTTOM4
BOTTOM5
BOTTOM6
BOTTOM7
SIDE1
SIDE2
SIDE3
SIDE4
SIDE5
CLEarmenu
SETUp
MODE
EDIT
LOADSave
UTILity
FG
CURSor
VALue
LEFTarrow
RIGHtarrow
KNOBLeft
KNOBRight
ZERo
ONE
TWO
THREe
FOUR
FIVe
SIX
SEVen
EIGHt
NINe
POINt
A
MINUs
B
HZ
SEC
V
C
KHZ
MS
MV
D
MHZ
US
E
NS
GHz
F
DELete
ENTer
INFo
HARDcopy
MANual
CH1 Normal
CH1 Inverted
CH1
CH2
CH3
CH4

Arguments Sending any of the arguments that are shown in Figure 2-3 is the equivalent of operating a front panel control. The control operated is the one that the argument points to in Figure 2-3. Sending an argument corresponding to a front-panel button is the same as pressing that button once; if the argument sent corresponds to a knob, it is the same as rotating the knob clockwise or counterclockwise by $\frac{1}{25}$ of a turn.

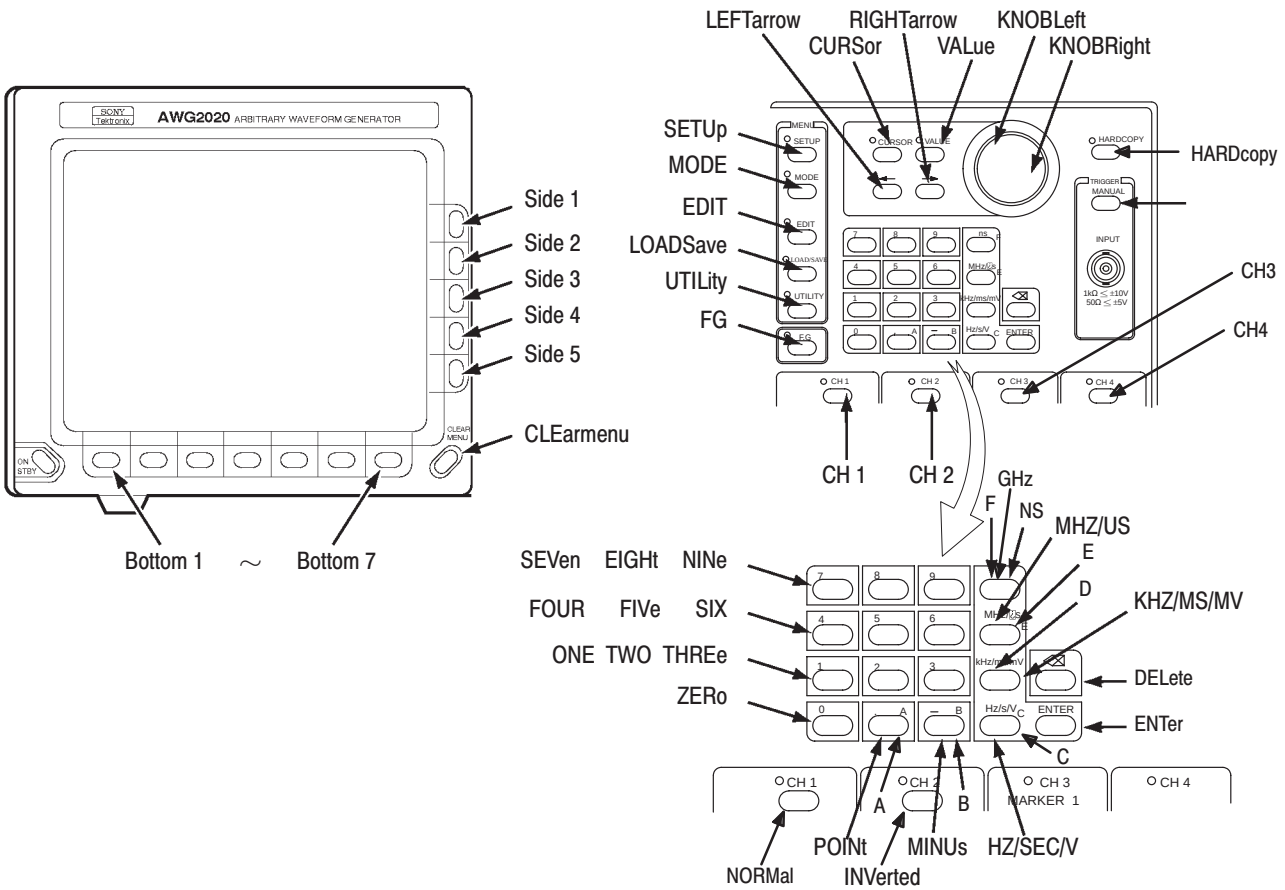


Figure 2-3: ABSTouch Arguments and Associated Controls

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

ALLEv?

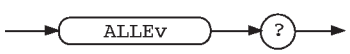
The ALLEv? query dequeues all event codes and their corresponding event messages. Use the *ESR? query to make events available for dequeuing using ALLEv? query.

Group STATUS and EVENT

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

Syntax

```
ALLEv?
```



Arguments None

Responses

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

Examples

```
ALLEV?
```

might return the string

```
:ALLEV 113,"Undefined header; unrecognized command - FG:CH1:AMP";
420, "Query UNTERMINATED".
```

AUTOStep:DEFine(?)

The AUTOStep:DEFine command sends auto-step data for the specified channel to a specified file internal to the waveform generator. The AUTOStep:DEFine? query returns the auto-step data for the specified channel from the specified file internal to the waveform generator.

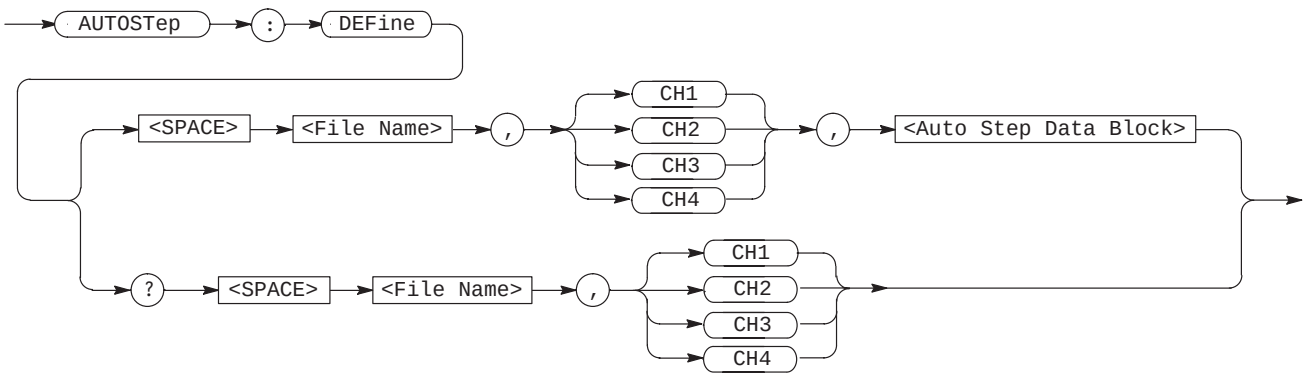
Group WAVEFORM

Related Commands

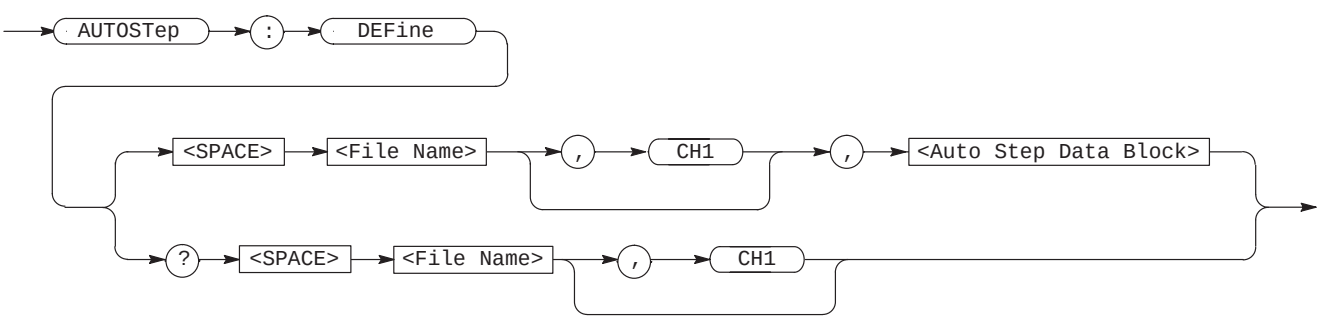
Syntax AWG2005/20/21

```
AUTOStep:DEFine <File Name>, {CH1 | CH2 | CH3 (AWG2005) |
CH4 (AWG2005)}, <Autostep Data Block>
```

AUTOStep:DEFine? <File Name>, {CH1 | CH2 | CH3 (AWG2005) | CH4 (AWG2005)}



AWG2040/41
AUTOStep:DEFine <File Name>[,CH1],<Autostep Data Block>
AUTOStep DEFine? <File Name>[,CH1]



Arguments

<File Name>::=<string>
which is the name of the file to which the auto-step data is transmitted.

CH1, CH2, CH3, CH4
which designates channel 1, channel 2, cannel 3, channel 4 respectively.

Auto-step data can be specified by ASCII code as follows.

Each auto-step is followed by comma (,), and is separated by Line Feed (LF) code.

Waveform or sequence file name <waveform>, clock source <clock source>, internal clock frequency <clock>, operation mode <operation> (AWG2005/20/21), frequency cut-off filter <filter>, output voltage range <amplitude>, offset voltage <offset>, marker 1 high level <mark 1H> (AWG2040/41), marker 1 low level <mark 1L> (AWG2040/41), marker 2 high level <mark 2H> (AWG2040/41), marker 2 low level <mark 2L> (AWG2040/41)

AWG2020/21 accepts the format which separates only waveform or sequence file name <saveform> by Line Feed (LF) code..

```
<wavefor>::=<string>
<clock>::=<NR3>[<unit1>]
<operation>::={INTernal | EXTernal}

<filter>::= {THROUGH | THR | THRU | 500KHZ | K500 | 1MHZ | M1 |
2MHZ | M2 | 5MHZ | M5 | 10MHZ | M10 | 20MHZ | M20 | 50MHZ | M50 |
100MHZ | M100}

<amplitude>::=<NR2>[<unit2>]
<offset>::=<NR2>[<unit2>]
<mark1H>::=<NR2>[<unit2>]
<mark1L>::=<NR2>[<unit2>]
<mark2H>::=<NR2>[<unit2>]
<mark2L>::=<NR2>[<unit2>]
<unit1>::=[{Hz | KHz | MHz | GHz}]
<unit2>::=[{V | mV}]
```

AWG2005/20/21

```
#3109WAVE01.WFM, INTERNAL, 10.00000E+06, NORMAL, THROUGH, 1.000,
0.000 <LF >WAVE02.WFM, 2.00000E+06, NORMAL, THROUGH, 2.000, 0.000
```

AWG2040/41

```
#3135WAVE01.WFM, INTERNAL, 1.000000E+09, 10MHZ, 1.000, 0.000, 2.0,
0.0, 2.0, 0.0 <LF> WAVE02.WFM, INTERNAL, 1.000000E+09, THROUGH,
1.000, 0.000, 2.0, 0.0, 2.0, 0.0
```

Examples

```
AUTOSTEP:DEFINE "AUTOS01.AST", CH1, #287WAVE01.WFM, 10MHZ,
NORMAL, THRU, 1.000, 0.000 <LF> WAVE02.WFM, 2.00000E+06, NORMAL,
THRU, 2.000, 0.000
```

sets the AWG2020 to transfer the auto-step data to a file AUTOSO1.AST on channel 1.

*CAL?

The *CAL? common query performs an internal calibration and returns status that indicates whether the waveform generator completes the self calibration without error. If an error is detected during calibration, execution immediately stops and an error code is returned.

***NOTE**. Up to 15 seconds are required to complete the internal calibration. During this time, the waveform generator does not respond to any commands or queries issued.*

Group CALIBRATION and DIAGNOSTIC

Related Commands SELFcal:RESUlt, SELFcal:SELect, SELFcal:STATe

Syntax *CAL?

Arguments None

Responses <Result>
where <Result>::=<NR1>, which is one of following decimal integers:

0	terminated without error.
200	detected errors in the clock unit (AWG2020/21).
600	detected errors in the setup-related unit.
800	detected errors in the trigger unit (AWG2005).

Examples *CAL?
performs an internal calibration and returns the results (for example, it might return 0, which indicates the calibration terminated without any detected errors).

CH1:OPERation (?) (AWG2005/20/21)

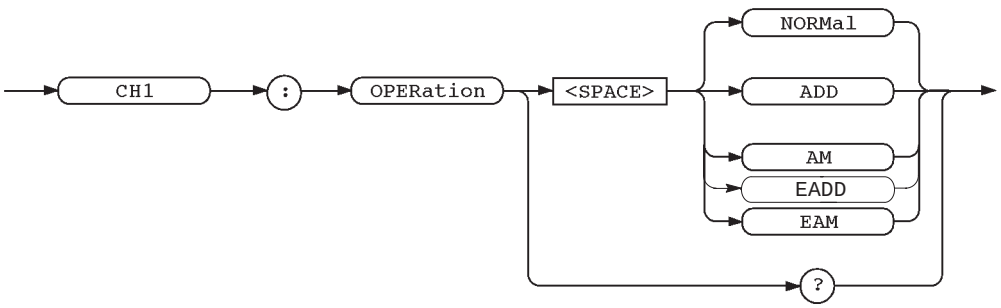
The CH1:OPERation command selects an operator that mathematically modifies the waveform on channel 1. The CH1:OPERation? query returns the currently selected operation.

Group SETUP

Related Commands CH<x>:AMPLitude, CH<x>:FILTer, CH<x>:OFFSet, CH<x>:TRACK:AMPLitude, CH<x>:TRACK:OFFSet, CH<x>:WAVeform

Syntax

```
CH1:OPERation {NORMal | ADD | AM | EADD(AWG2005) | EAM}
CH1:OPERation?
```



Arguments The choices are tabulated below.

Argument	Description
NORMal	Applies no operation to the channel 1
ADD	Adds the output of channel 2 to the channel 1 and turns off the output of channel 2. The formula below describes the output voltage Vout(t) that appears on the channel 1 connector at time t. Vout(t)=Vch1(t)+Vch2(t)+Voffset : ch1
AM	Multiplies channel 1 by the output of channel 2 and turns off the output of channel 2. The formula below describes the output voltage Vout(t) that appears on the channel 1 connector at time t. Vout(t)=Vch1(t) * Vch2+Voffset : ch1
EADD	The voltage added to EXTADD is added to channel 1 and output without change.
EAM	Multiplies channel 1 by the external signal applied through the external BNC connector. The formula below describes the output voltage Vout(t) that appears on the channel 1 connector at time t. Vout(t)=Vch1(t) * (Vext(t)+1)/2+Voffset : ch1

*** The terms Vch1(t), Vch2(t), and Vext(t) express respectively the channel 1, channel 2, and external input signal voltages before processing at time t. Voffset:ch1 and Voffset:ch2 express the settings of the channel 1 and channel 2 offset voltages.**

***NOTE**. It is possible for the voltage output from channel 1 to exceed the maximum output voltage (10 V_{p-p} for the AWG2005 and 5.0 V_{p-p} for the AWG2020/21) when calculating waveforms. Since the waveforms of such output voltages are likely to be distorted, care should be used in specifying waveform calculations.*

Examples

```
:CH1:OPERATION ADD
```

selects ADD mode.

CH<x>?

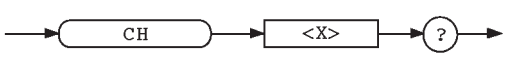
The CH<x>? query returns all current waveform output settings for the specified channel.

Group SETUP

Related Commands CH<x>:AMPLitude, CH<x>:FILTer, CH<x>:OFFSet, CH1:OPERation, CH<x>:TRACK:AMPLitude, CH<x>:TRACK:OFFSet, CH<x>:WAVeform

Syntax

```
CH<x>?
```



Arguments None

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

Examples

```
:CH1?
```

might return

AWG2005/20/21

```
:CH1:WAVEFORM "WAVE01.WFM";OPERATION NORMAL;FILTER THRU;AMPLITUDE
1.000;OFFSET 0.000
```

AWG2040/41

```
:CH1:WAVEFORM"WAVE01.WFM";FILTER THRU;AMPLITUDE 1.000;OFFSET
0.000;MARKERLEVEL1:HIGH 2.0;LOW 0.0;: H:MARKERLEVEL2:HIGH 2.0;
LOW 0.0
```

CH<x>:AMPLitude (?)

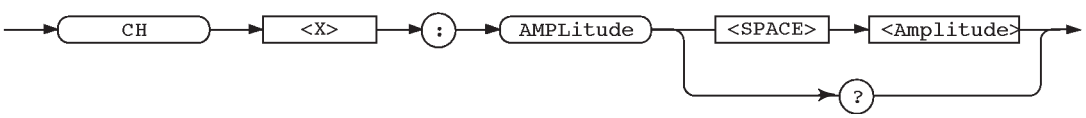
The CH<x>:AMPLitude command sets maximum full scale voltage for the waveform output at the specified channel. The CH<x>:AMPLitude? query returns the maximum voltage currently set.

Group SETUP

Related Commands CH<x>:FILTer, CH<x>:OFFSet, CH1:OPERation, CH<x>:TRACK:AMPLitude, CH<x>:TRACK:OFFSet, CH<x>:WAVeform

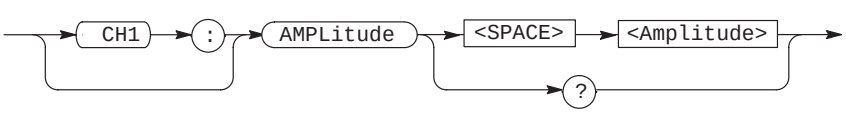
Syntax AWG2005/20/21

```
CH<x>:AMPLitude <Amplitude>
CH<x>:AMPLitude?
```



AWG2040/41

```
[CH1:]AMPLitude <Amplitude>
[CH1:]AMPLitude?
```



Arguments <Amplitude>::=<NR2>[<unit>]
where <NR2> has a range of 0.050 V to 10,000 V (AWG2005), 0.050 V to 5.000 V (AWG20/21), 0.020 V to 2.000 V (AWG2040/41) in steps of 0.001 V and <unit>::={V | mV}, for volts or millivolts.

Examples :CH1:AMPLITUDE 230.0mV
sets the amplitude of the channel 1 waveform to 230 mV.

CH<x>:FILTer (?)

The CH<x>:FILTer command selects one of four low pass filters (or selects no filter). The CH<x>:FILTer? query returns the name of the currently selected filter.

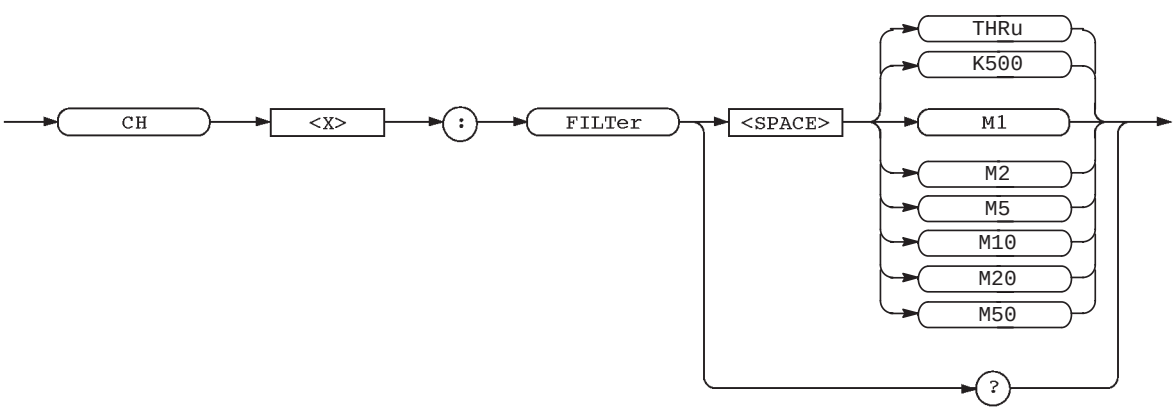
Group SETUP

Related Commands CH<x>:AMPLitude, CH<x>:OFFSet, CH<x>:OPERation, CH<x>:TRACK:AMPLitude, CH<x>:TRACK:OFFSet, CH<x>:WAVeform

Syntax AWG2005/20/21

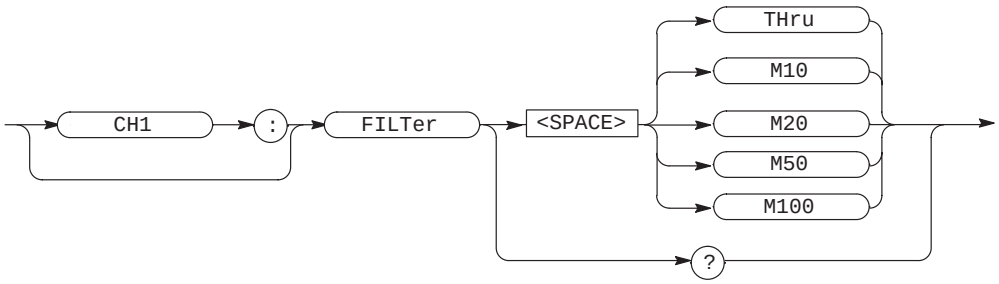
```
CH<x>:FILTer {THRu |K500(AWG2005) | M1 | M2(AWG2005) | M5
M20(AWG2020/21) | M50(AWG2020/21)}

CH<x>:FILTer?
```



AWG2040/41

```
[CH1:]FILTer {M10 | M20 | M50 | M100}
[CH1:]FILTer?
```



Arguments

THRu	OFF (no filter is used)
K500	500 kHz (AWG2005)
M1	1 MHz (AWG2005/20/21)
M2	2 MHz (AWG2005)
M5	5 MHz (AWG2005/20/21)
M10	10 MHz (AWG2040/41)
M20	20 MHz (AWG2020/21/40/41)
M50	50 MHz (AWG2020/21/40/41)
M100	100 MHz (AWG2040/41)

Examples

`:CH1:FILTER M20`
selects a low-pass filter that rolls off frequencies above a 20 MHz cut off frequency.

CH<x>:MARKERLEVEL1? (AWG2040/41)

The `CH<x>:MARKERLEVEL1?` query returns the currently specified marker 1 marker levels.

Group SETUP

Related Commands `CH<x>:MARKERLEVEL1:HIGH`,`CH<x>:MARKERLEVEL1:LOW`

Syntax `[CH1:]MARKERLEVEL1?`

Arguments None

Responses See Examples

Examples CH1:MARKERLEVEL1?
might return :CH1:MARKERLEVEL1:HIGH 2.0;LOW 0.0

CH<x>:MARKERLEVEL1:HIGH (?) (AWG2040/41)

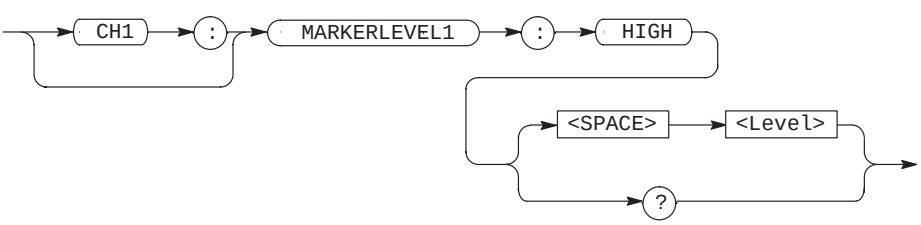
The CH<x>:MARKERLEVEL1:HIGH command sets the high level for marker 1.

The CH<x>:MARKERLEVEL1:HIGH? query returns the currently specified high level for marker 1.

Group SETUP

Related Commands CH<x>:MARKERLEVEL1:LOW

Syntax
[CH1:]MARKERLEVEL1:HIGH <Level>
[CH1:]MARKERLEVEL1:HIGH?



Arguments <Level>::=<NR2>[<unit>]
where <NR2> is a decimal number that combined with [<unit>] specifies a value in the range –1.9 V to 2.0 V in steps of 0.1 V, and [<unit>]::= {V|mV}, for volt or millivolt.
(Note that the high level must be larger than the low level.)

Examples :CH1:MARKERLEVEL1:HIGH 1.0
sets the marker 1 high level to 1 V.

CH<x>:MARKERLEVEL1:LOW (?) (AWG2040/41)

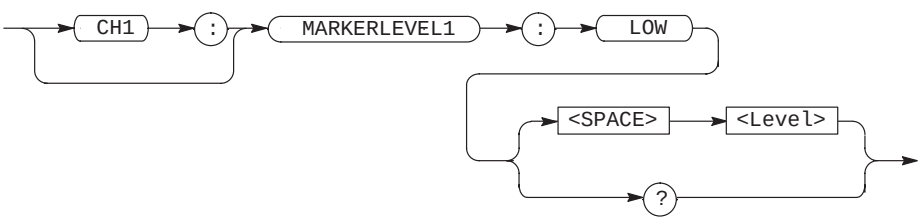
The CH<x>:MARKERLEVEL1:LOW command sets the low level for marker 1.

The CH<x>:MARKERLEVEL1:LOW? query returns the currently specified low level for marker 1.

Group SETUP

Related Commands CH<x>:MARKERLEVEL1:HIGH

Syntax
```
[CH1:]MARKERLEVEL1:LOW <Level>
[CH1:]MARKERLEVEL1:LOW?
```



Arguments <Level>::=<NR2>[<unit>]
where <NR2> is a decimal number that combined with [<unit>] specifies a value in the range –1.9 V to 2.0 V in steps of 0.1 V, and [<unit>]::= {V|mV}, for volt or millivolt.

Examples :CH1:MARKERLEVEL1:LOW 0.5
sets the marker 1 low level to 0.5 V.

CH<x>:MARKERLEVEL2? (AWG2040/41)

The CH<x>:MARKERLEVEL2? query returns the currently specified marker 2 marker levels.

Group SETUP

Related Commands CH<x>:MARKERLEVEL2:HIGH, CH<x>:MARKERLEVEL2:LOW

Syntax [CH1:]MARKERLEVEL2?

Arguments None

Responses See Examples

Examples CH1:MARKERLEVEL2?
might return :CH1:MARKERLEVEL2:HIGH 2.0;LOW 0.0

CH<x>:MARKERLEVEL2:HIGH (?) (AWG2040/41)

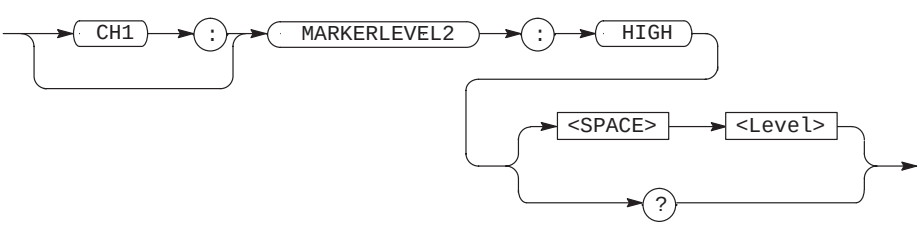
The CH<x>:MARKERLEVEL2:HIGH command sets the high level for marker 2.

The CH<x>:MARKERLEVEL2:HIGH? query returns the currently specified high level for marker 2.

Group SETUP

Related Commands CH<x>:MARKERLEVEL2:LOW

Syntax [CH1:]MARKERLEVEL2:HIGH <Level>
[CH1:]MARKERLEVEL2:HIGH?



Arguments <Level>::=<NR2>[<unit>]
where <NR2> is a decimal number that combined with [<unit>] specifies a value in the range –1.9 V to 2.0 V in steps of 0.1 V, and [<unit>]::= {V|mV}, for volt or millivolt.
(Note that the high level must be larger than the low level.)

Examples :CH1:MARKERLEVEL2:HIGH 1.0
sets the marker 2 high level to 1 V.

CH<x>:MARKERLEVEL2:LOW (?) (AWG2040/41)

The CH<x>:MARKERLEVEL2:LOW command sets the low level for marker 2.

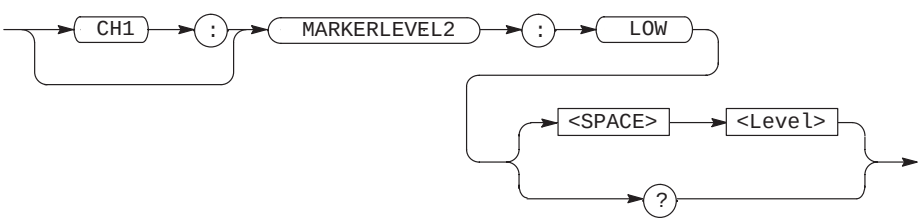
The CH<x>:MARKERLEVEL2:LOW? query returns the currently specified low level for marker 2.

Group SETUP

Related Commands CH<x>:MARKERLEVEL2:HIGH

Syntax

```
[CH1:]MARKERLEVEL2:LOW <Level>
[CH1:]MARKERLEVEL2:LOW?
```



Arguments <Level>::=<NR2>[<unit>]
where <NR2> is a decimal number that combined with [<unit>] specifies a value in the range –1.9 V to 2.0 V in steps of 0.1 V, and [<unit>]::= {V|mV}, for volt or millivolt.
(Note that the high level must be larger than the low level.)

Examples :CH1:MARKERLEVEL2:LOW 1.5
sets the marker 2 low level to 1.5 V.

CH<x>:OFFSet (?)

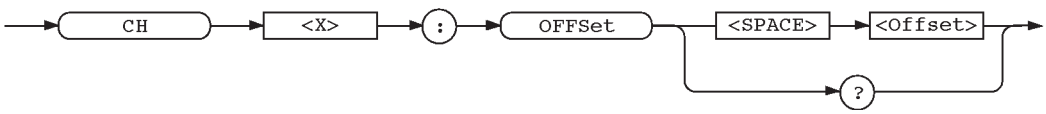
The CH<x>:OFFSet command sets the offset voltage of waveforms output from the specified channel. The CH<x>:OFFSet? query returns the offset voltage currently set.

Group SETUP

Related Commands CH<x>:AMPLitude, CH<x>:FILTer, CH<1>:OPERation, CH<x>:TRACK:AMPLitude, CH<x>:TRACK:OFFSet, CH<x>:WAVeform

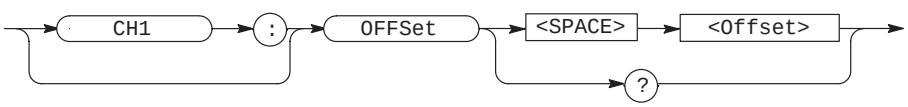
Syntax AWG2005/20/21

```
CH<x>:OFFSet <Offset>
CH<x>:OFFSet?
```



AWG2040/41

```
[CH1:] OFFSet <Offset>
[CH1:] OFFSet ?
```



Arguments `<Offset>::=<NR2>[<unit>]`
where `<NR2>` has a range of –5.000 V to 5.000 V (AWG2005), –2.500 V to 2.500 V (AWG2020) in steps of 0.005 V, and –1.000 V to 1.000 V in steps of 0.001 V (AWG2040/41) and `<unit>::={V | mV}`.

Examples `:CH1:OFFSET 50.0mV`
sets the offset voltage of channel 1 to 50 mV.

CH<x>:TRACk? (AWG2005/20/21)

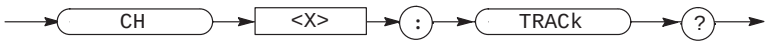
The `CH<x>:TRACk?` query returns all amplitude and offset linkage settings for the specified channel.

Note that only CH2, CH3, and CH4 are valid header mnemonics.

Group SETUP

Related Commands `CH<x>:TRACk:AMPLitude`, `CH<x>:TRACk:OFFSet`

Syntax `CH<x>:TRACk?`



Arguments None

Responses See Examples

Examples CH2:TRACK?
might return :CH2:TRACK:AMPLITUDE OFF;OFFSET OFF

CH<x>:TRACk:AMPLitude (?) (AWG2005/20/21)

The CH<x>:TRACk:AMPLitude command sets the amplitude linkage for the channel specified in the header.

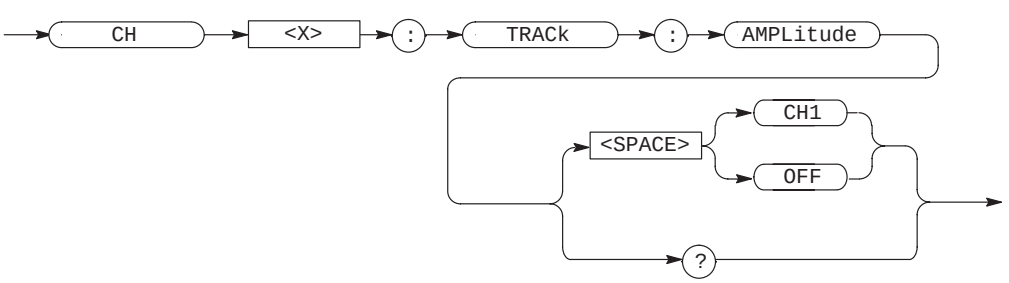
The CH<x>:TRACk:AMPLitude? query returns the amplitude linkage for the channel specified in the header from the settings.

Note that only CH2, CH3, and CH4 are valid header mnemonics.

Group SETUP

Related Commands CH<x>:TRACk?, CH<x>:TRACk:OFFSet

Syntax CH<x>:TRACk:AMPLitude {CH1 | OFF}
CH<x>:TRACk:AMPLitude?



Arguments CH1
links the specified channel to the channel 1 voltage range.

OFF
does not use the amplitude linkage function.

Examples CH2:TRACK:AMPLITUDE CH1
links the channel 2 voltage range to the channel 1 voltage range.

CH<x>:TRACk:OFFSet (?) (AWG2005/20/21)

The CH<x>:TRACk:OFFSet command sets the offset linkage for the channel specified in the header.

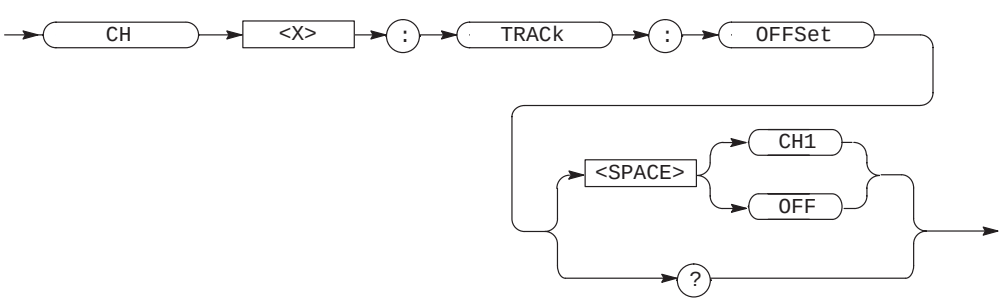
The CH<x>:TRACk:OFFSet? query returns the offset linkage for the channel specified in the header from the settings.

Note that only CH2, CH3, and CH4 are valid header mnemonics.

Group SETUP

Related Commands CH<x>:TRACk?, CH<x>:TRACk:AMPLitude

Syntax
```
CH<x>:TRACk:OFFSet {CH1 | OFF}
CH<x>:TRACk:OFFSet?
```



Arguments CH1
links the specified channel to the channel 1 voltage range.

OFF
does not use the offset linkage function.

Examples :CH2:TRACK:OFFSET CH1
links the channel 2 voltage range to the channel 1 voltage range.

CH<x>:WAVeform (?)

The CH<x>:WAVeform command selects a waveform or a sequence for output on the specified channel. The CH<x>:WAVeform? query returns the currently specified waveform or sequence file on the specified channel.

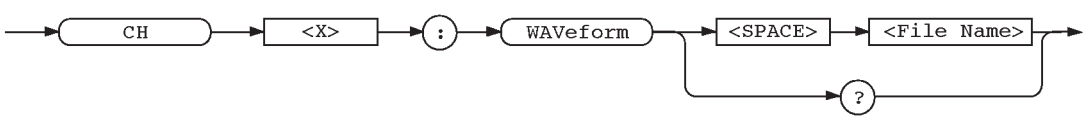
If the dual channel option (Option 02) is not installed, CH1 is only valid header mnemonic.

Group SETUP

Related Commands CH<x>:AMPLitude, CH<x>:FILTer, CH<x>:OFFSet, CH<x>:TRACk:AMPLitude, CH<x>:TRACk:OFFSet, CH<1>:OPERation

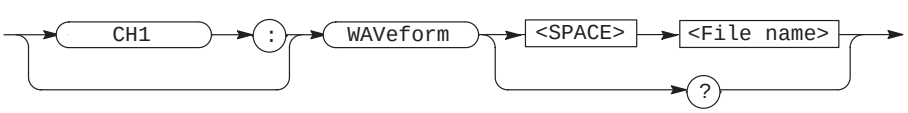
Syntax AWG2005

```
CH<x>:WAVeform <File Name>
CH<x>:WAVeform?
```



AWG2040/41

```
[CH1:]WAVeform <File name>
[CH1:]WAVeform?
```



Arguments <File Name>::=<string>
where <string> is a waveform file name or sequence file name.

Examples :CH1:WAVEFORM "SQUARE.WFM"
selects the waveform in the waveform file SQUARE.WFM as the waveform output on channel 1.

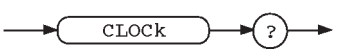
CLOCk?

The CLOCk? query returns all clock settings.

Group SETUP

Related Commands CLOCk:FREQuency, CLOCk:SOURce, CLOCk:CH2:DIVider, CLOCK:SWEep

Syntax CLOCk?



Arguments None

Examples :CLOCK?
might return CLOCK:FREQUENCY 1.000E+08; SOURCE INTERNAL;
CH2 DIVIDER 1

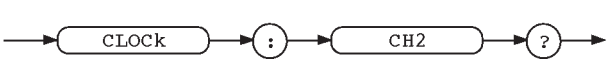
CLOCk:CH2? (AWG2020/21)

The CLOCk:CH2? query returns all clock settings currently set for channel 2. This command is effective only when the dual channel option (Option 02) is installed.

Group SETUP

Related Commands CLOCk:CH2:DIVider

Syntax CLOCk:CH2?



Arguments None

Examples :CLOCK:CH2?
might return CLOCK:CH2:DIVIDER 1

CLOCk:CH2:DIVider (?) (AWG2020/21)

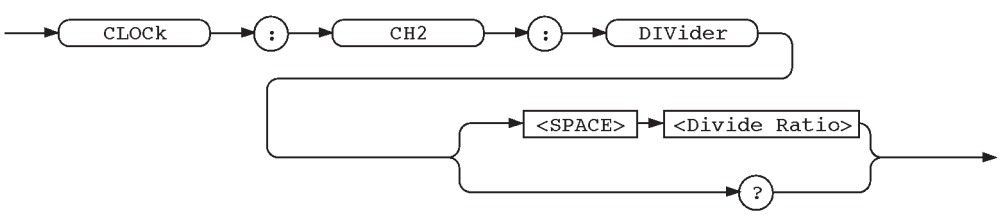
The CLOCk:CH2:DIVider command sets the ratio for the clock divider. The divided clock frequency is the channel 2 clock frequency. The CLOCk:CH2:DIVider? query returns the divide ratio currently set.

This command is effective only when the dual channel option (Option 02) is installed.

Group SETUP

Related Commands CLOCk?, CLOCk:FREQuency, CLOCk:SOURce

Syntax
```
CLOCk:CH2:DIVider <Divide Ratio>
CLOCk:CH2:DIVider?
```



Arguments
<Divide Ratio>::=<NR1>
where <NR1> has a range from 1 to 2^{24}; (<NR1> must be a power of 2).

Examples
:CLOCK:CH2:DIVIDER 256
sets the divide ratio to 256 (2^8).

:CLOCK:CH2:DIVIDER?
might return CLOCK:CH2:DIVIDER 256

CLOCk:FREQuency (?)

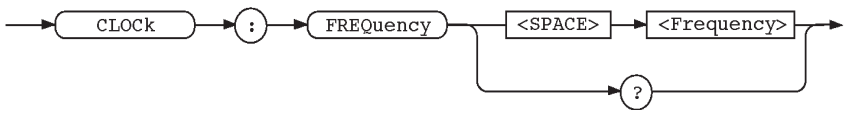
The CLOCk:FREQuency command sets source clock frequency. The CLOCk:FREQuency? query returns the frequency currently set.

This command is effective only when the internal clock source is selected.

Group SETUP

Related Commands CLOCk:SOURce, CLOCk:CH2:DIVider

Syntax
```
CLOCk:FREQuency <Frequency>
CLOCk:FREQuency?
```



Arguments <Frequency>::=<NR3>[<unit>]
where <NR3> is a decimal number that combines with [<unit>] to have a range of 10.00E–3 ～ 20.00E+6Hz (AWG2005), 10.0～250.0E+6Hz (AWG2020/21), 1.000000E+3 ～1.024000E+9Hz (AWG2040/41), and[<unit>]::={HZ | KHZ | MHZ | GHz(AWG2040/41)}, for hertz, kilohertz, megahertz or gigahertz.

Examples :CLOCK:SOURCE INTERNAL; FREQUENCY 245.0KHZ
selects internal clock as a clock source and sets the frequency to 245 kHz.

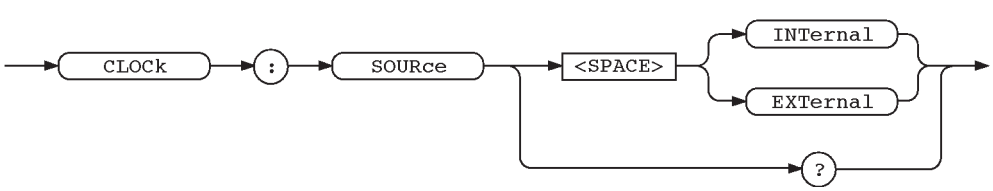
CLOCk:SOURce (?)

The CLOCk:SOURce command selects clock source. The CLOCk:SOURce? query returns the currently selected clock source.

Group SETUP

Related Commands CLOCk?, CLOCk:FREQuency, CLOCk:CH2:DIVider, CLOCk:SWEep:STATe

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



Arguments

`INTernal`
use the internal clock source.

`EXTernal`
use the external clock source supplied through the external connector.

Examples

`:CLOCK:SOURCE EXTERNAL`
selects the external clock source.

CLOCk:SWEep:DEFine (?) (AWG2005)

The `CLOCK:SWEep:DEFine` command writes the arbitrary clock sweep data in a specified file.

The `CLOCK:SWEep:DEFine?` query returns the arbitrary clock seep data written in a specified file.

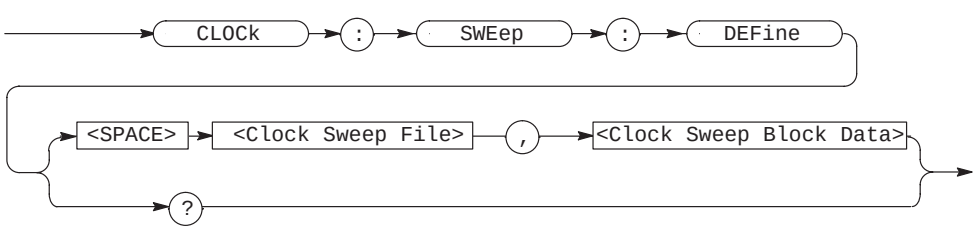
This command is effective only when Option 05 (clock sweep) is installed.

Group SETUP

Related Commands `CLOCk:SWEep:DWEL1, CLOCk:SWEep:TYPE`

Syntax

```
CLOCk:SWEep:DEFine <Clock Sweep File>,<Clock Sweep Block Data>
CLOCk:SWEeep:DEFine? <Clock Sweep File>
```



Arguments

`<Clock Sweep File>::=<string>` Clock sweep file

`<Clock Sweep Block Data>::=<Arbitrary Block>` Clock sweep data

Clock sweep data is specified in dwell time <dwell> followed by frequency <clock> and hold–bit <event> in every step in binary.

```
<dwell><clock(1)><event (1)><clock
(2)><event(2)>...<clock(N)><event(N)>
```

<dwell>::=double precision floating point

<clock>::=double precision floating point

<event>::=16–bit interger without sign

Examples `:CLOCK:SWEEP:DEFINE "SWEEP.CLK", #510008...`
specifies the 1000–step clock sweep data in the clock sweep file SWEEP.CLK.

CLOCk:SWEep:DWELl (?) (AWG2005)

The `CLOCk:SWEep:DWELl` command sets the length of the period for which a single frequency is output when the clock sweep type is “arbitrary”.

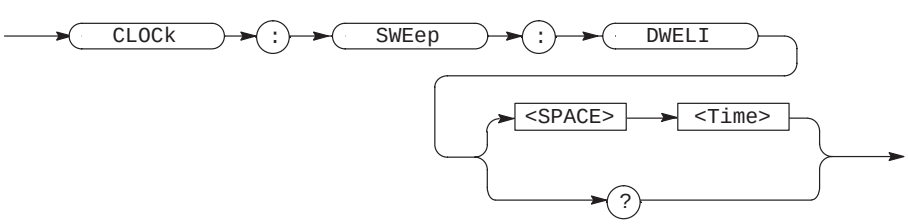
The `CLOCk:SWEep:DWELl?` query returns the length of the period for which a single frequency is output.

This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands `CLOCk:SWEep:TYPE`

Syntax
```
CLOCk:SWEep:DWELl <Time>
CLOCk:SWEep:DWELl?
```



Arguments `<Time>::=<NR3>[<unit>]`
where `<NR3>` combined with `[<unit>]` specifies a time in the range 1 μs to 65.535 ms, and `[<unit>] ::={s|ms|μs}`, for seconds, milliseconds, or microseconds.

Examples `:CLOCK:SWEEP:DWELl 1MS`
specifies the output of a single frequency for a period of 1 ms.

CLOCk:SWEep:FREQuency? (AWG2005)

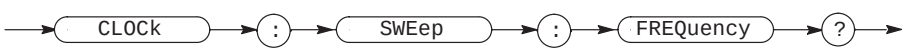
The CLOCk:SWEep:FREQuency? query returns the clock sweep start and stop frequencies.

This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands CLOCk:SWEep:FREQuency:STARt,CLOCk:SWEep:FREQuency:STOP, CLOCk:SWEep:TYPE

Syntax CLOCk:SWEep:FREQuency?



Arguments None

Responses See Examples

Examples CLOCK:SWEep:FREQuency?
might return :CLOCK:SWEEP:FREQUENCY:START 1.00000E+06;STOP 20.0000E+06

CLOCk:SWEep:FREQuency:STARt (?) (AWG2005)

The CLOCk:SWEep:FREQuency:STARt command sets the clock sweep start frequency.

The CLOCk:SWEep:FREQuency:STARt? query returns the clock sweep start frequency.

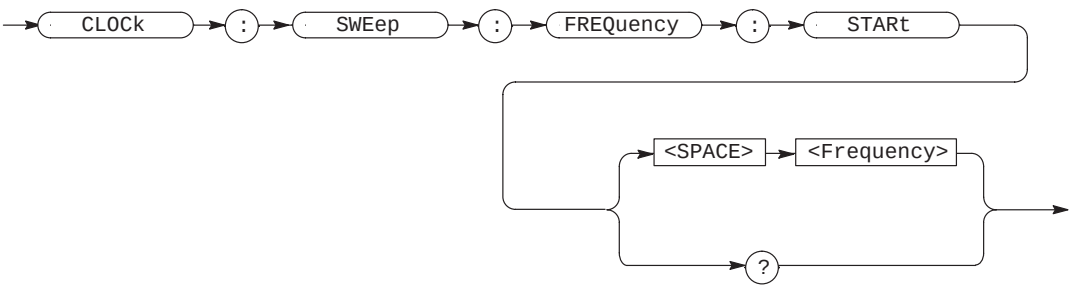
This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands CLOCk:SWEep:FREQuency?, CLOCk:SWEep:FREQuency:STOP, CLOCk:SWEep:TYPE

Syntax

```
CLOCk:SWEep:FREQuency:STARt <Frequency>
CLOCk:SWEep:FREQuency:STARt?
```



Arguments

<Frequency>::=<NR3>[<unit>]
where <NR3> combined with [<unit>] specifies a value in the range 0.01 Hz to 20.0000 MHz, and <unit>::={Hz|KHz|MHz}, for hertz, kilohertz or megahertz.

Examples

```
:CLOCK:SWEEP:FREQUENCY:START 1000
```
sets the clock sweep start frequency.

CLOCk:SWEep:FREQuency:STOP (?) (AWG2005)

The CLOCk:SWEep:FREQuency:STOP command sets the clock sweep stop frequency.

The CLOCk:SWEep:FREQuency:STOP? query returns the clock sweep stop frequency.

This command is effective only when the Option 05 (clock sweep) is installed.

Group

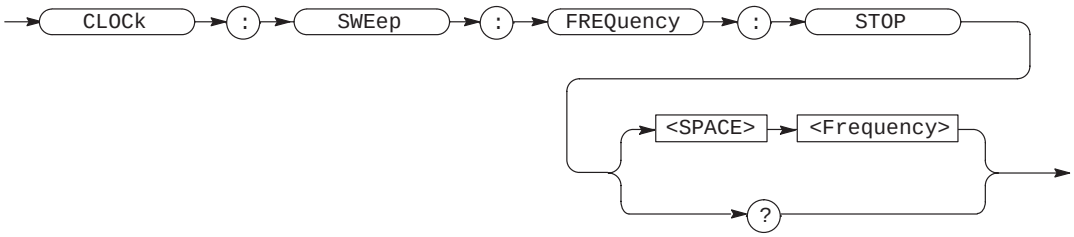
SETUP

Related Commands

CLOCk:SWEep:FREQuency?, CLOCk:SWEep:FREQuency:STARt, CLOCk:SWEep:TYPE

Syntax

```
CLOCk:SWEep:FREQuency:STOP <Frequency>
CLOCk:SWEep:FREQuency:STOP?
```



Arguments <Frequency>::=<NR3>[<unit>]
where <NR3> combined with [<unit>] specifies a value in the range 0.01 Hz to 20.0000 MHz, and <unit>::={Hz|KHz|MHz}, for hertz, kilohertz or megahertz.

Examples :CLOCK:SWEEP:FREQUENCY:STOP 20KHZ
sets the clock sweep stop frequency.

CLOCk:SWEep:MODE (?) (AWG2005)

The CLOCk:SWEep:MODE command sets the sweep mode.

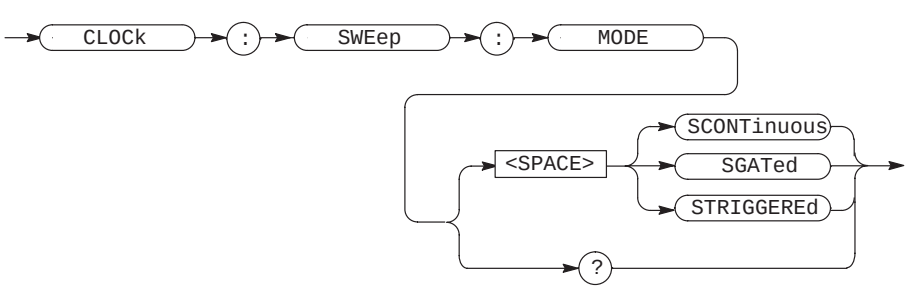
The CLOCk:SWEep:MODE? query returns the currently specified sweep mode.

This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands CLOCk:SWEep:STAte,MODE

Syntax CLOCk:SWEep:MODE { SCONTinuous| SGATed| STRIGGERED}
CLOCk:SWEep:MODE?



Arguments

SCONTinuous
sets the sweep mode to Continuous mode. In Continuous mode, the sweep operation is performed continuously.

SGATed
sets the sweep mode to Gated mode. In Gated mode, the sweep operation is performed only when the gate signal is valid.

STRIGGERed
sets the sweep mode to Triggered mode. In Triggered mode, the sweep operation is performed each time a trigger occurs.

Examples

```
:CLOCK:SWEEP:MODE SGATE
```
sets the sweep mode to Gated mode.

CLOCk:SWEep:STATe (?) (AWG2005)

The CLOCK:SWEep:STATe command turns the clock sweep on or off.

The CLOCK:SWEep:STATe? query returns whether or not the clock sweep is on.

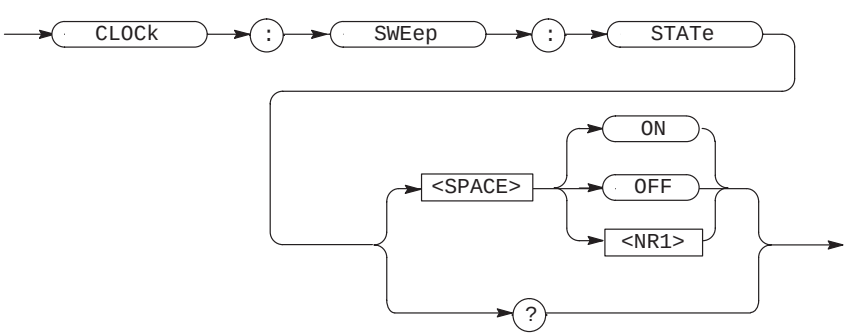
This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands CLOCk:SOURce,MODE

Syntax

```
CLOCk:SWEep:STATe {ON | OFF | <NR1>}
CLOCk:SWEep:STATe?
```



Arguments

ON or nonzero value
turns the clock sweep mode on.

OFF or zero value
turns the clock sweep off.

Responses

1 clock sweep is currently turned on.
0 clock sweep is currently turned off.

Examples

:CLOCK:SWEEP:STATE ON
turns the clock sweep on.

CLOCk:SWEep:TIME (?) (AWG2005)

The CLOCk:SWEep:TIME command sets the length of the period from the start to the end of the sweep.

The CLOCk:SWEep:TIME? query returns the length of the period from the start to the end of the sweep.

This command is effective only when the Option 05 (clock sweep) is installed.

Group

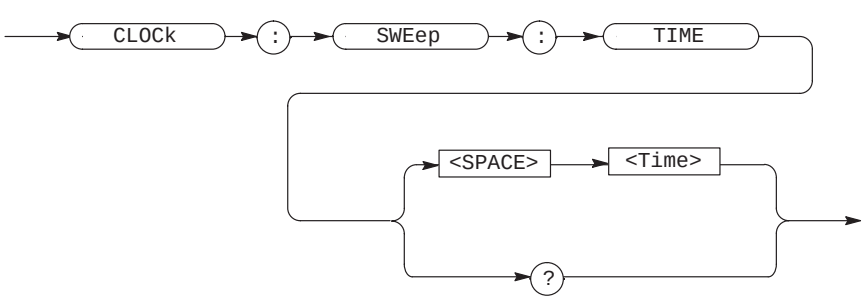
SETUP

Related Commands

CLOCk:SWEep:TYPE

Syntax

CLOCk:SWEep:TIME <Time>
CLOCk:SWEep:TIME?



Arguments

<Time>::=<NR3>[<unit>]
where <NR3> combined with [<unit>] specifies a time in the range 1 ms to 65.535 s, and [<unit>]::= {s|ms|μs}, for seconds, milliseconds, or microseconds.

Examples

```
:CLOCK:SWEEP:TIME 5MS
```

sets the length of the period from the start to the end of the sweep to 5 ms.

CLOCk:SWEep:TYPE (?) (AWG2005)

The CLOCk:SWEep:TYPE command sets the clock sweep type.

The CLOCk:SWEep:TYPE? query returns the clock sweep type.

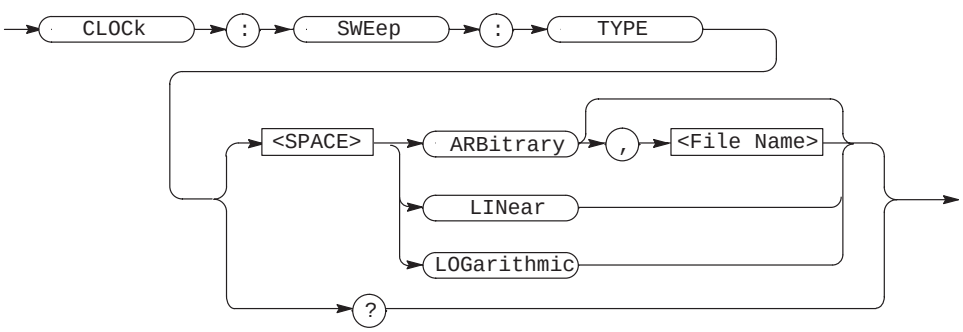
This command is effective only when the Option 05 (clock sweep) is installed.

Group SETUP

Related Commands CLOCk:SWEep:DWELl, CLOCk:SWEep:FREQuency:STARt, CLOCk:SWEep:FREQuency:STOP, CLOCk:SWEep:TIME

Syntax

```
CLOCk:SWEep:TYPE {ARBitrary[,<File Name>] | LINear | LOGarithmic}
CLOCk:SWEep:TYPE?
```



Arguments

<File>::= <string> Clock sweep file (.CLK)

ARBitrary
sweeps with the frequency varying according to the contents of the clock sweep file. If the clock sweep file is not specified, the previously specified file is used.

LINear
sweeps with the frequency varying linearly.

LOGarithmic
sweeps with the frequency varying logarithmically.

Examples

```
:CLOCK:SWEEP:TYPE ARBITRARY,"CLKSWEEP.CLK"
```

sets the clock sweep type to ARBitrary.

*CLS

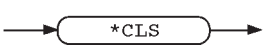
The *CLS common command clears SESR (Standard Event Status Register), the SBR (Status Byte Register) and the Event Queue, which are used in the waveform generator status and event reporting system. For more details, refer to Section 3 *Status and Events*.

Group STATUS and EVENT

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

Syntax

```
*CLS
```



Examples

```
*CLS
```

clears the SESR, the SBR, and the Event Queue.

CONFigure (?) (AWG2005)

The CONFigure command controls the I/O of control and clock signals when an AWG2005 is operated in parallel.

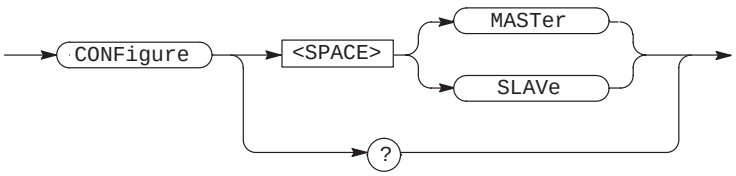
The CONFigure? query returns the operating mode when an AWG2005 is operated in parallel.

Group MODE

Related Commands MODE,CLOCk:SOURce

Syntax

```
CONFigure {MASTer | SLAVe}
CONFigure?
```



Arguments

MASTer
this waveform generator supplies control and clock signals to the slave AWG2005 operating in parallel.

SLAVe
this waveform generator receives control and clock signals from the master AWG2005 operating in parallel.

Examples

CONFIGURE?
might return :CONFIGURE MASTER

CURVe (?)

The CURVe command transmits unscaled, binary-formatted waveform data from an external controller to the location inside the waveform generator specified with the DATA:DESTination command.

The CURVe? query transmits unscaled data for a waveform in binary format to the external controller from the source located inside the waveform generator specified with the DATA:SOURce command.

The unscalled waveform data can be converted to the waveform data of an absolute scall using preamble information.

Group

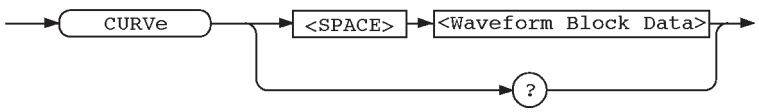
WAVEFORM

Related Commands

WAVFrm?, WFMPre?, DATA:SOURce, DATA:DESTination, DATA:ENCDG, DATA:WIDTH

Syntax

CURVe <Block Data>
CURVe?



Arguments `<Block Data>::=<Arbitrary Data>`
where `<arbitrary data>` is the unscaled waveform data in binary format.

Examples `:CURVE #3256...`
transmits an unscaled waveform to the waveform generator. The block data element #3256 indicates that 256 bytes of binary data are to be transmitted.

DATA (?)

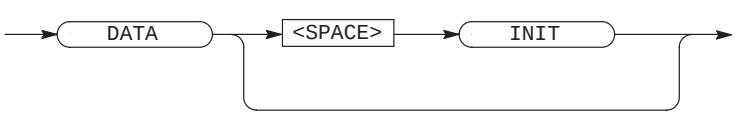
The DATA command restores all currently specified settings related to waveform or marker transfer to their default values.

The DATA? query returns all settings related to the data command currently in effect for waveform or marker transfer.

Group WAVEFORM

Related Commands DATA:DESTination, DATA:ENCDG, DATA:SOURce, DATA:WIDTh

Syntax
```
DATA INIT
DATA?
```



Arguments INIT
restores all currently specified settings related to waveform or marker transfer to their default values.

Examples DATA?
might return :DATA:DESTINATION "GPIB.WFM";ENCDG RPBINARY;SOURCE"CH1";WIDTH

DATA:DESTination (?)

The DATA:DESTination command specifies the destination inside the waveform generator to which the waveform or the marker data is transmitted and stored using CURVe:DATA or MARKer:DATA command.

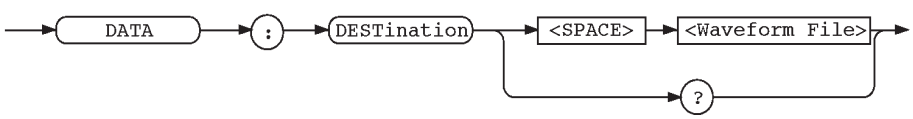
The DATA:DESTination? query returns the destination currently specified.

Group WAVEFORM

Related Commands CURVe, MARKER<x>:AOFF, MARKER<x>:POINt, MARKer:DATA

Syntax

```
DATA:DESTination <Waveform File>
DATA:DESTination?
```



Arguments

<Waveform File>::=<string>
where <string> must be the name of a waveform file to be transferred into the internal memory of the waveform generator. If the waveform file name specified already exists in internal memory, the file is overwritten. Also, if the overwritten file contains a waveform currently loaded and output on a channel, transmitting the new waveform replaces the current waveform at the channel output as well as in the file.

Examples

```
:DATA:DESTINATION "WAVE_EXT.WFM"
```
specifies the waveform file: WAVE_EXT.WFM as a destination.

DATA:ENCDG (?)

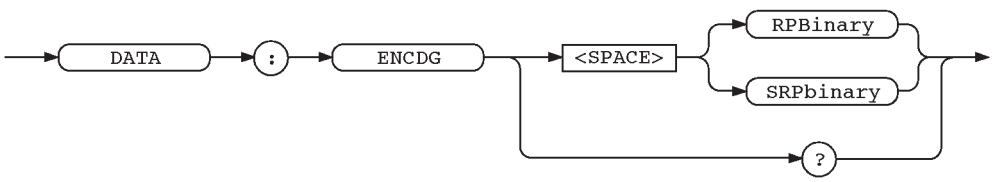
The DATA:ENCDG command sets the encoding format for the waveform transferred using the CURVe command or WAVFrm command when the data width is 2 bytes.

The DATA:ENCDG? query returns the waveform encoding format currently set.

Group WAVEFORM

Related Commands CURVe, WAVFrm?, WFMPre:ENCDG, WFMPre:BYT_OR, WFMPre:BIT_NR, DATA:WIDTH

Syntax
```
DATA:ENCDG {RPBinary | SRPbinary}
DATA:ENCDG?
```



Arguments

RPBinary
specifies positive integer data point representation with the most significant byte transferred first.

SRPbinary
specifies positive integer data point representation with the least significant byte transferred first.

The data transfer time byte order can also be specified using the WFMPre:BYT_OR command. When both this command and the WFMPre:BYT_OR command are used, the most recently issued, i.e., the last, command takes effect. For example, if the byte order is set to high order byte first using this command (DATA:ENCDG RPBinary), and then a WFMPre:BYT_OR LSB command is executed, the setting will be changed so that the low order byte is transmitted first.

Examples
```
:DATA:ENCDG RPBINARY
```
specifies the format RPBinary, which is described under *Arguments* above.

DATA:SOURce (?)

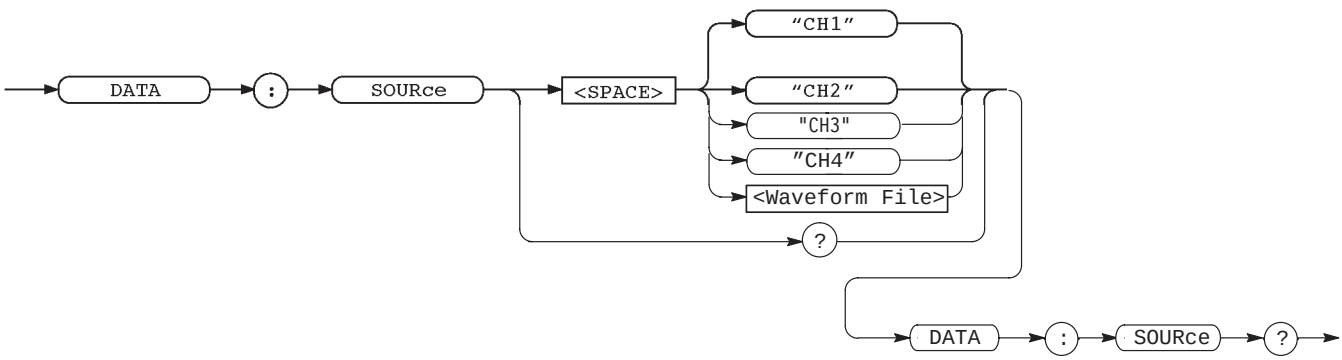
The DATA:SOURce command specifies the waveform generator source (channel or waveform file) from which the waveform is transmitted to an external controller using the CURVe? query.

The DATA:SOURce? query returns the source that is currently specified.

Group WAVEFORM

Related Commands CURVe?

Syntax DATA:SOURce {"CH1" | "CH2"(AWG2005/20/21) | "CH3"(AWG2005) | "CH4"(AWG2005) | <Waveform File>}DATA:SOURce?



Arguments <Waveform File>::=<string>
where the string is "CH1", "CH2", "CH3", or "CH4" for asociated channels respectively, or is the name of a waveform file located in internal memory. No other source strings are allowed.

Examples :DATA:SOURCE "CH1"
specifies channel 1 as a source.

DATA:WIDTh (?)

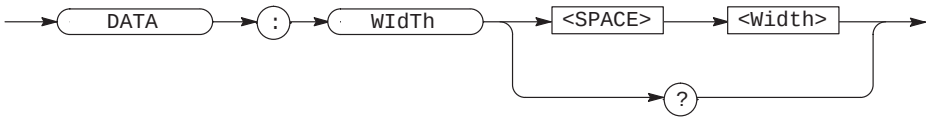
The DATA:WIDTh command sets the number of bytes per data point during waveform data transfer.

The DATA:WIDTh? query returns the number of bytes per data point during waveform data transfer.

Group WAVEFORM

Related Commands DATA:ENCDG, WFMPre:BIT_NR, WFMPre:BYT_NR, WFMPre:BIT_OR

Syntax DATA:WIDTh <Width>
DATA:WIDTh?



Arguments

<Width>::=<NR1>
where <NR1> is a decimal number with a value of either 1 or 2.
The data width during data transfers can also be set by the WFMPre:BYT_NR command. When both this command and the WFMPre:BYT_NR command are issued, the most recently issued, i.e., the last, command takes effect. For example, if the data width is set to 1 using this command (DATA:WIDTH 1), and then a WFMPre:BYT_NR 2 command is executed, the data width will be two bytes.

Examples

:DATA:WIDTH 1
sets the number of bytes per data point during waveform data transfers to be one byte.

DATE (?)

The DATE command sets the date for the waveform generator operating system. The DATE? query returns the date currently set.

Group

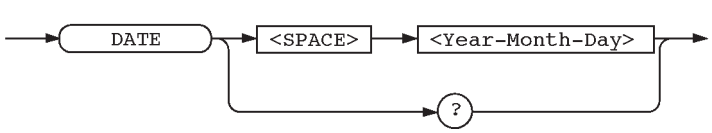
SYSTEM

Related Commands

TIME

Syntax

DATE <Year-Month-Day>
DATE?



Arguments

<Year-Month-Day>::=<string>
where the string must be in the format"YYYY-MM-DD"and the string elements are:

YYYY	the year expressed in 4-digits
MM	the month (1 to 12)
DD	the day (01 to 31 or to the last DD available for the month)

Examples `:DATE "1993-11-11"`
sets the date.

DEBug?

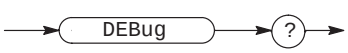
The `DEBug?` query returns all current settings for the remote command debugging function.

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

Group SYSTEM

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

Syntax `DEBug?`



Arguments None

Responses See Examples

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

DEBug:SNOop?

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

This query is equivalent to the `DEBug?` query.

Group SYSTEM

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

Syntax `DEBug:SNOop?`

Arguments None

Responses See Examples

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

DEBug:SNOop:DELAy?

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

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

Group SYSTEM

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

Syntax DEBug:SNOop:DELAy?

Arguments None

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

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

DEBug:SNOop:DELAy:TIME (?)

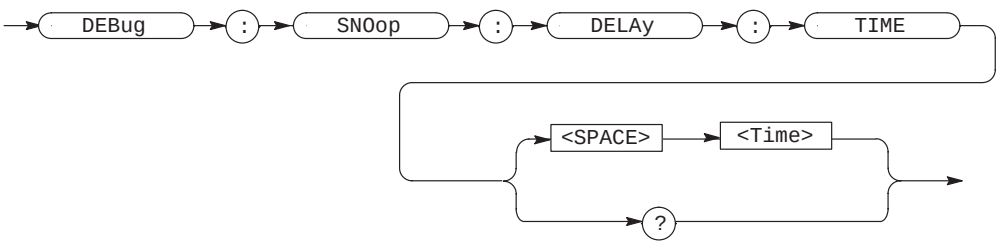
The DEBug:SNOop:DELAy:TIME command sets the display time for commands in a sequence of commands that are connected by semicolons.

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

Group SYSTEM

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

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



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

Examples :DEBUG:SNOOP:DELAY:TIME 0.5
sets the command display time to 0.5 seconds.

DEBug:SNOop:STATe (?)

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

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

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

The display format is as follows.

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

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

Message termination — "<PMT>"

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

Block data — "#0"

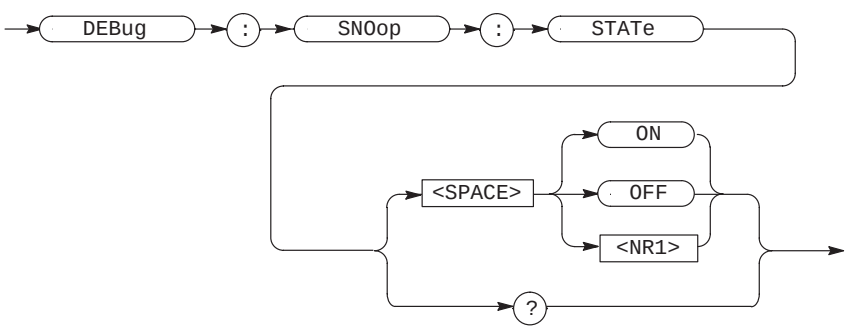
Any data other than one of the above — "<code decimal display>", e.g. a code value of 80 (hexadecimal) would be displayed as <128>.

Group SYSTEM

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

Syntax

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



Arguments

ON or nonzero value
enables the debugging function.

OFF or zero value
clears the debugging function.

Responses

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

Examples

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

DESE (?)

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

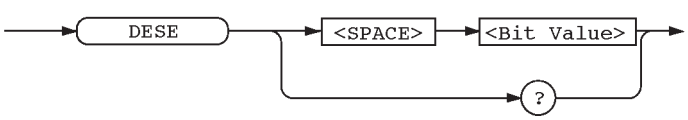
The power-on default for the DESER is to set all bits to 1 if the power-on status flag is TRUE. If this flag is set to FALSE, the DESER maintains its current value through a power cycle.

Group STATUS and EVENT

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

Syntax

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



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

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

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

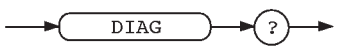
DIAG?

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

Group CALIBRATION and DIAGNOSTIC

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

Syntax DIAG?



Arguments None

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

ALL	all routines
CPU	CPU unit check routine
LOCk	clock unit check routine
DISPlay	display unit check routine
FPP	floating point processor unit check routine
FPANel	front panel control unit check routine
SETUp	setup related unit check routine
TRIGger	TRIGGER unit test routine (AWG2005/40/41)
WMEMory	waveform memory check routine.

and where <Result>::=<NR1> is one of following responses in AWG2005/40/41 instruments:

0	terminated without error
100	detected an error in the CPU unit
200	detected an error in the clock unit
300	detected an error in the display unit
400	detected an error in the floating point processor unit
500	detected an error in the front panel unit
600	detected an error in the setup-related unit
700	detected an error in the waveform memory

***NOTE**. The AWG2000 Series Arbitrary Waveform Generators do not respond to any commands or queries issued during Self Test.*

Examples DIAG?
might return :DIAG:SELECT ALL;RESULT 0.

DIAG:RESUlt?

The DIAG:RESUlt? query returns results of self-test execution.

Group CALIBRATION and DIAGNOSTIC

Related Commands DIAG:SELect, DIAG:STATe

Syntax DIAG:RESUlt?

Arguments None

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

0	terminated without error
100	detected an error in the cpu unit
200	detected an error in the clock unit
300	detected an error in the display unit
400	detected an error in the floating point processor unit
500	detected an error in the front panel unit
600	detected an error in the setup-related unit
700	detected an error in the waveform memory
800	detected an error in the trigger unit

Examples DIAG:RESULT?
might return :DIAG:RESULT 200

DIAG:SELect (?)

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

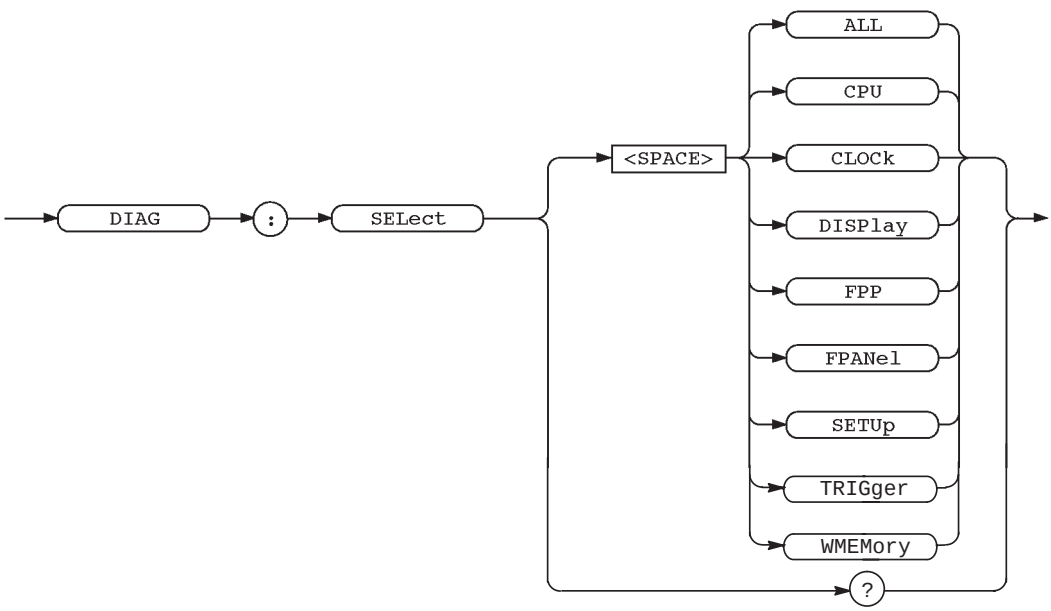
Group CALIBRATION and DIAGNOSTIC

Related Commands DIAG:STATe, DIAG:RESULt?

Syntax

```
DIAG:SELect { ALL | CPU | CLOCk | DISPlay | FFP | FPANel | SETup
| TRIGger (AWG2005/40/41) | WMEMory }

DIAG:SELect?
```



Arguments

ALL	checks all routines that follow
CPU	checks the CPU unit
CLOCk	checks the clock unit
DISPlay	checks the display unit
FPP	checks the floating point processor unit
FPANel	checks the front panel control unit
SETUp	checks the unit for setup
TRIGger	TRIGGER unit test routine (AWG2005/40/41)
WMEMory	checks the waveform memory

Examples

```
:DIAG:SELECT CPU ; STATE EXECUTE
```

executes the CPU self-test routine.

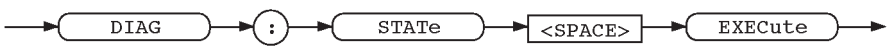
DIAG:STATe

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

Group CALIBRATION and DIAGNOSTIC

Related Commands DIAG:SELect, DIAG:RESUlt?

Syntax DIAG:STATe EXECute



Arguments EXECute
Performs the self-test using the selected routine.

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

DISK?

The DISK? query returns all settings currently set for floppy disk operation.

Group MEMORY

Related Commands DISK:CDIRectory?, DISK:DIRectory?, DISK:MDIRectory

Syntax DISK?

Arguments None

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

Examples :DISK?
might return :DISK:FORMAT:TYPE HD3

DISK:CDIRectory

The DISK:CDIRectory command changes current working directory.

Group MEMORY

Related Commands DISK:DIRectory?, DISK:MDIRectory

Syntax DISK:CDIRectory <Directory Path>

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

Examples :DISK:CDIRECTORY "\FG\WORK3"
changes the current working directory to \FG\WORK3.

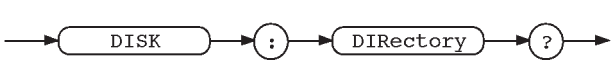
DISK:DIRectory?

The DISK:DIRectory? query returns current working directory path.

Group MEMORY

Related Commands DISK:CDIRectory, DISK:MDIRectory

Syntax DISK:DIRectory?



Arguments None

Examples :DISK:DIRECTORY?
might return :DISK:CDIRECTORY "\FG\WORK3"

DISK:FORMat?

The DISK:FORMat? query returns currently selected format type for formatting new floppy disks.

Group MEMORY

Related Commands DISK:FORMat:TYPE, DISK:FORMat:STATe

Syntax DISK:FORMat?

Arguments None

Responses Following format types are returned:

DD1 2DD, 720 KB, for IBM PC and TOSHIBA J3100
DD2 2DD, 640 KB, for NEC PC–9800
HD1 2HD, 1.232 MB, for NEC PC–9800
HD2 2HD, 1.200 MB, for TOSHIBA J3100
HD3 2HD, 1.440 MB, IBM PC

For details on each of these formats, refer to DISK:FORMat:TYPE command (page 2-76).

Examples :FORMAT:TYPE?
might return :DISK:FORMAT:TYPE HD3

DISK:FORMat:STATe

The DISK:FORMat:STATe command formats a floppy disk in the waveform generator disk drive, using the format type selected with the DISK:FORMat:TYPE command.

Group MEMORY

Related Commands DISK:FORMat:TYPE

Syntax DISK:FORMat:STATe EXECute

Arguments EXECute
initiates a floppy disk format.

Examples :DISK:FORMAT:TYPE DD1 ;STATE EXECUTE
formats a floppy disk for IBM PC 2DD.

DISK:FORMat:TYPE (?)

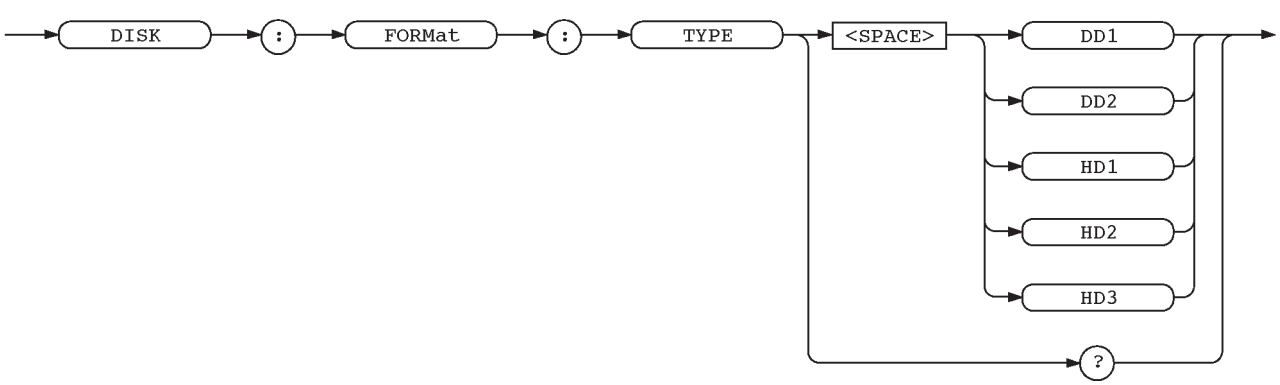
The DISK:FORMat:TYPE command selects the format type the waveform generator uses when formatting floppy disk in its disk drive. (Use the DISK:FORMat:STATe command to format a disk.)

The DISK:FORMat:TYPE? query returns currently selected format type.

Group MEMORY

Related Commands DISK:FORMat:STATe

Syntax DISK:FORMat:TYPE {DD1 | DD2 | HD1 | HD2 | HD3}
DISK:FORMat:TYPE?



Arguments You can select from the following formats:

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

Examples

```
:DISK:FORMAT:TYPE HD3 ;STATE EXECUTE
```

formats a floppy disk for IBM PC 2HD.

DISK:MDIRectory

The DISK:MDIRectory command creates a new directory.

Group MEMORY

Related Commands DISK:CDIRectory, DISK:DIRectory?

Syntax

```
DISK:MDIRectory <Directory Path>
```

Arguments

<Directory Path>::=<string>
where <string> is the complete path of the new directory.

Examples

```
:DISK:MDIRECTORY "WORK4"
```

creates the new directory WORK4 in current working directory.

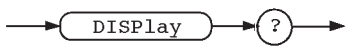
DISPlay?

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

Group DISPLAY

Related Commands None

Syntax DISPlay?



Arguments None

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

Examples DISPLAY?
might return :DISPLAY:BRIGHTNESS 75;CATALOG:ORDER NAME1;DISPLAY:CLOCK 0;MENU:SETUP:FORMAT GRAPHICS;DISPLAY:MESSAGE:SHOW" "

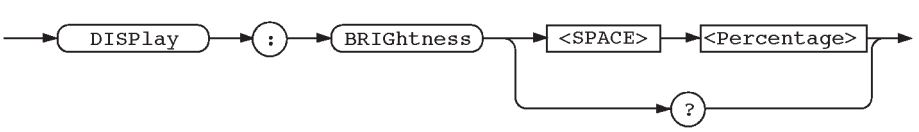
DISPlay:BRIGhtness (?)

The DISPlay:BRIGhtness command adjusts the brightness of the screen as a percentage of full intensity; the DISPlay:BRIGhtness? query returns the current brightness setting as a percentage of full intensity.

Group DISPLAY

Related Commands DISPlay?

Syntax DISPlay:BRIGhtness <Percentage>
DISPlay:BRIGhtness?



Arguments <Percentage>::=<NR1>[<unit>]
where <NR1> is a integer ranging from 0 to 100%, in 1% steps 1%, and <unit> is PCT for percent.

Examples :DISPLAY:BRIGHTNESS 80
sets screen brightness to 80% of maximum intensity.

DISPlay:CATalog?

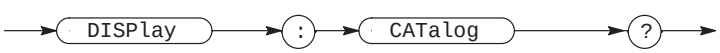
The DISPlay:CATalog? query returns the catalog display sorting conditions.

This query is equivalent to the DISPlay:CATalog:ORDer? query.

Group DISPLAY

Related Commands DISPlay:CATalog:ORDer

Syntax DISPlay:CATalog?



Arguments None

Responses [:DISPLAY:CATALOG:ORDER]<Catalog order>
where <catalog order> is one of following arguments:

NAME1
orders the display according to the ASCII collating sequence of the file names (Name).

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

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

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

TYPE1
orders the display according to the ASCII collating sequence of the file extensions (Type).

TYPE2
orders the display according to the ASCII collating sequence of the file extensions (Type) and also according to the ASCII collating sequence of the file names (Name).

TYPE3
orders the display according to the ASCII collating sequence of the file extensions (Type) and also with more recent (Date and Time) files first.

TYPE4
orders the display according to the ASCII collating sequence of the file extensions (Type) and also with older (Date and Time) files first.

Examples

DISPLAY:CATALOG?
might return :DISPLAY:CATALOG:ORDER NAME1

DISPlay:CATalog:ORDer (?)

The DISPlay:CATalog:ORDer command sets the catalog display sorting conditions.

The DISPlay:CATalog:ORDer? query returns the currently specified catalog display sorting conditions.

Group

DISPLAY

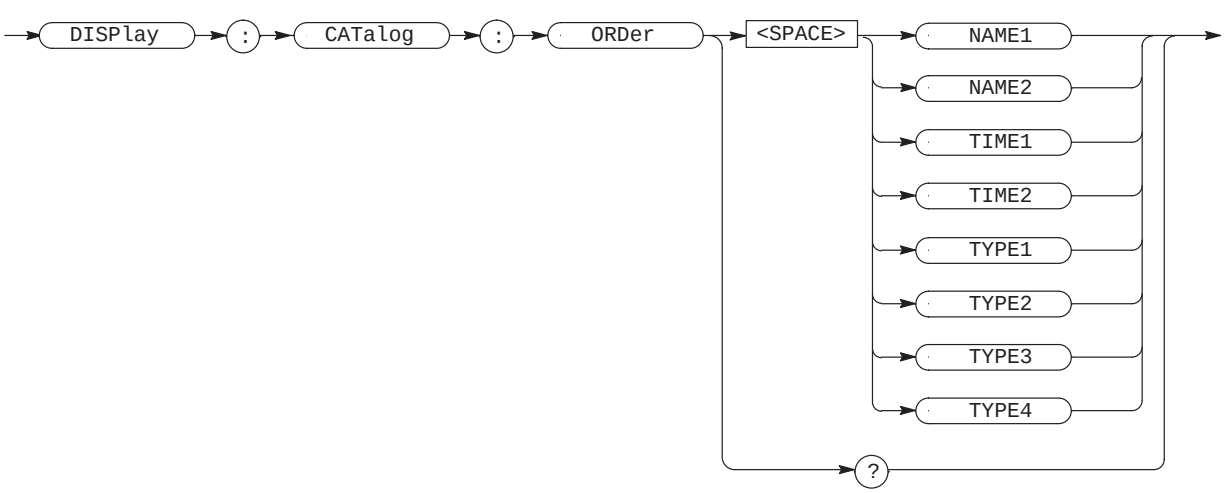
Related Commands

DISPlay:CATalog?

Syntax

DISPlay:CATalog:ORDer
{NAME1 | NAME2 | TIME1 | TIME2 | TYPE1 |TYPE2 | TYPE3 | TYPE4}

DISPlay:CATalog:ORDer?



Arguments

NAME1
orders the display according to the ASCII collating sequence of the file names (Name).

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

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

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

TYPE1
orders the display according to the ASCII collating sequence of the file extensions (Type).

TYPE2
orders the display according to the ASCII collating sequence of the file extensions (Type) and also according to the ASCII collating sequence of the file names (Name).

TYPE3
orders the display according to the ASCII collating sequence of the file extensions (Type) and also with more recent (Date and Time) files first.

TYPE4
orders the display according to the ASCII collating sequence of the file extensions (Type) and also with older (Date and Time) files first.

Examples

```
:DISPLAY:CATALOG:ORDER TIME1
```

sets the catalog display to be in the order of more recent (Date and Time) files displayed first.

DISPlay:CLOCk (?)

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

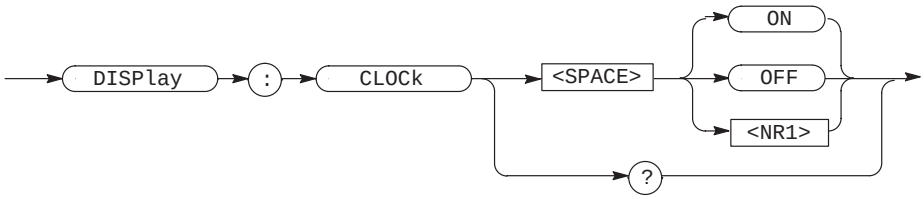
The DISPlay:CLOCk? query returns whether or not the data and time are displayed.

Group DISPLAY

Related Commands None

Syntax

```
DISPlay:CLOCk {ON | OFF | <NR1>}
DISPlay:CLOCk?
```



Arguments

`ON or nonzero value`
sets the waveform generator to display the date and time.

`OFF or zero value`
sets the waveform generator to not display the date and time.

Responses

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

Examples

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

DISPlay:MENU?

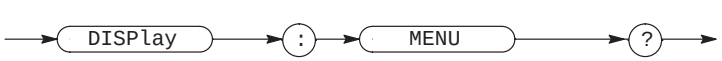
The `DISPlay:MENU?` query returns the SETUP menu display format.

This query is equivalent to the DISPlay:MENU:SETUp:FORMat? query.

Group DISPLAY

Related Commands `DISPlay:MENU:SETUp:FORMat`

Syntax `DISPlay:MENU?`



Arguments None

Responses

`[:DISPLAY:MENU:SETUP:FORMAT]<Menu format>`
where `<Menu format>` is one of the following:

`GRAPHICS` Graphics display mode is used for the SETUP menu.
`TEXT` Text display mode is used for the SETUP menu.

Examples DISPLAY:MENU?
would return :DISPLAY:MENU:SETUP:FORMAT TEXT if text display mode was used for the SETUP menu.

DISPlay:MENU:SETUp?

The DISPlay:MENU:SETUp? query returns the SETUP menu display format.

This query is equivalent to the DISPlay:MENU:SETUp:FORMat? query.

Group DISPLAY

Related Commands DISPlay:MENU:SETUp:FORMat

Syntax DISPlay:MENU:SETUp?

Arguments None

Responses [:DISPLAY:MENU:SETUP:FORMAT]<Menu format>
where <Menu format> is one of the following:

GRAPHICS	Graphics display mode is used for the SETUP menu.
TEXT	Text display mode is used for the SETUP menu.

Examples DISPLAY:MENU:SETUP?
would return :DISPLAY:MENU:SETUP:FORMAT GRAPHICS if graphics display mode was used for the SETUP menu.

DISPlay:MENU:SETUp:FORMat (?)

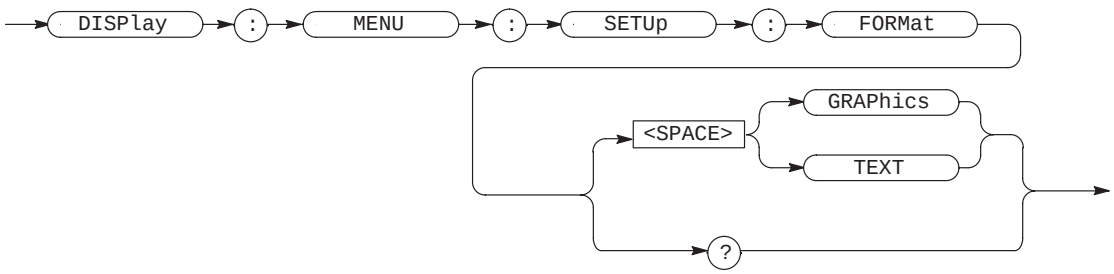
The DISPlay:MENU:SETUp:FORMat command sets the SETUP menu display format.

The DISPlay:MENU:SETUp:FORMat? query returns the SETUP menu display format.

Group DISPLAY

Related Commands DISPlay:MENU?, DISPlay:MENU:SETUp?

Syntax
```
DISPlay:MENU:SETUp:FORMat {GRAPhics | TEXT}
DISPlay:MENU:SETUp:FORMat?
```



Arguments

GRAPhics	Graphics display mode is used for the SETUP menu.
TEXT	Text display mode is used for the SETUP menu.

Examples :DISPLAY:MENU:SETUP:FORMAT TEXT
sets the SETUP menu display format to text display mode.

DISPlay:MESSage (?)

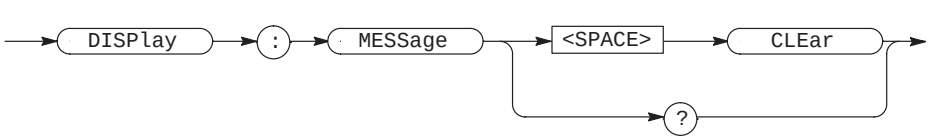
The DISPlay:MESSage command clears (erases) the message displayed in the message area.

The DISPlay:MESSage? query returns the message displayed in the message area.

Group DISPLAY

Related Commands DISPlay:MESSage:SHOW

Syntax
```
DISPlay:MESSage CLEar
DISPlay:MESSage?
```



Arguments CLEar
clears (erases) the message displayed in the message area.

Examples :DISPLAY:MESSAGE CLEAR
clears (erases) the message displayed in the message area.

DISPlay:MESSage:SHOW (?)

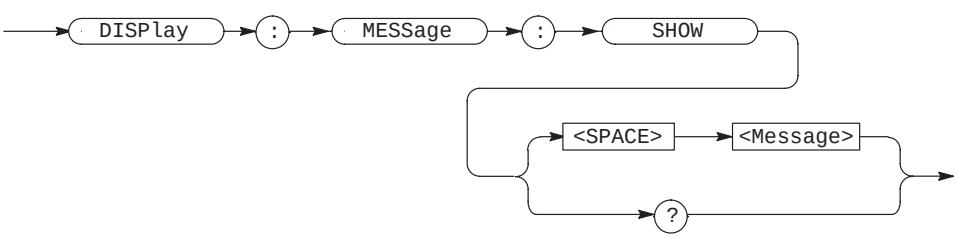
The DISPlay:MESSage:SHOW command displays the message displayed in the message area.

The DISPlay:MESSage:SHOW? query returns the message displayed in the message area.

Group DISPLAY

Related Commands DISPlay:MESSage

Syntax
```
DISPlay:MESSage:SHOW <Message>
DISPlay:MESSage:SHOW?
```



Arguments <Message>::=<string>
where <string> is a message of up to 60 characters.

Examples :DISPLAY:MESSAGE:SHOW "TEST No.1"
displays the message "TEST No.1" in the message area.

EQUAtion:COMPile (?)

The EQUAtion:COMPile command compiles the specified equation file into a waveform file.

The EQUAtion:COMPile? determines whether or not an equation file compilation is in progress.

This command is equivalent to the EQUAtion:COMPile:STATe command.

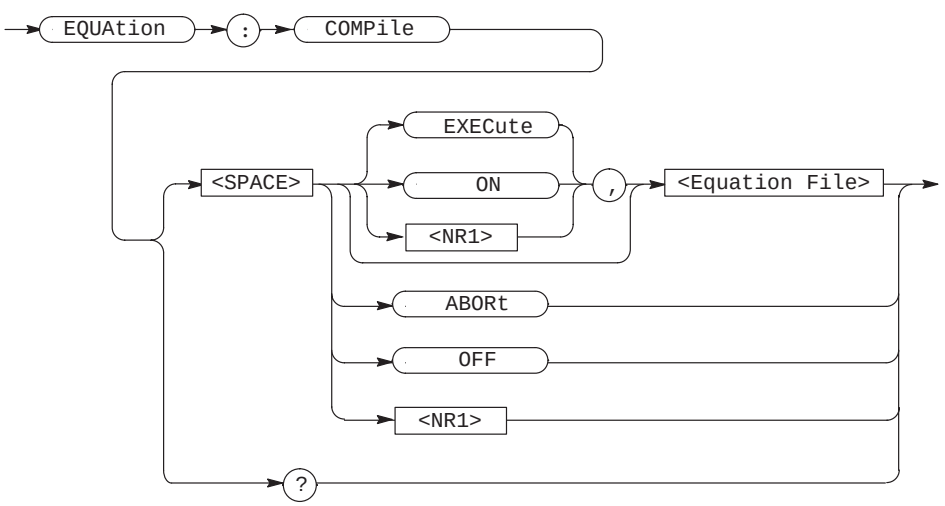
Group WAVEFORM

Related Commands EQUAtion:COMPile:STATe

Syntax

```
EQUAtion:COMPile {[EXECute, | ON, | <NR1>,]<Equation File>| ABORt
| OFF | <NR1>}

EQUAtion:COMPile?
```



Arguments

<Equation File>::=<string>
<Equation File> must be an internal memory equation file. The waveform data that is created as a result of the compilation is stored in a waveform file. The base name of the waveform file is the same as the base name of the equation file.

EXECute
compiles the specified equation file.

ON or nonzero value
compiles the specified equation file.

ABORt
forcibly terminates the currently executing compilation.

OFF or zero value
forcibly terminates the currently executing compilation.

Responses

[:EQUATION:COMPILE] 1,<Equation File>
Compilation in progress

[:EQUATION:COMPILE] 0
No compilation in progress

Examples

:EQUATION:COMPILE ON, "EXP_SAMP.EQU"
compiles the equation file “EXP_SAMP.EQU” and stores the generated waveform data in the file “EXP_SAMP.WFM”.

EQUAtion:COMPile:STATe (?)

The EQUAtion:COMPile:STATe command compiles the specified equation file into a waveform file.

The EQUAtion:COMPile:STATe? determines whether or not an equation file compilation is in progress.

Group

WAVEFORM

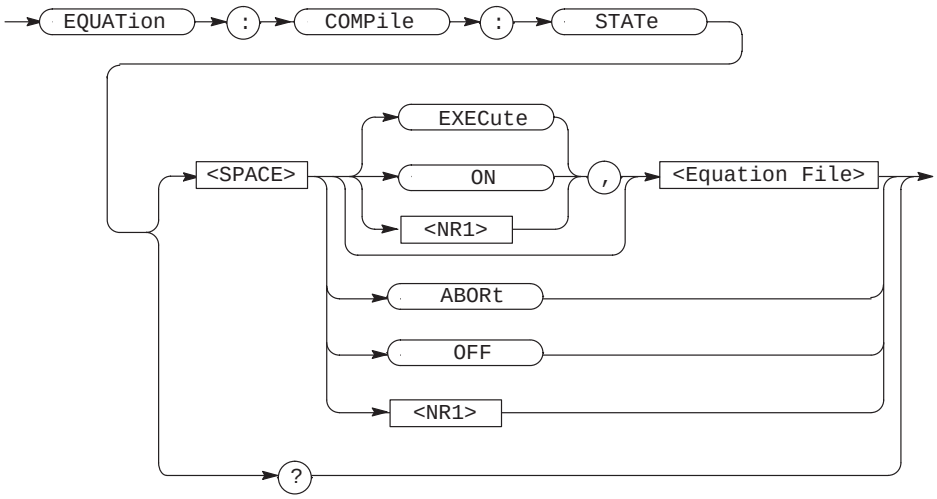
Related Commands

EQUAtion:COMPile

Syntax

EQUAtion:COMPile:STATe {[EXECute, | ON, | <NR1>,] <Equation File> | ABORt | OFF | <NR1>}

EQUAtion:COMPile:STATe?



Arguments

<Equation File>::=<string>
<Equation File> must be an internal memory equation file. The waveform data that is created as a result of the compilation is stored in a waveform file. The base name of the waveform file is the same as the base name of the equation file.

EXECute
compiles the specified equation file.

ON or nonzero value
compiles the specified equation file.

ABORt
forcibly terminates the currently executing compilation.

OFF or zero value
forcibly terminates the currently executing compilation.

Responses

[:EQUATION:COMPILE:STATE]1,<Equation File>
Compilation in progress

[:EQUATION:COMPILE:STATE]0
No compilation in progress

Examples

:EQUATION:COMPILE:STATE EXECUTE,"EXP_SAMP.EQU"
compiles the equation file “EXP_SAMP.EQU” and stores the generated waveform data in the file “EXP_SAMP.WFM”.

EQUAtion:DEFine(?)

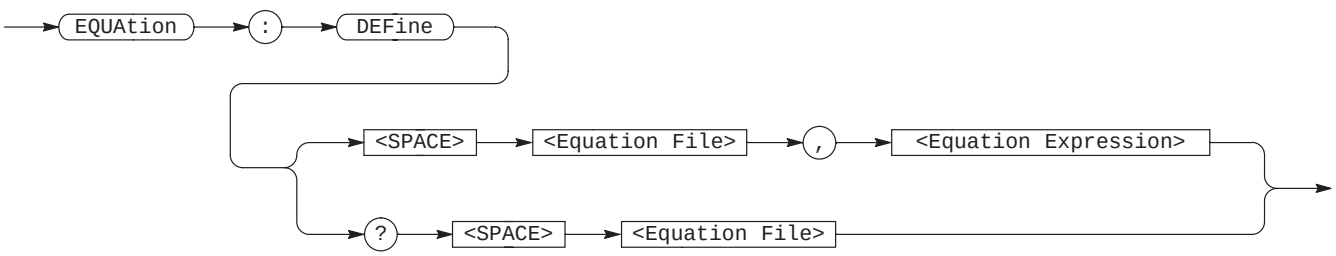
The EQUAtion:DEFine command writes an equation expression into the specified equation file. The EQUAtion:DEFine? query returns the equation expression that is stored in the specified equation file.

Group WAVEFORM

Related Commands EQUAtion:COMPile:STATe, EQUAtion:WPOints

Syntax

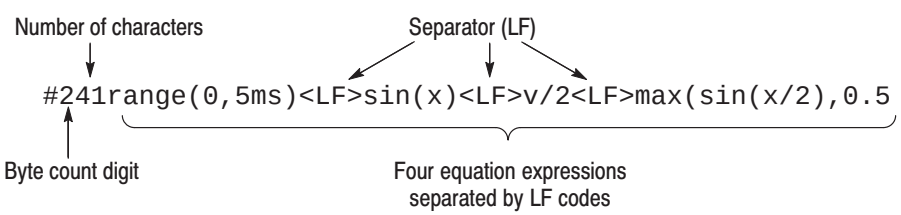
```
EQUAtion:DEFine <Equation File>, <Equation Expression>
EQUAtion:DEFine? <Equation File>
```



Arguments

<Equation File>::=<string>
where <string> must be the name of an equation file to be stored in internal memory.

<Equation Expression>::=<Arbitrary Data>
where the <Arbitrary Data> for the equation expression must be written in ASCII code with each expression separated by a Line Feed (LF) code as follows.



Equation file can be compiled to waveform file using EQUAtion:COMPile:STATe. EQUAtion:WPOints command sets the number of waveform points (use after compile) for equation file.

Examples

```
:EQUATION:DEFINE "EXP_SAMP.EQU", #241range(0.5ms)
<LF>sin(x)<LF>v/2<LF>max(sin ...
```

writes an equation expression into the equation file EXP_SAMP.EQU.

EQUAtion:WPOints (?)

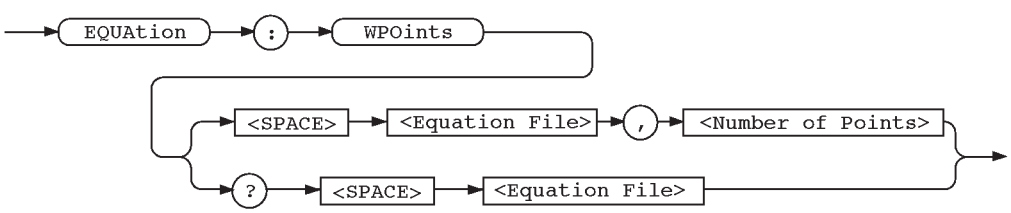
The EQUAtion:WPOints command specifies the number of waveform points, from the equation file, to be written to the waveform file when an equation file is compiled. The EQUAtion:WPOints? query returns the number of waveform points set to be written to the equation file.

Group WAVEFORM

Related Commands EQUAtion:COMPile:STATe, EQUAtion:DEFine

Syntax

```
EQUAtion:WPOint <Equation File>, <Number of Points>
EQUAtion:WPOints? <Equation File>
```



Arguments

<Equation File>::=<string>
where <string> must be the name of an equation file in internal memory. Equation file can be compiled to waveform file using EQUAtion:COMPile:STATe.

<Number of Points>::=<NR1>
where <NR1> must be in the range of 1 to 32768 (32 K)

Examples

:EQUATION:WPOINTS "EXP_SAMP.EQU", 1000
specifies 1000 as a number of waveform points to be written to the file EXP_SAMP.EQU.

*ESE (?)

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

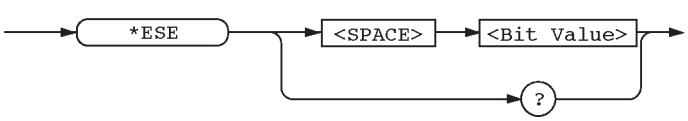
If the power on status flag is TRUE, the power-on default for the ESER is to reset all bits to zero. If this flag is set to FALSE, the ESER bits do not change value during the power-on cycle.

Group STATUS and EVENT

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

Syntax

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



Arguments

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

Examples

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

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

*ESR?

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

Group STATUS and EVENT

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

Syntax *ESR?

Arguments None

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

EVENT?

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

Group STATUS and EVENT

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

Syntax EVENT?

Arguments None

Examples EVENT?
might return :EVENT 113

EVMsg?

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

Group STATUS and EVENT

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

Syntax EVMsg?

Arguments None

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

EVQty?

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

Group STATUS and EVENT

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

Syntax EVQty?

Arguments None

Examples

```
:EVQty?
```
might return `:EVQTY 5.`

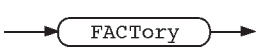
FACTory

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

Group SYSTEM

Related Commands `*RST`, `SECUre`

Syntax `FACTory`



Arguments None

Examples

```
:FACTORY
```
resets the waveform generator to its factory default settings.

FG?

The `FG?` query returns all settings currently set with the FG (Function Generator) commands.

Group FG

Related Commands None

Syntax `FG?`

Arguments None

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

Examples `:FG?`
might return the following response:
`:FG:STATE 0;FREQUENCY 2.500E+06;CH1:AMPLITUDE 1.000;OFFSET 0.000;POLARITY NORMAL;SHAPE SINUSOID;:FG:CH2:AMPLITUDE 1.000;OFFSET 0.000;POLARITY NORMAL;SHAPE SINUSOID`

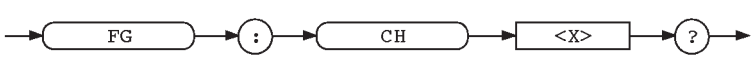
FG:CH<x>?

The `FG:CH<x>?` query returns all current settings of the function waveform parameters for the specified channel.

Group FG

Related Commands `FG:CH<x>:AMPLitude`, `FG:CH<x>:OFFSet`, `FG:CH<x>:POLarity`, `FG:CH<x>:SHAPe`

Syntax `FG:CH<x>?`



Arguments None

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

Examples `:FG:CH1?`
might return `:FG:CH1:AMPLITUDE 1.000;OFFSET 0.000;POLARITY NORMAL;SHAPE SINUSOID`

FG:CH<x>:AMPLitude (?)

The FG:CH<x>:AMPLitude command adjusts peak-to-peak voltage of the function waveform on the selected channel. The FG:CH<x>:AMPLitude? query returns peak-to-peak voltage currently set.

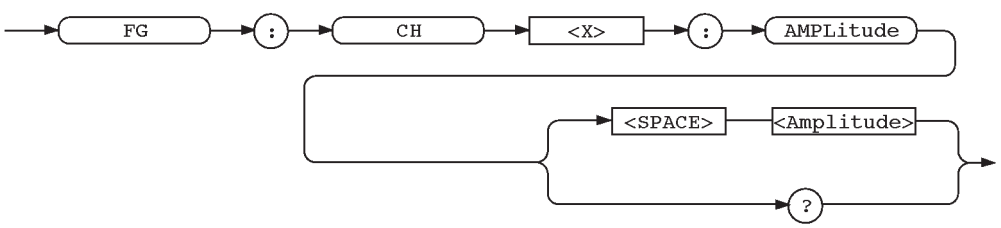
Group FG

Related Commands FG:CH<x>:OFFSet, FG:CH<x>:POLarity, FG:CH<x>:SHAPe, FG:CH<x>?

Syntax

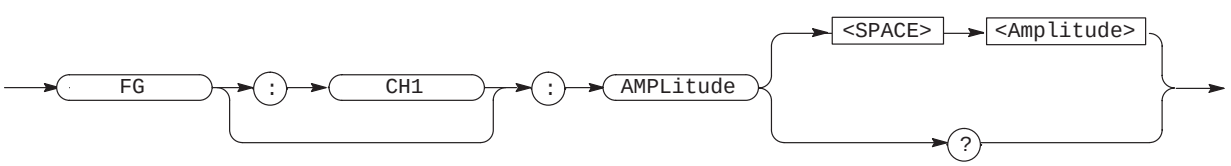
AWG2005/20/21

```
FG:CH<x>:AMPLitude <Amplitude>
FG:CH<x>:AMPLitude?
```



AWG2040/41

```
FG[:CH1]:AMPLitude <Amplitude>
FG[:CH1]:AMPLitude?
```



Arguments <Amplitude>::=<NR2>[<unit>]
where <NR2> is a decimal number to specify an amplitude that must range from 0.05 V to 10.000 V (AWG2005), 0.05 V to 5.000 V (AWG2020/21), 0.020 V to 2.000 V (AWG2040/41), in steps of 0.001 V, and optionally add <unit>::={V | mV}, for volts or millivolts.

Examples :FG:CH1:AMPL 100.0mV
sets peak-to-peak voltage to 100 mV.

FG:CH<x>:OFFSet (?)

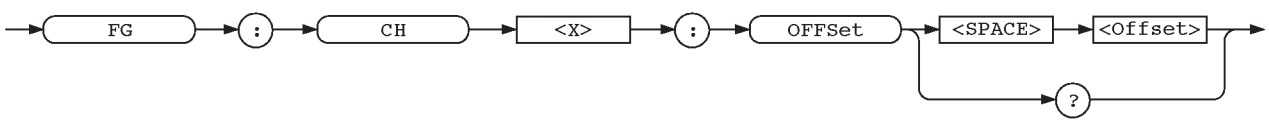
The FG:CH<x>:OFFSet command adjusts offset voltage of the function waveform on the selected channel. The FG:CH<x>:OFFSet? query returns offset voltage currently set.

Group FG

Related Commands FG:CH<x>:AMPLitude, FG:CH<x>:POLarity, FG:CH<x>:SHAPe, FG:CH<x>?

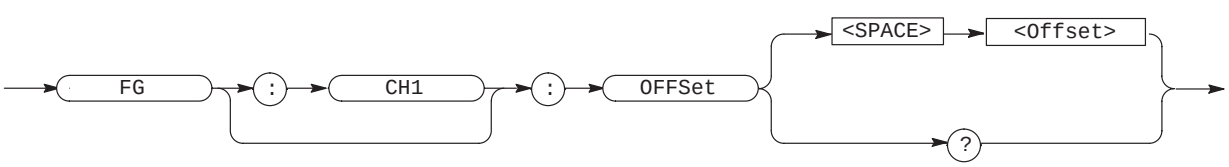
Syntax AWG2005/20/21

```
FG:CH<x>:OFFSet <Offset>
FG:CH<x>:OFFSet?
```



AWG2040/41

```
FG[:CH1]:OFFSet <Offset>
FG[:CH1]:OFFSet?
```



Arguments <Offset>::=<NR2>[<unit>]
where <NR2> is a decimal number that combines with [<unit>] to specify an offset that must range from –5.000 V to 5.000 V (AWG2005), –2.500 V to 2.500 V (AWG2020/21), in steps of 0.005 V, and –1.000 V to 1.000 V, in steps of 0.001 V (AWG2040/41), and <unit>::={V | mV}, for volts or millivolts.

Examples :FG:CH1:OFFS 50.0mV
sets offset voltage at channel 1 to 50.0 mV.

FG:CH<x>:POLarity (?)

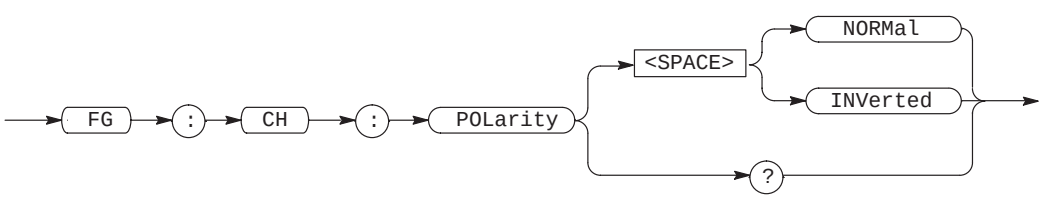
The FG:CH<x>:POLarity command sets polarity of the function waveform on the selected channel. The FG:CH<x>:POLarity? query returns polarity currently set.

Group FG

Related Commands FG:CH<x>:AMPLitude, FG:CH<x>:OFFSet, FG:CH<x>:SHAPe, FG:CH<x>?

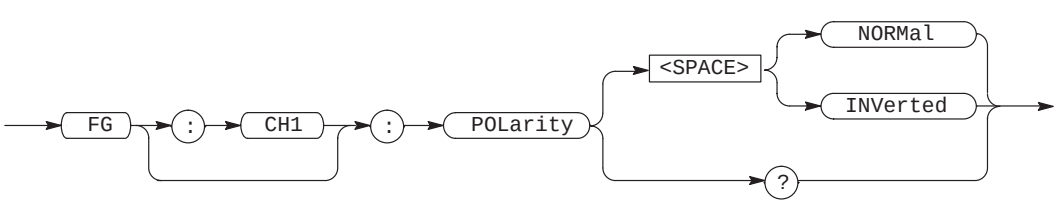
Syntax AWG2005/20/21

```
FG:CH<x>:POLarity {NORMal | INVerted}
FG:CH<x>:POLarity?
```



AWG2040/41

```
FG[:CH1]:POLarity {NORMal | INVerted}
FG[:CH1]:POLarity?
```



Arguments

NORMal
sets waveform to normal polarity.

INVerted
sets waveform to inverted polarity.

Examples

```
:FG:CH1:POLARITY INVERTED
```
inverts the waveform.

FG:CH<x>:SHAPe (?)

The FG:CH<x>:SHAPe command selects a standard function waveform (as opposed to a waveform file), and turns it on for display in the specified channel. The waveform generator displays the function waveform using its current parameters settings for the channel.

The FG:CH<x>:SHAPe? query returns the currently selected standard function waveform.

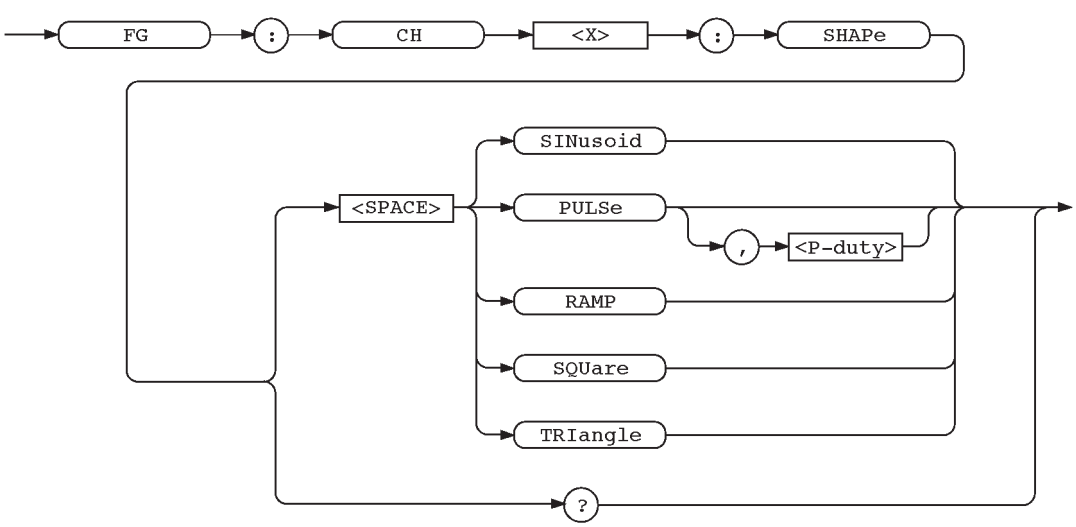
Group FG

Related Commands FG:CH<x>:AMPLitude, FG:CH<x>:POLarity, FG:CH<x>:OFFSet, FG:CH<x>?

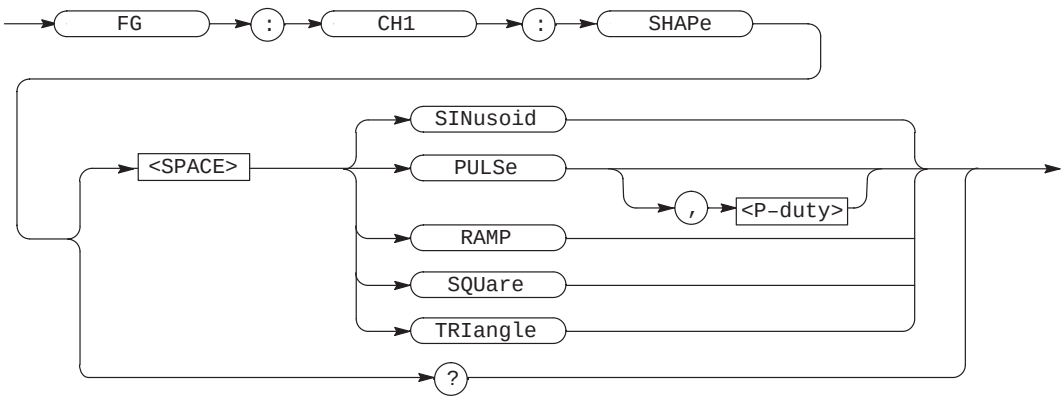
Syntax AWG2005/20/21

```
FG:CH<x>:SHAPe {SINusoid | PULSe[,<P-duty>] | RAMP | SQUare |
TRIangle}
```

```
FG:CH<x>:SHAPe?
```



AWG2040/41
FG[:CH1]:SHAPe
FG[:CH1]:SHAPe?



Arguments

SINusoid
selects a sine wave function waveform.

PULSe
selects a pulse function waveform, with its duty cycle defined as a percentage of the pulse function waveform period as follows:

<P-duty>::=<NR1>[<unit>]
where <NR1> has a range of 0 to 100, in steps of 1, and
<unit>::=PCT

RAMP
selects a ramp function waveform

SQUare
selects a square wave function waveform

TRIangle
selects a triangle function waveform

Examples

:FG:CH1:SHAPE PULSE, 40
selects pulse function waveform and sets duty cycle to 40%.

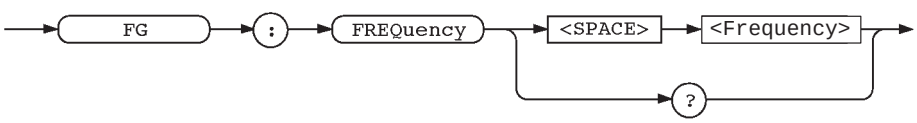
FG:FREQuency (?)

The FG:FREQuency command adjusts the frequency of the function waveform on selected channels. The FG:FREQuency? query returns the frequency currently set.

Group FG

Related Commands FG:STATe, FG?

Syntax
```
FG:FREQuency <Frequency>
FG:FREQuency?
```



Arguments <Frequency>::=<NR3>[<unit>]
where <NR3> is a decimal number to specify a frequency that must range from 1.000 Hz to 200.0 KHz (AWG2005), 1.000 Hz to 2.500 MHz (AWG2020/21), 1.000000 Hz to 10.00000 MHz (AWG2040/41), and optionally add <unit>::={HZ | KHZ | MHZ}, for hertz, kilohertz, and megahertz respectively.

Examples
```
:FG:FREQ 1.2MHZ
```
sets the waveform frequency to 1.2 MHz.

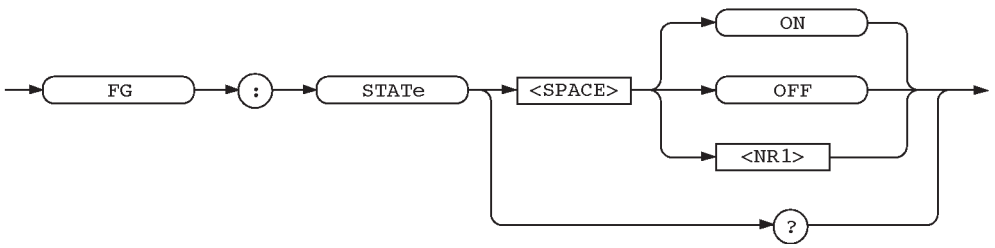
FG:STATe (?)

The FG:STATe command turns the FG (Function Generator) mode on or off. The FG:STATe? query returns status indicating whether the waveform generator is set to the function generator mode.

Group FG

Related Commands FG?

Syntax
```
FG:STATe {ON | OFF | <NR1>}
FG:STATe?
```



Arguments

`ON or nonzero value`
turns the FG mode on.

`OFF or zero value`
turns the FG mode off.

Responses

1 FG mode is currently turned on.
0 FG mode is currently turned off.

Examples

```
:FG:STATE 1
```
turns the FG mode on.

HCOPy (?)

The `HCOPy` command starts or terminates hard copy output from the specified output port.

The `HCOPy?` query returns all currently specified hard copy settings.

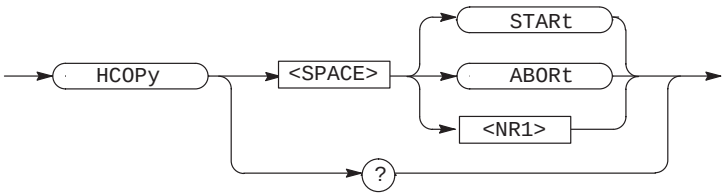
***NOTE**. This command is not compatible with the ANSI/IEEE Std 488.2–1987 standard.*

Group HARDCOPY

Related Commands `HCOPy:FORMat`,`HCOPy:PORT`,`HCOPy:DATA?`

Syntax

```
HCOPy {STARt | ABORt | <NR1>}
HCOPy?
```



Arguments

`STARt or nonzero value`
starts hard copy output.

`ABORt or zero value`
stops hard copy output.

Examples

`:HCOPY START`
starts hard copy output on the specified output destination.

***NOTE**. During the execution of a HCOPy START command, use the *WAI command to confirm the completion of the first hard copy before starting the next hard copy output.*

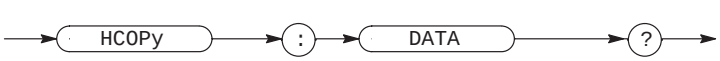
HCOPy:DATA?

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

Group HARDCOPY

Related Commands `HCOPy,HCOPy:PORT`

Syntax `HCOPy:DATA?`



Examples

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

HCOPy:FORMat (?)

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

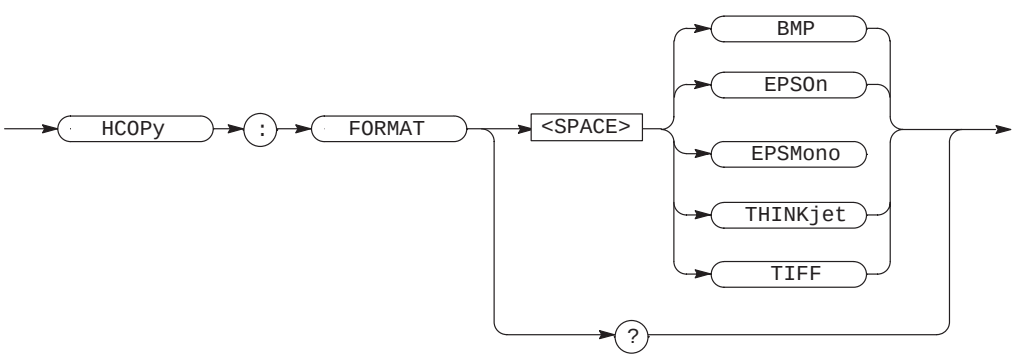
The HCOPy:FORMat? query returns the currently specified hard copy output format.

Group HARDCOPY

Related Commands HCOPy

Syntax

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



Arguments

BMP
the Windows monochrome file format.

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

EPSMono
the encapsulated Postscript format monochrome image file format.

THINkjet
the format used by HP inkjet printers.

TIFF
the TIFF format.

Examples

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

HCOPy:PORT (?)

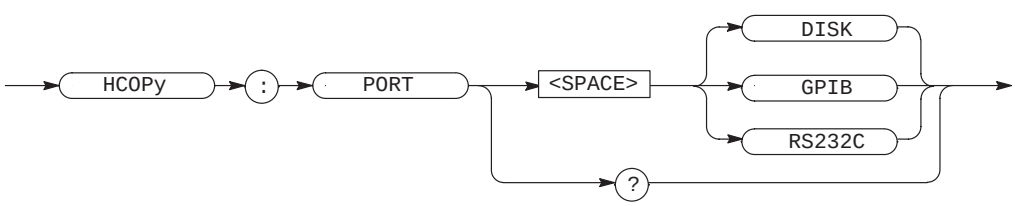
The HCOPy:PORT command sets the hard copy output port.

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

Group HARDCOPY

Related Commands HCOPy

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



Arguments

DISK
outputs to a file on the floppy disk.

GPIB
outputs to the GPIB port.

RS232c
outputs to the RS-232C port.

Examples

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

HEADer (?)

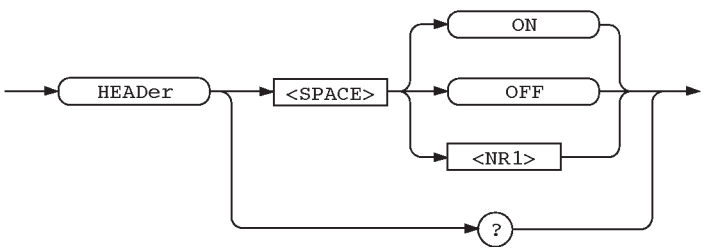
The HEADer command enables or disables the command header responses to all queries except IEEE Std 488.2 common commands. The HEADer? query returns the status indicating whether the command header responses are enabled or not.

Group SYSTEM

Related Commands VERBose

Syntax

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



Arguments

`ON or nonzero value`
enables the command header responses.

`OFF or zero value`
disables the command header responses.

Responses

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

Examples

`:HEADER OFF`
disables the command header responses.

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

HWSequencer? (AWG2041)

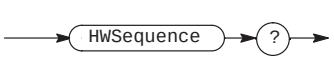
The `HWSequencer?` query returns whether or not the instrument is equiped with a hardware sequencer, and the instrument is currently using the hardware sequencer function.

Group SYSTEM

Related Commands `HWSequencer:MODE`, `HWSequencer: MODE?`, `HWSequencer:INSTalled?`

Syntax

```
HWSequencer?
```



Arguments None

Responses `[:HWSEQUENCER:INSTALLED] <Installed State>;[MODE] <Mode State>`
where
`<Installed State>::={1|0}`
1 The instrument is equiped with a hardware sequencer.
0 The instrument is not equiped with a hardware sequencer.
`<Mode State>::={1|0}`
1 The instrument is using the hardware sequencer function.
0 The instrument is not using the hardware sequencer function.

Examples `:HWSequencer?`
returns `:HWSEQUENCER:INSTALLED 1;MODE 0`

HWSequencer:INSTalled? (AWG2041)

The `HWSequencer:INSTalled?` query returns whether or not the instrument is equiped with a hardware sequencer.

Group SYSTEM

Related Commands `HWSequencer:MODE, HWSequencer: MODE?`

Syntax `HWSequencer:INSTalled?`

Arguments None

Responses `[:HWSEQUENCER:INSTALLED] <Installed State>`
where
`<Installed State>::={1|0}`
1 The instrument is equiped with a hardware sequencer.
0 The instrument is not equiped with a hardware sequencer.

Examples `:HWSequncer:INSTalled?`
returns `:HWSEQUENCER:INSTALLED 1`

HWSequencer:MODE (?) (AWG2041)

The HWSequencer:MODE command sets whether or not the hardware sequencer function is available. When the instrument is not equiped with a hardware sequencer, this command has no effect. After changing the setting, the instrument initiates a reboot.

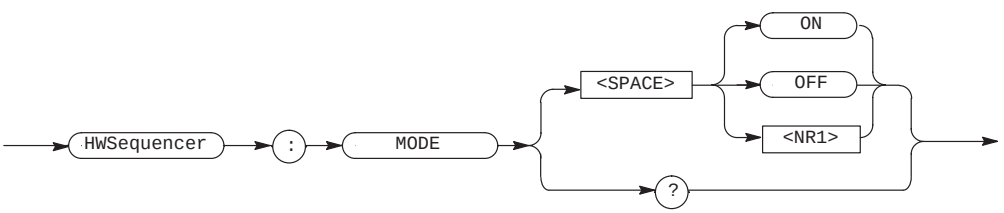
The HWSequencer:MODE? query returns whether or not the hardware sequencer function is currently available. When the instrument is not equiped with a hardware sequencer, the system returns 0 (zero).

***NOTE**. When you change the hardware sequencer mode, the files in the catalog memory of the instrument are lost. Before changing the hardware sequencer mode, save the files that you do not want to loose in the instrument's nonvolatile memory or a floppy disk.*

Group SYSTEM

Related Commands HWSequencer?, HWSequencer:INSTalled?

Syntax
```
HWSequencer:MODE {ON | OFF | <NR1>}
HWSequencer:MODE?
```



Arguments

ON or nonzero value	Set the instrument to use the hardware sequencer.
OFF or zero value	Set the instrument to not use the hardware sequencer.

Responses
```
[:HWSEQUENCER:MODE] <Mode State>
where   <Mode State>::={1|0}
```
1 The instrument is using the hardware sequencer mode.
0 The instrument is not using the hardware sequencer mode.

Examples
```
:HWSequencer:MODE ON
```
sets the hardware sequencer mode to on.

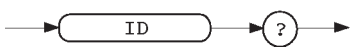
ID?

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

Group SYSTEM

Related Commands *IDN?

Syntax ID?



Arguments None

Responses ID <Manufacturer>/<Model>, <Firmware Level>
where
<Manufacturer>::=SONY_TEK,
<Model>::=AWG2005 | AWG2020 | AWG2021 | AWG2040 | AWG2041
<Firmware Level>::=CF:<Code and Format Version>, and
FV:<Firmware Version>.

Examples :ID?
returns SONY_TEK/AWG2020,CF:91.1CT,FV:1.00

*IDN?

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

Group SYSTEM

Related Commands ID?

Syntax *IDN?

Arguments None

Responses

<Manufacturer>, <Model>, <Serial Number>, <Firmware Level>
where
<Manufacturer>::=SONY/TEK,
<Model>::=AWG2005 | AWG2020 | AWG2021 | AWG2040 | AWG2041
<Serial Number>::=0,
<Firmware Level>::=CF:<Code and Format Version>,
<sp>FV:<Firmware Version>, and
<sp>::= Space.

Examples

*IDN?
might return SONY/TEK,AWG2020,0,CF:91.1CT FV:1.00

LOCk (?)

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

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

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

Group

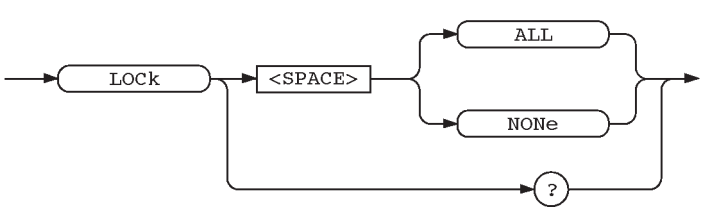
SYSTEM

Related Commands

UNLock

Syntax

LOCk {ALL | NONe}
LOCk?



Arguments

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

NONe
enables the front panel buttons and the knob.

Examples

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

*LRN?

The *LRN? common query returns all current settings for the waveform generator. The settings returned are in the format of a sequence of commands. If you save this query response, you can send it back later as a command sequence to reestablish the saved settings.

Group SYSTEM

Related Commands

Syntax *LRN?

Arguments None

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

Examples

*LRN?
might return the following response.

```
:HEADER 1;:VERBOSE 1;:DIAG:SELECT ALL;:SELFCAL:SELECT ALL;:DIS-
PLAY:BRIGHTNESS 70;CATALOG:ORDER NAME1;:DISPLAY:CLOCK 0;MENU:SET-
UP:FORMAT GRAPHICS;:DISPLAY:MESSAGE:SHOW"";:FG:STATE 0;FREQUENCY
10.00000E+06;CH1:AMPLITUDE 1.000;OFFSET 0.000;POLARITY NORMAL;
SHAPE SINUSOID;:HCOPY:FORMAT BMP;PORT DISK;:DISK:FORMAT:TYPE
HD3;:MMEMORY:MSIS DISK;ALOAD:MSIS DISK;STATE 0;:MODE CONTINUOUS;
:TRIGGER:IMPEDANCE HIGH;LEVEL 1.4;POLARITY POSITIVE;SLOPE
POSITIVE;:CH1:WAVEFORM "";FILTER THRU;AMPLITUDE 1.000;OFFSET
0.000;MARKERLEVEL1:HIGH 2.0;LOW 0.0;:CH1:MARKERLEVEL2:HIGH
2.0;LOW 0.0;:CLOC:FREQUENCY 1.000000E+09;SOURCE INTERNAL;:OUTPUT
```

```
:CH1:NORMAL:STATE 0;:OUTPUT:CH1:INVERTED:STATE 0;:DEBUG:SNOOP:
STATE 0;DELAY:TIME 0.2;:DATA:DESTINATION "GPIB.WFM";ENCDG
RPBINARY;SOURCE "CH1";WIDTH 2
```

MARKer:DATA (?)

The MARKer:DATA command writes marker data to the file specified with the DATA:DESTination command.

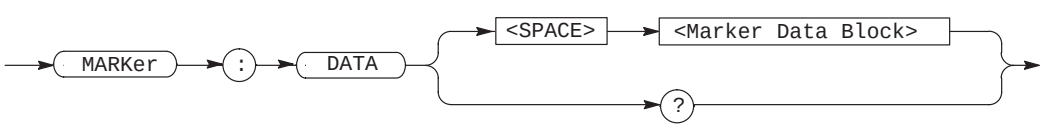
The MARKer:DATA? query returns marker data written in the file specified with the DATA:SOURce command.

Group WAVEFORM

Related Commands MARKER<x>:AOFF, MARKER<x>:POINt, DATA:DESTination, DATA:SOURce

Syntax

```
MARKer:DATA <Marker Data Block>
MARKer:DATA?
```



Arguments <Marker Data Block>::=<Arbitrary Block>
The format of a <Marker Data Block> is as follows:

#<x><yyy><marker(1)><marker(2)><marker(3)>....<marker(n)>

Here <yyy> is the number of bytes in the (ASCII format) marker data that follows, and <x> is the number of digits in <yyy>. The marker data items <marker(i)> consist of a single byte in which only the lower 2 bits are valid. These bits take on the values shown in the following table. The upper 6 bits must be set to 0.

Binary Data	Descriptions
0	Turn off marker 1 and marker 2
1	Turn on marker 2 and turn off marker 1
2	Turn on marker 1 and turn off marker 2
3	Turn on marker 1 and marker 2

Marker 2 in the table above is only used with models AWG2020, AWG2021, AWG2040, and AWG2041.

This command sets all the marker data in a single operation. Use the MARKER<x>:POINt command to set sections of the marker data.

Examples

```
:MARKER:DATA #41000
```
sets marker data.

MARKER<x>:AOFF

The MARKER<x>:AOFF command resets all markers in the file specified by the DATA:DESTination command.

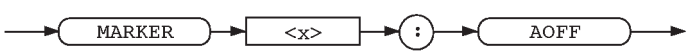
Group WAVEFORM

Related Commands MARKER<x>:POINt, MARKer:DATA, DATA:DESTination

Syntax

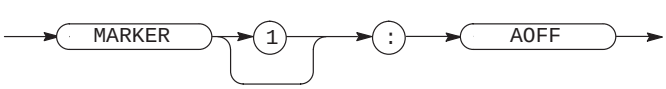
AWG2020/21/40/41

```
MARKER<x>:AOFF
```



AWG2005

```
MARKER[1]:AOFF
```



Arguments None

Examples

```
:DATA:DESTINATION "WAVE01.WFM";:MARKER1:AOFF
```
resets all channel 1 markers in the file: WAVE01.WFM.

MARKER<x>:POINt (?)

The MARKER<x>:POINt command sets or resets the marker of the channel specified at the data position specified in the file specified using the DATA:DESTination command.

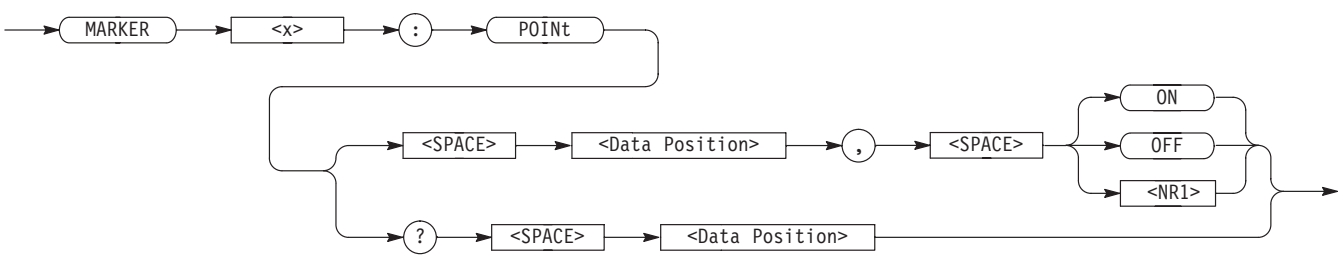
The MARKER<x>:POINt? query returns marker data state at the specified data position of the channel specified in the file specified using the DATA:SOURce command.

Group WAVEFORM

Related Commands MARKER<x>:AOFF, MARKer:DATA, DATA:DESTination, DATA:SOURce

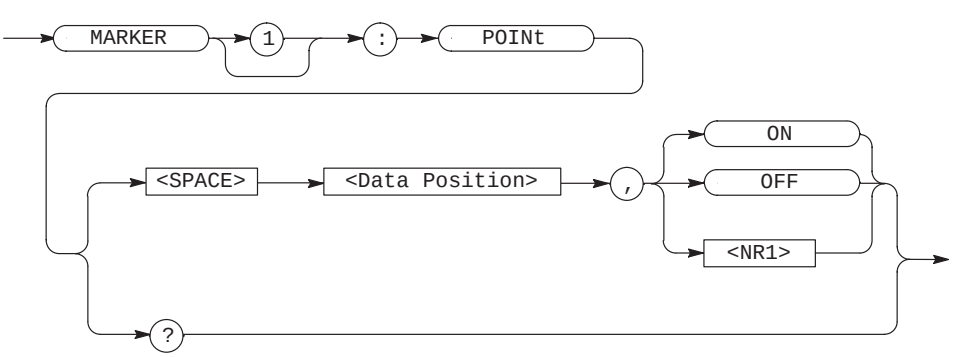
Syntax AWG2020/21/40/41

```
MARKER<x>:POINt <Data Position>, {OFF | ON | <NR1>}
MARKER<x>:POINt? <Data Position>
```



AWG2005

```
MARKER[1]:POINt <Data Position>,{ON | OFF | <NR1>}
MARKER[1]:POINt?
```



Arguments <Data Position>::=<NR1>, ON, or OFF
where <NR1> is a decimal integer, ON or nonzero sets a marker at <Data Position>, and OFF or zero value resets a marker at <Data Position>.

Use the MARKer:DATA command to set all the marker data in a single operation.

Responses AWG2020/21/40/41
[:MARKER{1 | 2]POINT]<Data Position,{0|1}

AWG2005
[:MARKER[1]:POINT]<Data Position>,{0|1}

Examples :DATA:DESTINATION "WAVE01.WFM";:MARKER1:POINT 2001, ON
sets marker at 2001st data point in channel 1 in the file WAVE01.WFM.

:DATA:SOURce "WAVE02.WFM";:MARKER1:POINT? 1400
might return :MARKER1:POINT 1400,1

MEMory?

The MEMory? query returns file-specific information on all files in the internal memory, and used size and unused size of the internal memory. This query is equivalent to sending the MEMory:CATalog:ALL? followed by the MEMory:FREE:ALL? queries.

Group MEMORY

Related Commands MEMory:CATalog:ALL?, MEMory:FREE:ALL?

Syntax MEMory?

Arguments None

Responses :MEMORY:CATALOG:ALL<File Entry>[,<File Entry>]...;
:MEMORY:FREE:ALL<Unused Size>, <Used Size>
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,

<File Size>::=<NR1>,
<Time Stamp>::=<string>,
<Unused Size>::=<NR1>, and
<Used Size>::=<NR1>.

Examples

:MEMORY?
might return the following response.
:MEMORY:CATALOG:ALL "AUTOSTEP.AST",142,"93-11-11 16:49","EQUATION.EQU",296,"93-11-11 16:54","SEQUENCE.SEQ",960,"93-11-11 16:48","WAVE2.WFM", 2948,"923-11-11 16:47","WAVEFORM.WFM", 2948,"93-11-11 16:47";:MEMORY:FREE:ALL 1696220,28500

MEMory:CATalog?

The MEMory:CATalog? query returns file-related information about all files in the internal memory. This query is equivalent to the MEMory:CATalog:ALL? query.

Group

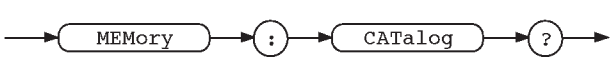
MEMORY

Related Commands

MEMory:CATalog:ALL?, MEMory?

Syntax

MEMory:CATalog?



Arguments

None

Responses

:MEMORY:CATALOG:ALL<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

The files with extensions of .BMP, .EPS, .EQA, .ESC, .ISF, .TIF, .TJ, .WFB, and .WVN on the floppy disk can be referenced only by MMEMory:CATalog?, MMEMory?, and MMEMory:CATalog:ALL? query.

Examples

:MEMORY:CATALOG?
might return the following response:

```
:MEMORY:CATALOG:ALL "AUTOSTEP.AST",142,"93-11-11 16:49","EQUA-
TION.EQU",296,"93-11-11 16:54","SEQUENCE.SEQ",960,"93-11-11
16:48","WAVE2.WFM",2948, "93-11-11 16:47","WAVE-
FORM.WFM",2948,"93-11-11 16:47"
```

MEMory:CATalog:ALL?

The MEMory:CATalog:ALL? query returns file-related information about all files in the internal memory.

Group MEMORY

Related Commands MEMory:CATalog?, MEMory:CATalog:AST?, MEMory:CATalog:CLK?, MEMory:CATalog:EQU?, MEMory:CATalog:SEQ?, MEMory:CATalog:WFM?, MEMory?

Syntax MEMory:CATalog:ALL?

Arguments None

Responses [:MEMORY:CATALOG:ALL]<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

The files with extensions of .BMP, .EPS, .EQA, .ESC, .ISF, .TIF, .TJ, .WFB, and .WVN on the floppy disk can be referenced only by MMEMory:CATalog:ALL?, MMEMory?, and MMEMory:CATalog? query.

Examples :MEMORY:CATALOG:ALL?
might return the following response.

```
:CATALOG:ALL "AUTOSTEP.AST",142,
"93-11-11 16:49","EQUATION.EQU",296,"93-11-11 16:54",
"SEQUENCE.SEQ",960,"93-11-11 16:48","WAVE2.WFM",2948,
"93-11-11 16:47","WAVEFORM.WFM",2948,"93-11-11 16:47"
```

MEMory:CATalog:AST?

The MEMory:CATalog:AST? query returns file-related information about all auto step files in the internal memory of the waveform generator.

Group MEMORY

Related Commands MEMory:CATalog:ALL?, MEMory?

Syntax MEMory:CATalog:AST?

Arguments None

Responses :MEMORY:CATALOG:AST<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

Examples :MEMORY:CATALOG:AST?
might return :MEMORY:CATALOG:AST "AUTOSTEP.AST",142,"93-11-11 16:49"

MEMory:CATalog:EQU?

The MEMory:CATalog:EQU? query returns file-related information about all equation files in the internal memory of the waveform generator.

Group MEMORY

Related Commands MEMory:CATalog:ALL?, MEMory?

Syntax MEMory:CATalog:EQU?

Arguments None

Responses
```
:MEMORY:CATALOG:EQU<File Entry>[,<File Entry>]...
```
where
```
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.
```

Examples
```
:MEMORY:CATALOG:EQU?
```
might return `:MEMORY:CATALOG:EQU "EQUATION.EQU", 296,"93-11-11 16:54"`

MEMory:CATalog:CLK? (AWG2005)

The `MEMory:CATalog:CLK?` query returns file-specific information about all clock sweep files in the internal memory.

Group MEMORY

Related Commands `MEMory:CATalog:ALL?`, `MEMory?`

Syntax
```
MEMory:CATalog:CLK?
```

Arguments None

Responses
```
[:MEMORY:CATALOG:CLK]<File Entry>[,<File Entry>]...
```
where
```
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.
```

Examples

```
MEMORY:CATALOG:CLK?
```

might return

```
:MEMORY:CATALOG:CLK "CLKSWEEP.CLK",10876,"93-10-10
12:53"
```

MEMory:CATalog:CLK? (AWG2005)

The MMEMory:CATalog:CLK? query returns file-specific information about all clock sweep files in the current mass memory.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog:ALL?, MMEMory?

Syntax

```
MMEMory:CATalog:CLK?
```

Arguments None

Responses

```
[:MMEMORY:CATALOG:CLK]<File Entry>[,<File Entry>]...
```

where

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

Examples

```
:MMEMORY:CATALOG:CLK?
```

might return

```
:MMEMORY:CATALOG:CLK "CLKSWEEP.CLK",10876,"93-09-28
12:53"
```

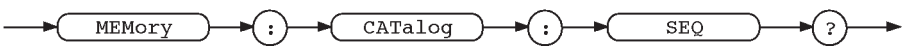
MEMory:CATalog:SEQ?

The MEMory:CATalog:SEQ? query returns file information on all sequence files in the internal memory of the waveform generator.

Group MEMORY

Related Commands MEMory:CATalog:ALL?, MEMory?

Syntax MEMory:CATalog:SEQ?



Arguments None

Responses :MEMORY:CATALOG:SEQ<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

Examples :MEMORY:CATALOG:SEQ?
might return :MEMORY:CATALOG:SEQ "SEQUENCE.SEQ",960,"93-11-11 16:48"

MEMory:CATalog:WFM?

The MEMory:CATalog:WFM? query returns file-specific information about all waveform files in the internal memory of the waveform generator.

Group MEMORY

Related Commands MEMory:CATalog:ALL?, MEMory?

Syntax MEMory:CATalog:WFM?

Arguments None

Responses
```
:MEMORY:CATALOG:WFM<File Entry>[,<File Entry>]...
```
where
```
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.
```

Examples
```
:MEMORY:CATALOG:WFM?
```
might return the following response:
```
:MEMORY:CATALOG:WFM "WAVE2.WFM",2948,"92-04-23 16:47","WAVE-
FORM.WFM", 2948,"93-11-11 16:47"
```

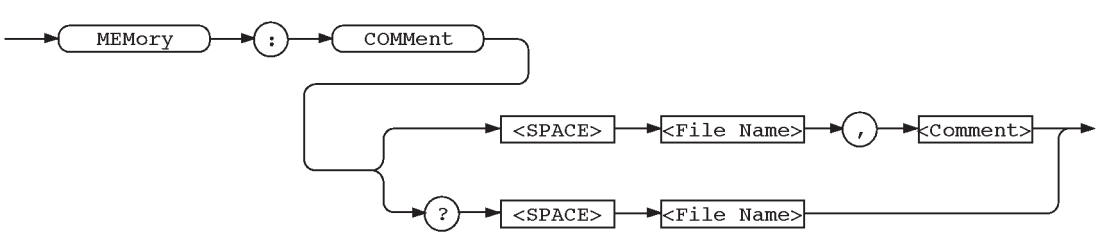
MEMory:COMMent (?)

The MEMory:COMMent command writes a comment into the comment column of the specified file in the internal memory of the waveform generator. The MEMory:COMMent? query returns comments in the comment column of the specified file. A comment cannot be written to a file that is locked using the MEMory:LOCk command.

Group MEMORY

Related Commands MEMory:COPY, MEMory:DELete, MEMory:REName, MEMory:LOCk

Syntax
```
MEMory:COMMent <File Name>, <Comment>
MEMory:COMMent? <File Name>
```



Arguments
```
<File Name>::=<string>
```
where <string> is the name of the file to which to write the comment.

<Comment>::=<string>
where <string> is a comment of up to 24 characters.

Examples :MEMORY:COMMENT "TDS_REF.WFM","COPIED FROM TDS REF."
writes the comment into the file TDS_REF.WFM.

MEMory:COPY

The MEMory:COPY command copies a file in internal memory. If the destination file <To-file> does not exist, it will be created. If the destination file already exists, it will be overwritten. (Files locked using the MEMory:LOCk command cannot be overwritten by MEMory:COPY.)

Group MEMORY

Related Commands MEMory:DELete, MEMory:REName, MEMory:COMMent

Syntax MEMory:COPY <From-file>, <To-file>

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

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

Examples :MEMORY:COPY "TDS_REF.WFM","AWGCH1.WFM"
copies the file TDS_REF.WFM to the file AWGCH1.WFM.

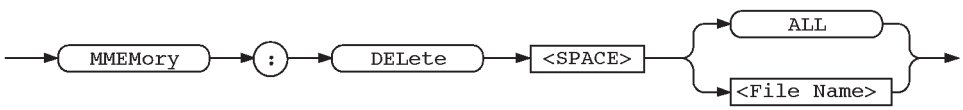
MEMory:DELete

The MEMory:DELete command deletes a file in the internal memory. A file locked with the MEMory:LOCk command cannot be deleted.

Group MEMORY

Related Commands MEMory:COPY, MEMory:REName, MEMory:COMMent

Syntax MEMory:DELete {All | <File Name>}



Arguments <File Name>::=<string>
where <string> is either the name of the file to be deleted or ALL when every file in internal memory is to be deleted.

Examples :MEMORY:DELETE "AWGCH2.WFM"
deletes the file AWGCH2.WFM from internal memory.

MEMory:FREE?

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

Group MEMORY

Related Commands MEMory:FREE:ALL?, MEMory?

Syntax MEMory:FREE?

Arguments None

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

Examples :MEMORY:FREE?
might return :MMEMORY:FREE:ALL 1696220,28500

MEMory:FREE:ALL?

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

Group MEMORY

Related Commands MEMory:FREE?, MEMory?

Syntax MEMory:FREE:ALL?

Arguments None

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

Examples :MEMORY:FREE:ALL?
might return :MEMORY:FREE:ALL 1696220,28500.

MEMory:LOCk(?)

The MEMory:LOCk command locks or unlocks a file in the internal memory; the MEMory:LOCk? query returns status indicating whether a file is locked or not. The following operations can not be performed on a locked file:

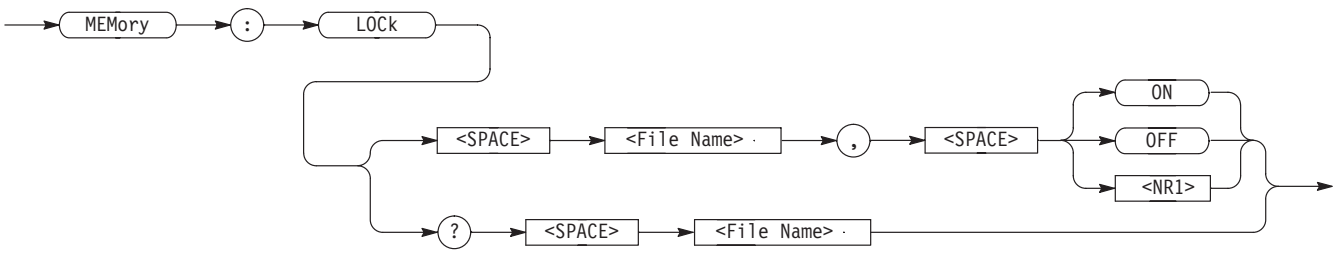
- File deletion using MEMory:DELete
- File overwriting using MEMory:COPY or load operations
- Commenting of files using MEMory:COMMent
- File renaming using MEMory:REName

Group MEMORY

Related Commands MEMory:DELete, MEMory:COPY, MEMory:REName, MEMory:COMMent

Syntax

```
MEMory:LOCk <File Name>, {ON | OFF | <NR1>}
MEMory:LOCk? <File Name>
```



Arguments

<File Name>::=<string>
where <string> is the name of the file to be locked or unlocked,
ON or a nonzero value (locks the file), and
OFF or zero value (unlocks the file).

Examples

```
:MEMORY:LOCK "RAMP_W1.WFM", 1
```
locks the file RAMP_W1.WFM.

MEMory:REName

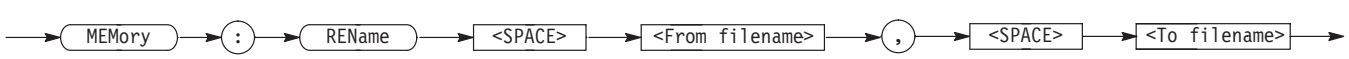
The MEMory:REName command changes the name of a file located in the internal memory of the waveform generator. A file that is locked using the MEMory:LOCk command cannot be renamed.

Group MEMORY

Related Commands MEMory:COPY, MEMory:DELete, MEMory:COMMent, MEMory:LOCk

Syntax

```
MEMory:REName <From-filename>, <To-filename>
```



Arguments

<From-filename>::=<string>
where <string> is the name of the file before it is renamed.

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

The file extensions in both files must be same. Specifying different extensions in both files causes an error.

Examples `:MEMORY:RENAME "TDS_REF.WFM","AWGCH2.WFM"`
renames the file TDS_REF.WFM to AWGCH2.WFM.

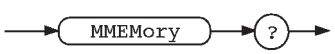
MMEMory?

The `MMEMory?` query returns all information, including autoload settings, used size, and unused sized, of all files in current mass memory. This query is equivalent to the `MMEMory:ALOad?` query, followed by `MMEMory:MSIS?` query, followed by the `MMEMory:CATalog:ALL?` query, followed by the `MMEMory:FREE:ALL?` query.

Group MEMORY

Related Commands `MMEMory:ALOad?`, `MMEMory:MSIS`, `MMEMory:CATalog:ALL?`, `MMEMory:FREE:ALL?`

Syntax `MMEMory?`



Arguments None

Responses
```
:MMEMORY:MSIS <Current Mass Memory>;CATALOG:ALL<File Entry>
[,<File Entry>]...;:MMEMORY:ALOAD:MSIS<AutoLoad Mass Memory>;
STATE<AutoLoad State>;:MMEMORY:FREE:ALL<Unused Size>, <Used Size>
```
where
```
<Current Mass Memory>::={DISK|NVRAM},
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>,
<Time Stamp>::=<string>,
<AutoLoad Mass Memory>::={DISK|NVRAM},
<AutoLoad State>::={0|1},
<Unused Size>::=<NR1>, and
<Used Size>::=<NR1>.
```

The files with extensions of `.BMP`, `.EPS`, `.EQA`, `.ESC`, `.ISF`, `.TIF`, `.TJ`, `.WFB`, and `.WVN` on the floppy disk can be referenced only by `MMEMory?` query, `MMEMory:CATalog?` query, and `MMEMory:ACTalog:ALL?` query.

Examples

```
:MMEMORY?
```

might return the following response

```
:MMEMORY:MSIS DISK;CATALOG:ALL "AUTOSTEP.AST",142,"93-11-11
16:49","EQUATION.EQU",296,"93-11-11 16:54","SEQUENCE.SEQ",
960,"93-11-11 16:48","WAVE2.WFM",2948,"93-11-11 16:47",
"WAVEFORM.WFM", 2948,"93-11-11 16:47;:MMEMORY:ALOAD:MSIS DISK;
STATE 0;:MMEMORY:FREE:ALL 801792,672760
```

MMEMory:ALOad?

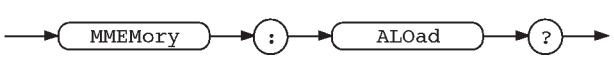
The MMEMory:ALOad? query returns status indicating whether an auto load is done at power up and which storage media, the floppy disk or NVRAM, is currently set to be loaded from. This query is equivalent to the MMEMory:ALOad:STATe? query.

Group MEMORY

Related Commands MMEMory:ALOad:MSIS, MMEMory:ALOad:STATe

Syntax

```
MMEMory:ALOad?
```



Arguments None

Responses

```
MMEMORY:ALOAD:MSIS<AutoLoad Mass Memory>;STATE<AutoLoad>
```

where

```
<AutoLoad Mass Memory>::={DISK|NVRAM},
<AutoLoad>::={0|1},
```

1 indicates the waveform generator is set to auto load at power up, and
0 indicates the waveform generator is set to not auto load at power up.

Examples

```
:MMEMORY:ALOAD?
```

might return `:MMEMORY:ALOAD:MSIS DISK;STATE 1`

MMEMory:ALOad:MSIS (?)

The MMEMory:ALOad:MSIS command designates the internal NVRAM or the floppy disk drive of the waveform generator to be the current mass memory. The current mass memory is the storage media from which files are loaded into internal memory when the auto load function is performed.

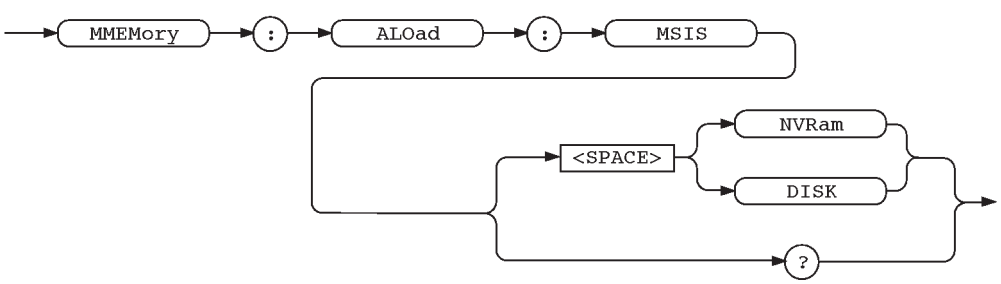
The MMEMory:ALOad:MSIS? query returns the storage type, NVRAM or DISK, currently selected as a mass memory for doing auto loads.

Group MEMORY

Related Commands MMEMory:ALOad:STATe, MMEMory:ALOad?

Syntax

```
MMEMory:ALOad:MSIS {NVRam | DISK}
MMEMory:ALOad:MSIS?
```



Arguments

NVRam
selects nonvolatile RAM.

DISK
selects floppy disk.

When DISK is specified as a mass memory, files in the directory of "\AWG2005", \AWG2020", \AWG2021", "\AWG2040" or "\AWG2041" are loaded.

***NOTE**. When the files are loaded from the floppy disk to the internal memory, the file names with extension of .ISF, .WFB, .WVN, and .EQA are converted to those of .WFM, .WFM, .WFM, and EQU.*

Examples

```
:MMEMORY:ALOAD:MSIS DISK
```
selects the floppy disk drive as a mass memory.

MMEMory:ALOad:STATe (?)

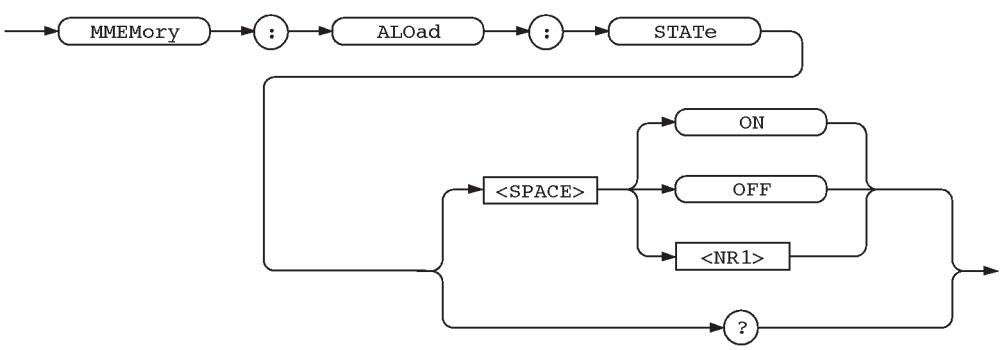
The MMEMory:ALOad:STATe command defines whether the auto load function is performed at power up. The MMEMory:ALOad:STATe? query returns status indicating whether the auto load function is performed or not at power up.

Group MEMORY

Related Commands MMEMory:ALOad:MSIS, MMEMory:ALOad?

Syntax

```
MMEMory:ALOad:STATe {ON | OFF | <NR1>}
MMEMory:ALOad:STATe?
```



Arguments

ON
or a nonzero value sets the instrument so as to perform the auto load at power up.

OFF
or a zero value resets the instrument so as not to perform the auto load at power up.

Responses

1 auto loading is currently enabled.
0 the auto loading is currently disabled.

Examples

```
:MMEMORY:ALOAD:STATE 1
```
sets the instrument so auto loading is performed upon power up.

MMEMory:CATalog?

The MMEMory:CATalog? query returns file-specific information about all files in the current mass memory. This query is equivalent to the MMEMory:CATalog:ALL? query.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog:ALL?, MMEMory?

Syntax MMEMory:CATalog?

Arguments None

Responses :MMEMORY:CATALOG:ALL<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

The files with extensions of .BMP, .EPS, .EQA, .ESC, .ISF, .TIF, .TJ, .WFB, and .WVN on the floppy disk can be referenced only by MMEMory? query, MMEMory:CATalog? query, and MMEMory:ACTalog:ALL? query.

Examples :MMEMORY:CATALOG?
might return the following response.
MMEMORY:CATALOG:ALL "AUTOSTEP.AST",142,"93-11-11 16:49","EQUATION.EQU",296,"93-11-11 16:54","SEQUENCE.SEQ",960,"93-11-11 16:48","WAVE2.WFM",2948,
"93-11-11 16:47","WAVEFORM.WFM",2948,"93-11-11 16:47"

MMEMory:CATalog:ALL?

The MMEMory:CATalog:ALL? query returns file-specific information about all the files in the current mass memory.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog?, MMEMory:CATalog:AST?, MMEMory:CATalog:CLK?, MMEMory:CATalog:EQU?, MMEMory:CATalog:SEQ?, MMEMory:CATalog:WFM?, MMEMory?

Syntax MMEMory:CATalog:ALL?

Arguments None

Responses :MMEMORY:CATALOG:ALL<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

The files of extensions of .BMP, .EPS, .EQA, .ESC, .ISF, .TIF, .TJ, .WFB, and .WVN on the floppy disk can be referenced only by MMEMory:CATalog:ALL? query, MMEMory? query, and MMEMory:CATalog? query.

Examples :MMEMORY:CATALOG:ALL?
might return the following response;
:MMEMORY:CATALOG:ALL "AUTOSTEP.AST",142,"93-11-11 16:49","EQUATION.EQU",296,"93-11-11 16:54",
"SEQUENCE.SEQ",960,"93-11-11 16:48","WAVE2.WFM",2948,
"93-11-11 16:47","WAVEFORM.WFM",2948,"93-11-11 16:47"

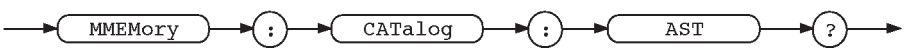
MMEMory:CATalog:AST?

The MMEMory:CATalog:AST? query returns file-specific information about all the auto step files in the current mass memory.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog:ALL?, MMEMory?

Syntax MMEMory:CATalog:AST?



Arguments None

Responses :MMEMORY:CATALOG:AST<File Entry>[,<File Entry>]...
where
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.

Examples :MMEMORY:CATALOG:AST?
might return :MMEMORY:CATALOG:AST "AUTOSTEP.AST",142,"93-11-11 16:49".

MMEMory:CATalog:EQU?

The MMEMory:CATalog:EQU? query returns file-specific information about all files in the current mass memory.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog:ALL?, MMEMory?

Syntax MMEMory:CATalog:EQU?

Arguments None

Responses
```
:MMEMORY:CATALOG:EQU<File Entry>[,<File Entry>]...
```
where
```
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>
```

Examples
```
:MMEMORY:CATALOG:EQU?
```
might return
```
:MMEMORY:CATALOG:EQU "EQUATION.
EQU",296,"93-11-11 16:54"
```

MMEMory:CATalog:SEQ?

The `MMEMory:CATalog:SEQ?` query returns file-specific information about all sequence files in the current mass memory.

Group MEMORY

Related Commands `MMEMory:MSIS`, `MMEMory:CATalog:ALL?`, `MMEMory?`

Syntax
```
MMEMory:CATalog:SEQ?
```

Arguments None

Responses
```
:MMEMORY:CATALOG:SEQ<File Entry>[,<File Entry>]...
```
where
```
<File Entry>::=<File Name>, <File Size>, <Time Stamp>,
<File Name>::=<string>,
<File Size>::=<NR1>, and
<Time Stamp>::=<string>.
```

Examples

```
:MMEMORY:CATALOG:SEQ?
```

might return `:MMEMORY:CATALOG:SEQ "SEQUENCE.SEQ", 960, "93-11-11 16:48"`

MMEMory:CATalog:WFM?

The `MMEMory:CATalog:WFM?` query returns file-specific information about all waveform files in the current mass memory.

Group MEMORY

Related Commands `MMEMory:MSIS`, `MMEMory:CATalog:ALL?`, `MMEMory?`

Syntax

```
MMEMory:CATalog:WFM?
```

Arguments None

Responses

```
:MMEMORY:CATALOG:WFM<File Entry>[,<File Entry>]...
```

where

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

Examples

```
:MMEMORY:CATALOG:WFM?
```

might return the following response:

```
:MMEMORY:CATALOG:WFM "WAVE2.WFM",2948,"93-11-11 16:47","WAVE-
FORM.WFM",2948,"93-11-11 16:47"
```

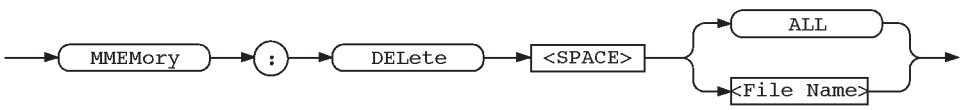
MMEMory:DELete

The `MMEMory:DELete` command deletes a file in the current mass memory. A file locked with the `MMEMory:LOCk` command cannot be deleted.

Group MEMORY

Related Commands `MMEMory:REName`, `MMEMory:MSIS`, `MMEMory`

Syntax `MMEMory:DELete {All | <File Name>}`



Arguments `<File Name>::=<string>`
where `<string>` is the name of the file to be deleted or the word `ALL` to delete all of the files in current mass memory.

Examples `:MMEMORY:DELETE "AWG2.WFM"`
deletes the file AWG2.WFM.

MMEMory:FREE?

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

Group MEMORY

Related Commands `MMEMory:MSIS`, `MMEMory:FREE:ALL?`, `MMEMory?`

Syntax `MMEMory:FREE?`

Arguments None

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

Examples `:MMEMORY:FREE?`
might return `:MMEMORY:FREE:ALL 1696220,28500`

MMEMory:FREE:ALL?

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

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:FREE?, MMEMory?

Syntax MMEMory:FREE:ALL?

Arguments None

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

Examples :MMEMORY:FREE:ALL?
might return :MMEMORY:FREE:ALL 801792,672760

MMEMory:LOAD

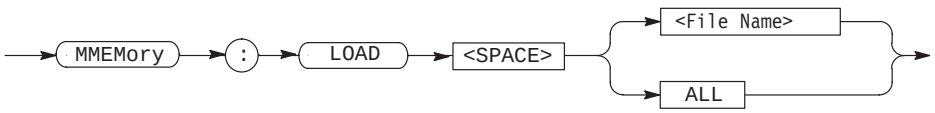
The MMEMory:LOAD command loads the file(s) in the current mass memory into internal memory. If the file to be loaded does not exist in internal memory, it will be created. If a file with the same file name already exists in internal memory, it will be overwritten unless it has been locked.

When the files are loaded from the floppy disk to the internal memory, the file names with extension of .ISF, .WFB, .WVN, and .EQA are converted to those of .WFM, .WFM, .WFM, and .EQU.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:SAVE

Syntax MMEMory:LOAD {<File Name> | ALL}



Arguments `<File Name>::=<string>`
where `<string>` is the name of the file to be loaded or the word `ALL` to load all of files in the current mass memory.

Examples `:MMEMORY:LOAD ALL`
loads all files in the current mass memory into the internal memory

MMEMory:LOCk (?)

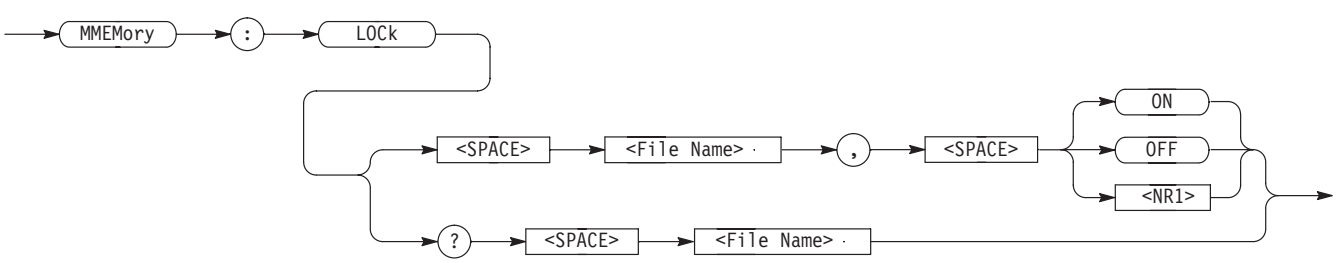
The `MMEMory:LOCk` command locks or unlocks a file in the current mass memory; the `MEMory:LOCk?` query returns status indicating whether a file is locked or not. The following operations can not be performed on a locked file:

- File deletion using `MMEMory:DELete`
- File overwriting using `MMEMory:COPY` or `MMEMory:LOAD`
- Commenting of files
- File renaming using `MMEMory:REName`

Group MEMORY

Related Commands `MMEMory:DELete`, `MMEMory:LOAD`, `MMEMory:REName`, `MMEMory:MSIS`

Syntax
```
MMEMory:LOCk <File Name>, {ON | OFF | <NR1>}
MMEMory:LOCk? <File Name>
```



Arguments <File Name>::=<string>
where <File Name> is the name of the file to be locked or unlocked,
ON or any nonzero value for <NR1> locks the file, and
OFF or a zero value for <NR1> unlocks the file.

Responses
0 the file is not locked
1 the file is locked

Examples :MMEMORY:LOCK "SINE_W1.WFM", 1
locks the file SINE_W1.WFM.

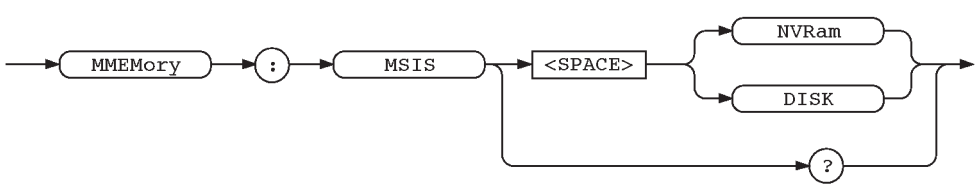
MMEMory:MSIS(?)

The MMEMory:MSIS command designates the internal NVRAM or the floppy disk of the waveform generator to be the current mass memory. The MMEMory:MSIS? query returns the type of current mass memory that is selected.

Group MEMORY

Related Commands MMEMory:MSIS, MMEMory:CATalog?, MMEMory:CATalog:AST?, MMEMory:CATalog:EQU?, MMEMory:CATalog:SEQ?, MMEMory:CATalog:WFM?, MMEMory?

Syntax
MMEMory:MSIS {NVRam | DISK}
MMEMory:MSIS?



Arguments
NVRam
selects non volatile RAM.

DISK
selects the floppy disk.

When DISK is selected as a current mass memory, you can change the current working directory using the DISK:CDIRectory command.

Examples `:MMEMORY:MSIS DISK;:DISK:CDIRECTORY "\SAMPLE1"`
selects `DISK` as a current mass memory and makes `\SAMPLE1` the current working directory.

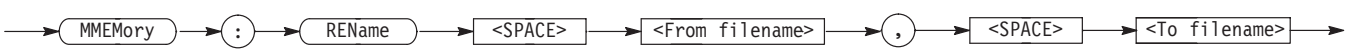
MMEMory:REName

The `MMEMory:REName` command changes the name of a file in the current mass memory. A file that is locked using the `MMEMory:LOCk` command cannot be renamed.

Group MEMORY

Related Commands `MMEMory:DELete`, `MMEMory:MSIS`, `MMEMory:LOCk`

Syntax `MMEMory:REName <From-filename>, <To-filename>`



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

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

The file extensions of both files must be same. Specifying a different file extension for the files causes an error.

Examples `:MMEMORY:RENAME "TDS_REF.WFM", "AWGCH2.WFM"`
renames the file `TDS_REF.WMF` to `AWGCH2.WFM`.

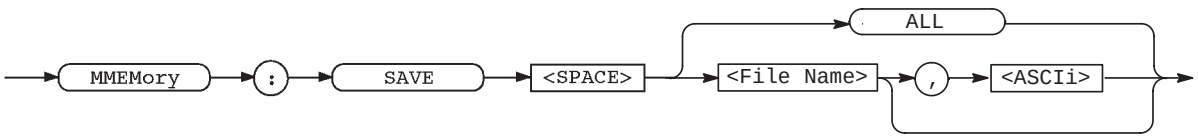
MMEMory:SAVE

The `MMEMory:SAVE` command saves files stored in the internal memory into the current mass memory. If the file to be saved does not exist in mass memory, it will be created. If a file with the same file name already exists in mass memory, it will be overwritten unless it has been locked.

Group MEMORY

Related Commands `MMEMory:LOAD`, `MMEMory:MSIS`

Syntax MMEMory:SAVE {<File Name>[,<ASCIi>] | ALL}



Arguments <File Name>::=<string>
where <string> is either the name of the file in internal memory to be saved or the word ALL, when all files in internal memory are to be saved in current mass memory. ASCIi saves the content of an equation file converted to ASCII code. Effective only when the <File Name> is a equation file (.EQU) and at the same time, current mass memory is DISK. The saved file has a extension of .EQA.

Examples :MMEMORY:SAVE ALL
saves all files in the internal memory into the current mass memory.

MODE (?)

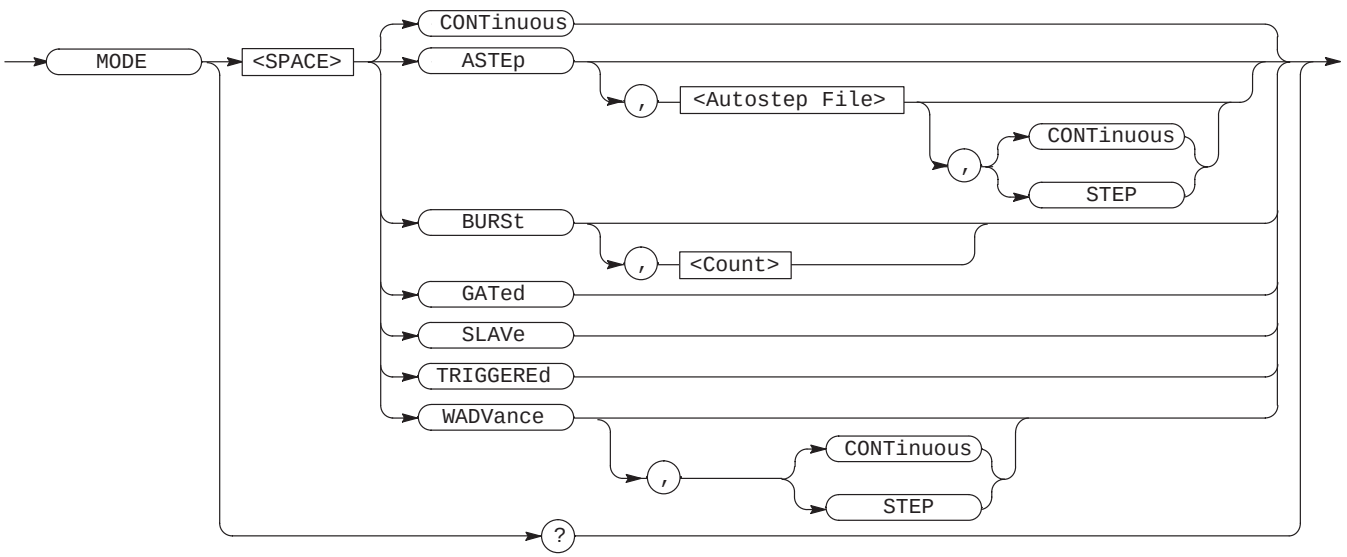
The MODE command selects the mode used to output a waveform or sequence. The MODE? query returns the current selected mode.

Group MODE

Related Commands STARt, STOP, *TRG

Syntax MODE {CONTinuous | ASTEp [,<Autostep File>[,CONTinuous | , STEP](AWG2005/40/41)] | BURst [,<Count>](AWG2020/21/40/41) | GATed SLAVe (AWG2040/41) | TRIGGERed | WADVance [,CONTinuous | ,STEP](AWG2005)}

MODE?



Arguments

Arguments	Descriptions
CONTinuous	Sets the continuous mode which continuously outputs waveform or sequence.
ASTep	Sets the auto step mode which outputs one cycle of a waveform or step of a sequence per trigger. For example, this mode advances one step per trigger of a sequence stored in an auto-step file. <Autostep File>::=<string> The optional argument can be added in the AWG2005/40/41. CONTinuous Perform the steps continuously. STEP Perform the step once.
BURSt (AWG2020/21/40/41)	Sets the burst mode which outputs <Count> waveform cycles or sequence steps for each trigger. <Count>::=<NR1> burst count (range: 1 to 65535)
GATed	Sets the gated mode which continuously outputs waveforms or sequences as long as the trigger remains enabled. The trigger remains effective as long as any of the following events occur: ■ the TRIGGER MANUAL button remains pressed ■ a valid external gate signal remains input ■ a STARt/*TRG command has been executed but a STOP command has not yet been issued
SLAVe (AWG2040/41)	Sets the slave mode which enable the slaved AWG's to operate as the master AWG's. The slaved AWG's synchronize with the trigger, gate, and stop signal generation in the master AWG's.

Arguments	Descriptions
TRIGGERED	Sets the triggered mode, which outputs one waveform cycle or sequence step for each trigger.
WADVance	Sets the waveform advance mode which continuously outputs one step of a sequence, as when advancing one step for each trigger. The optional argument can be added in the AWG2005. CONTinuous Outputs the waveforms continuously. STEP Outputs the waveform once.

Examples

```
:MODE BURST, 200
```

sets output for burst mode with 200 waveform cycles.

*OPC (?)

The *OPC common command generates the operation complete message by setting bit 0 in the SESR (Standard Event Status Register), when all pending operations are finished.

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

The following table lists the commands that generate an operation complete massage.

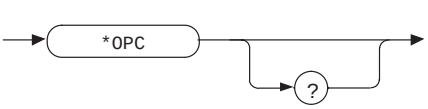
Command	Operation
EQUAtion:COMPile:STATe EXECute	Equation compile
HCOPy STARt	Hardcopy output

Group SYNCHRONIZATION

Related Commands *WAI

Syntax

```
*OPC
OPC?
```



Arguments None

Examples

```
EQUATION:COMPILE:STATE EXECUTE,"SAMPLE.EQU";*OPC
```

might wait for the completion of equation compile.

```
HCOPY:PORT DISK;HCOPY START;*OPC
```
might wait for the completion of hardcopy.

*OPT?

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

Group SYSTEM

Related Commands None

Syntax *OPT?

Arguments None

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

0	indicates no option,
CH2	indicates the option 02 (2 channel output) (AWG2020/21),
CH3/4	indicates the option 02 (4 channel output) (AWG2005),
DDO	indicates the option 03 (Digital data out) (AWG2020/21/40/41), indicates the option 04 (Digital data out) (AWG2005/21),
SWP	indicates the option 05 (Clock sweep) (AWG2005),
FPP	indicates the option 09 (Floating point processor), and
4M	indicates the option 01 (4MB word waveform memory) (AWG2040/41)

Examples *OPT?
might return CH2, FPP to indicate that the 2 channel and floating point processor options are installed in the instrument.

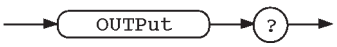
OUTPut?

The OUTPut? query returns all settings which can be set with the OUTPUT commands.

Group OUTPUT

Related Commands All output commands

Syntax OUTPut?



Arguments None

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

Examples :OUTPUT?
might return :OUTPUT:CH1:STATE 0;:OUTPUT:CH2:STATE 0;:OUTPUT:SYNC END

OUTPut:CH<x>?

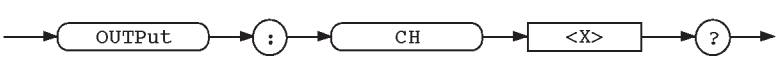
The OUTPut:CH<x>? query returns status indicating whether the output has been turned on or not.

In case of the AWG2040/41, CH1 is only valid header mnemonic.

Group OUTPUT

Related Commands OUTPut:CH<x>:STATe

Syntax OUTPut:CH<x>?



Arguments None

Responses

1 the output is currently turned on.
0 the output is currently turned off.

ON/OFF of the output changes the relay connected to output on the front panel, and is enabled by OUTPut:CH<x>:STATt. OUTPut:CH<x> checks the status of the relay, and has a same operation as OUTPut:CH<x>:STATt? query.

Examples

```
:OUTPUT:CH1?
```

might return `:OUTPUT:CH1:STATE 1`(AWG2005/20/21), or `OUTPUT: CH1:INVERTED:STATE1;OUTPUT:CH1:NORMAL:STATE1`(AWG2040/41)

OUTPut:CH<x>:STATe (?) (AWG2005/20/21)

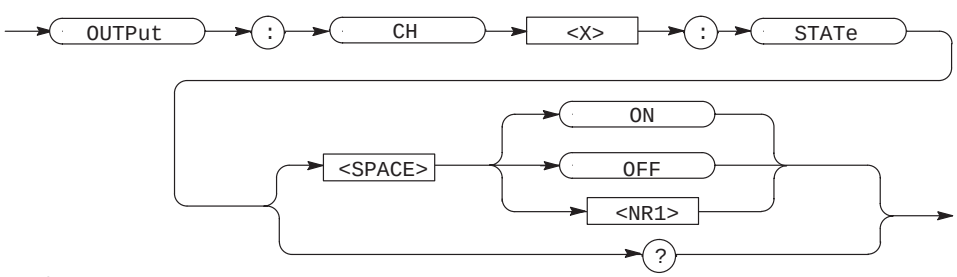
The `OUTPut:CH<x>:STATe` command turns waveform output on or off for the specified channel. The `OUTPut:CH<x>:STATe?` query returns status indicating whether the output is turned on or not.

Group OUTPUT

Related Commands `OUTPut:CH<x>?`, `OUTPut:CH1:NORMal:STATe`, `OUTPut:CH1:INVerted:STATe`

Syntax

```
OUTPut:CH<x>:STATe {ON | OFF | <NR1>}
OUTPut:CH<x>:STATe?
```



Arguments

`ON`
or any nonzero value for `<NR1>` turns the output on.

`OFF`
or any zero value turns output off.

ON/OFF of the output changes the relay connected to output on the front panel.

Responses

1	the output is currently turned on.
0	the output is currently turned off.

Examples

```
:OUTPUT:CH1:STATE 1
```

turns on the channel 1 output.

OUTPut:CH1:INVerted? (AWG2040/41)

The OUTPut:CH1:INVerted? query returns whether or not the inverting output is on.

Group OUTPUT

Related Commands OUTPut:CH1:INVerted:STATe

Syntax OUTPut:CH1:INVerted?

Arguments None

Responses [:OUTPUT:CH1:INVERTED:STATE]<State>
where

1 The inverting output is currently turned on.
0 The inverting output is currently turned off.

where <State>::=<NR1> is one of following responses:

Examples :OUTPUT:CH1:INVERTED?
queries whether the inverting output is on.

OUTPut:CH1:INVerted:STATe (?) (AWG2040/41)

The OUTPut:CH1:INVerted:STATe command sets the inverting output to be either on or off.

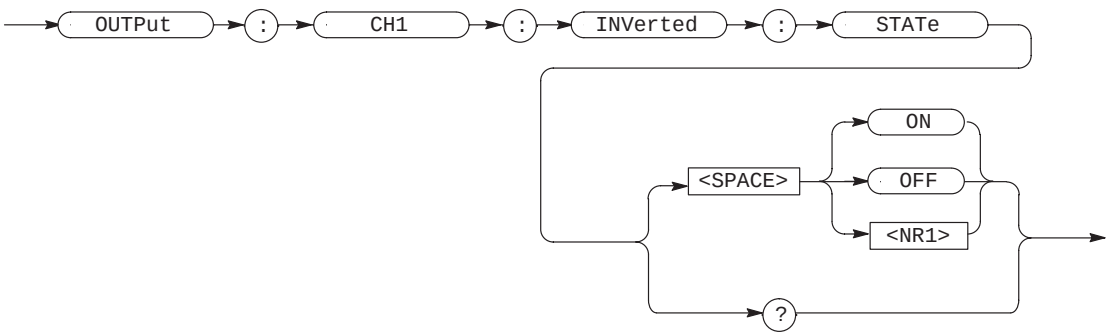
The OUTPut:CH1:INVerted:STATe? query returns whether or not the inverting output is on.

Group OUTPUT

Related Commands OUTPut:CH1:NORMal:STATe, OUTPut:CH1:INVerted?

Syntax

```
OUTPut:CH1:INVerted:STATe {ON | OFF | <NR1>}
OUTPut:CH1:INVerted:STATe?
```



Arguments

`ON or nonzero value`
turns the inverting output on.

`OFF or zero value`
turns the inverting output off.

Responses

1 The inverting output is currently on.
0 The inverting output is currently off.

Examples

```
:OUTPUT:CH1:INVERTED:STATE ON
```
turns the inverting output on.

OUTPut:CH1:NORMal? (AWG2040/41)

The `OUTPut:CH1:NORMal?` query returns whether or not the noninverting output is on.

Group OUTPUT

Related Commands `OUTPut:CH1:NORMal:STATe`

Syntax

```
OUTPut:CH1:NORMal?
```

Arguments None

Responses [:OUTPUT:CH1:NORMAL:STATE]<State>
where <State>::=<NR1> is one of following responses:

1 Noninverting output is currently turned on.
0 Noninverting output is currently turned off.

Examples :OUTPUT:CH1:NORMAL?
queries whether the noninverting output is on.

OUTPut:CH1:NORMal:STATe (?) (AWG2040/41)

The OUTPut:CH1:NORMal:STATe command sets the noninverting output to be either on or off.

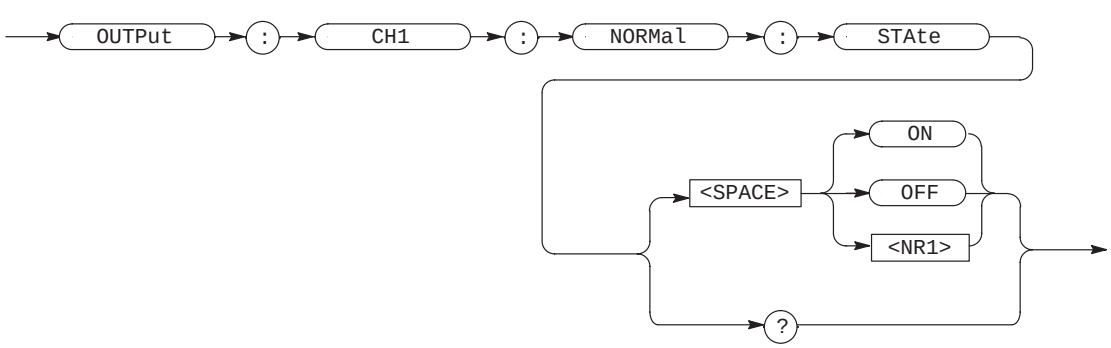
The OUTPut:CH1:NORMal:STATe? query returns whether or not the noninverting output is on.

This command is equivalent to the OUTPut:CH1:STATe command.

Group OUTPUT

Related Commands OUTPut:CH1:INVerted:STATe, OUTPut:CH1:NORMal?

Syntax OUTPut:CH1:NORMal:STATe {ON | OFF | <NR1>}
OUTPut:CH1:NORMal:STATe?



Arguments

`ON or nonzero value`
turns the noninverting output on.

`OFF or zero value`
turns the noninverting output off.

Responses

`1` The noninverting output is currently on.
`0` The noninverting output is currently off.

Examples

`:OUTPUT:NORMAL:STATE ON`
turns the noninverting output on.

OUTPut:SYNC (?) (AWG2020/21)

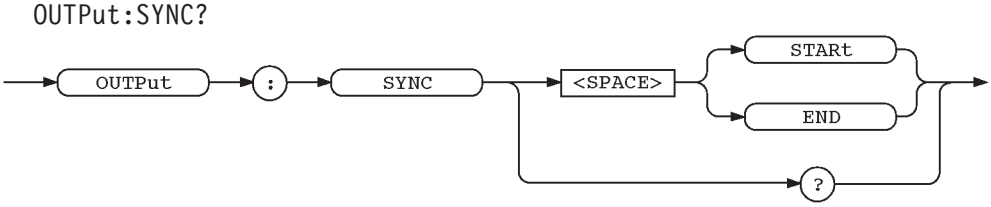
The `OUTPut:SYNC` command selects at what point on the waveform the sync signal is generated and output at the SYNC connector on the front panel. The `OUTPut:SYNC?` query returns the currently selected position.

Group OUTPUT

Related Commands

Syntax

```
OUTPut:SYNC {START | END}
OUTPut:SYNC?
```



Arguments

`START`
generates a sync signal when a waveform is triggered.

`END`
generates a sync signal at the end of a waveform.

Examples

`:OUTPUT:SYNC END`
sets the sync signal to output at the end of a waveform.

*PSC (?)

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

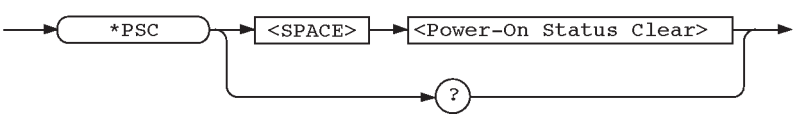
The *PSC? common query returns status of the power-on status clear flag.

Group STATUS and EVENT

Related Commands DESE, *ESE, FACTory, *SRE

Syntax

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



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

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

Responses

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

Examples

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

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

*RST

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

Group SYSTEM

Related Commands FACTory, SECUre

Syntax *RST

Arguments None

Examples *RST
resets the instrument.

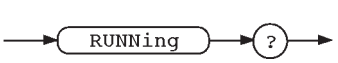
RUNNing (?)

The RUNNing? query returns status that indicates whether a waveform is being output or not.

Group MODE

Related Commands STARt, STOP

Syntax RUNNing?



Arguments None

Responses

1	a waveform or a sequence is being output.
0	nothing is being output.

Examples

```
:RUNNING?
```

might return `:RUNNING 1`.

SECUre

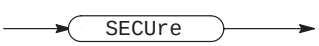
The `SECUre` command initializes all internal memory and internal nonvolatile memory and resets the waveform generator to its factory default settings.

Group SYSTEM

Related Commands `*RST`, `FACTory`

Syntax

```
SECUre
```



Examples

```
:SECURE
```

initializes all internal memory and internal nonvolatile memory and resets the waveform generator to its factory default settings.

SELFcal?

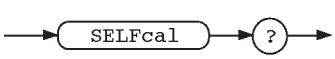
The `SELFcal?` query runs the selected calibration routine(s) and returns the results of its execution.

Group CALIBRATION and DIAGNOSTIC

Related Commands `SELFcal:SELect`, `SELFcal:STATe`, `SELFcal:RESUlt?`

Syntax

```
SELFcal?
```



Arguments None

Responses

:SELFCAL:SELECT<Calibration Routine>;RESULT <Result>[,<Result>]...
where <Calibration Routine>::= one of following arguments:

ALL	is all routines below
CLOCk	is the clock unit calibration routine (AWG2020/21)
SETUp	is the setup-related unit calibration routine
TRIGger	is the trigger unit calibration routine (AWG2005)

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

0	terminated without error
200	detected errors in the clock unit (AWG2020/21)
600	detected errors in the setup-related unit
800	detected errors in the trigger unit (AWG2005)

Examples

:SELFCAL?
might return :SELFCAL:SELECT ALL;RESULT 0

SELFcal:RESULt?

The SELFcal:RESUlt? query returns results of calibration execution.

Group

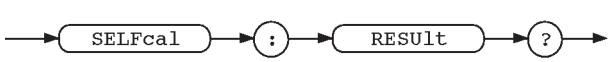
CALIBRATION and DIAGNOSTIC

Related Commands

SELFcal:SELect, SELFcal:STATe

Syntax

SELFcal:RESUlt?



Arguments

None

Responses

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

0	terminated without error.
200	detected errors in the clock unit. (AWG2020/21)
600	detected errors in the setup-related unit.
800	detected errors in the trigger unit (AWG2005/20/21)

Examples

```
:SELFCAL:RESULT?
```

queries the result of executing a calibration.

SELFcal:SELect (?)

The SELFcal:SELect command selects the calibration routine(s). The SELFcal:SELect? query returns the currently selected routine.

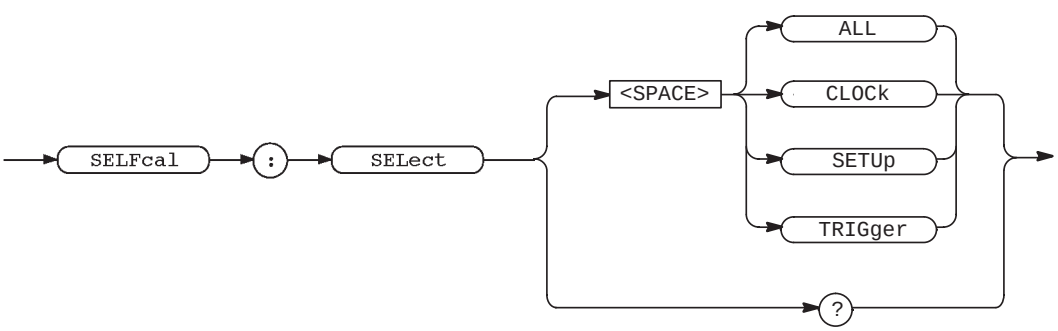
Group CALIBRATION and DIAGNOSTIC

Related Commands SELFcal:STATe, SELFcal:RESUlt?

Syntax

```
SELFcal:SELect {ALL | CLOCk(AWG2020/21) | SETUp |
TRIGger(AWG2005/20/21)}

SELFcal:SELect?
```



Arguments

ALL	calibrates all (both units listed below)
CLOCk	calibrates the clock unit (AWG2020/21)
SETUp	calibrates the unit related to instrument setup
TRIGger	calibrates the trigger unit (AWG2005/20/21)

Examples

```
:SELFCAL:SELECT CLOCK ;STATE EXECUTE
```

selects the clock for calibration and then calibrates it.

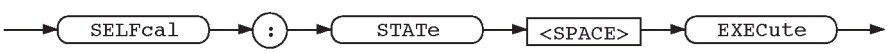
SELFcal:STATe

The SELFcal:STATe command executes the calibration routine(s) selected with the SELFcal:SELect command. If an error is detected during execution, the routine that detected the error stops immediately. If ALL (for all routines) is selected with the SELFcal:SELect command, self-calibration continues at the next routine.

Group CALIBRATION and DIAGNOSTIC

Related Commands SELFcal:SELect, SELFcal:RESult?

Syntax SELFcal:STATe EXECute



Arguments EXECute
performs calibration on selected routine.

Examples :SELFCAL:SELECT ALL; STATE EXECUTE; RESULT?
executes all calibration routines. After calibration is finished, the results are returned.

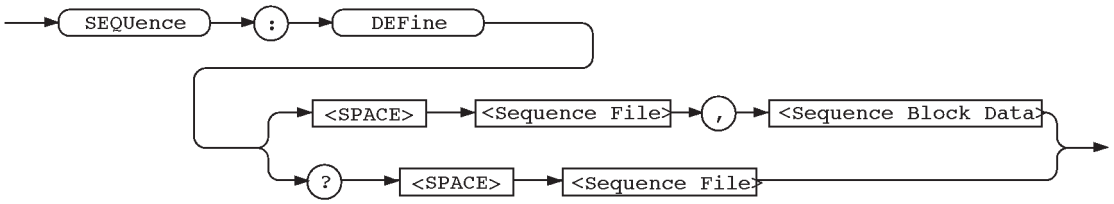
SEQUence:DEFine (?)

The SEQUence:DEFine command writes sequence data to the specified file. The SEQUence:DEFine? query returns sequence data that is written in the specified file.

Group WAVEFORM

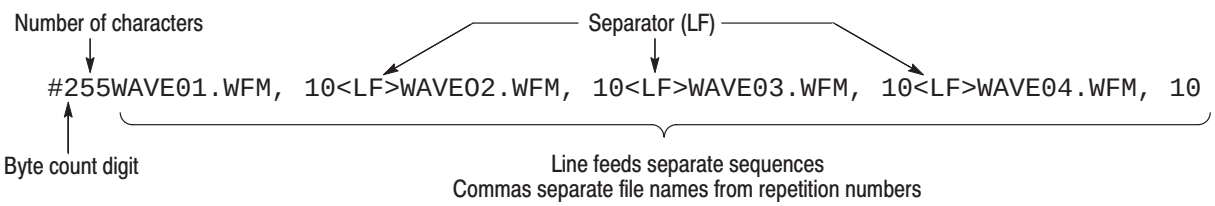
Related Commands None

Syntax SEQUence:DEFine <Sequence File>, <Sequence Block Data>
SEQUence:DEFine? <Sequence File>



Arguments <Sequence File>::=<string>
<Sequence Block Data>::=<Arbitrary Block>

where <Sequence Block Data> must be written in ASCII code and each sequence is separated by Line Feed (LF) code. The file name and repetition number are separated by a comma.



Examples

```
:SEQUENCE:DEFINE "SQWAVE.SEQ",
#255WAVEO1.WFM,1O<LF>WAVEO2.WFM,1O<LF>WAVEO
3.WFM,1O<LF>WAVEO4.WFM,1O
```

writes sequence data to the file SQWAVE.SEQ.

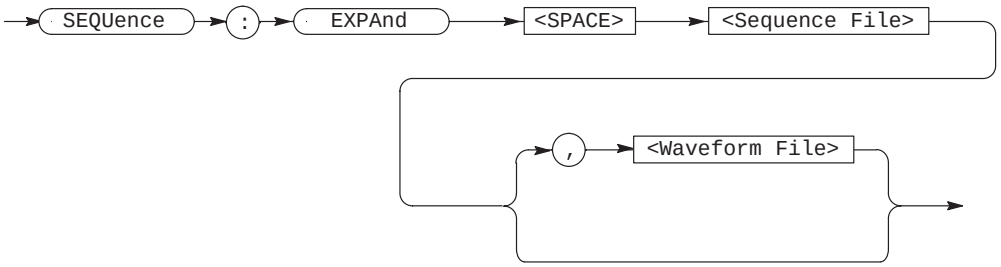
SEQUence:EXPAnd

The SEQUence:EXP and command expands the sequences recorded in the specified sequence file into waveform data and creates a waveform file.

Group WAVEFORM

Related Commands SEQUence:DEFine

Syntax SEQUence:EXPAnd <Sequence File>[,<Waveform File>]



Arguments

```
<Sequence File>::=<string>
<Waveform File>::=<string>
```

The sequence and the waveform files are files in internal memory. If the waveform file specification is omitted a waveform file with the same base name as the sequence file and the extension ”WFM” is created. An error is flagged if a waveform file with the same file name as the waveform file to be created already exists.

The number of waveform points in the expanded waveform file is the sum of the products of the iteration count and the number of points in each waveform file specified in the sequence.

Examples

In the following example the sequence file SQWAVE.SEQ is expanded into a waveform file. Here, the generated file is SQWAVE.WFM.

```
:SEQUENCE:EXPAND "SQWAVE.SEQ"
```

In the next example, the waveform file is created as the file SQWAVE01.WFM.

```
:SEQUENCE:EXPAND "SQWAVE.SEQ", "SQWAVE01.WFM"
```

*SRE (?)

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

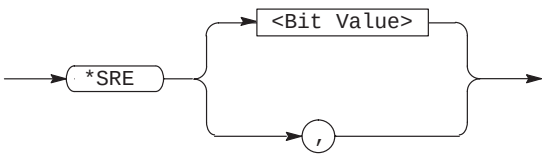
The power-on default for the SRER is all bits reset if the power-on status flag is TRUE. If this flag is set to FALSE, the SRER maintains its value through a power cycle.

Group STATUS and EVENT

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

Syntax

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



Arguments

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

Examples

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

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

STARt

The STARt command generates a trigger event to start the output of a waveform or a sequence.

Group MODE

Related Commands RUNNing?, STOP, *TRG

Syntax

STARt

Arguments None

Examples

:START
generates a trigger event.

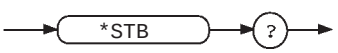
*STB?

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

Group STATUS and EVENT

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

Syntax *STB?



Arguments None

Responses <NR1>
which is a decimal number.

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

STOP

The STOP command terminates waveform output. When the mode is *not* set to continuous, it also resets the sequence pointer to output the waveform from the top of the sequence with next trigger event.

Group MODE

Related Commands RUNNing?, STARt, *TRG

Syntax STOP

Arguments None

Examples `:STOP`
stops the output of a waveform.

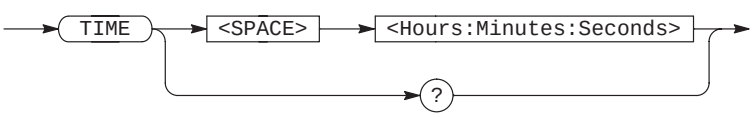
TIME (?)

The TIME command sets the time. The TIME? query returns the time.

Group SYSTEM

Related Commands DATE

Syntax
```
TIME <Hours:Minutes:Seconds>
TIME?
```



Arguments <Hours:Minutes:Seconds>::=<string>
where <string> is in the format "HH:MM:SS", with the elements given as follows.

HH	the hour in 24-hour format (0 to 23)
MM	the minutes (0 to 59)
SS	the seconds (0 to 59)

Examples `:TIME "11:23:58"`
sets the time.

*TRG

The *TRG common command generates trigger event. This command is equivalent to the STARt command.

Group MODE

Related Commands RUNNing?, STARt, STOP

Syntax *TRG

Arguments None

Examples *TRG
generates trigger event.

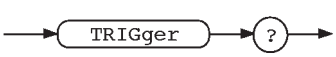
TRIGger?

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

Group MODE

Related Commands RUNNing?, STARt, STOP

Syntax TRIGger?



Arguments None

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

TRIGger:IMPedance (?) (AWG2020/21/40/41)

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

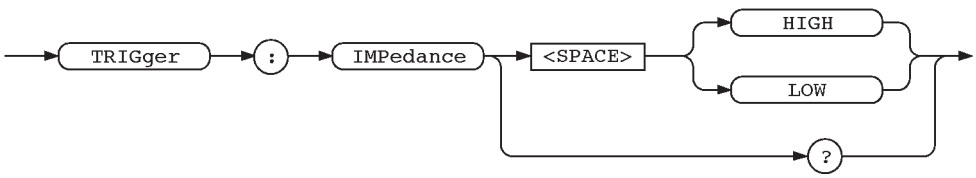
The TRIGger:IMPedance? query returns currently selected impedance.

Group MODE

Related Commands TRIGger:LEVel, TRIGger:POLarity, TRIGger:SLOPe

Syntax

```
TRIGger:IMPedance {HIGH (AWG2020/21/40/41) | LOW}
TRIGger:IMPedance?
```



Arguments

HIGH
selects high impedance: 1 MΩ

LOW
selects low impedance: 50 Ω

Examples

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

TRIGger:LEVel (?)

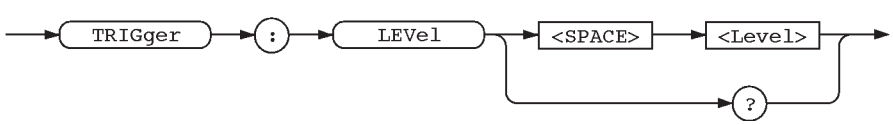
The TRIGger:LEVel command sets the level on the external trigger at which the trigger event is generated. The TRIGger:LEVel? query returns the level currently set.

Group MODE

Related Commands TRIGger:IMPedance, TRIGger:POLarity, TRIGger:SLOPe

Syntax

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



Arguments

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

Examples

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

TRIGger:POLarity (?)

The TRIGger:POLarity command selects polarity of the external trigger signal which generates the trigger event. The TRIGger:POLarity? query returns the currently selected polarity.

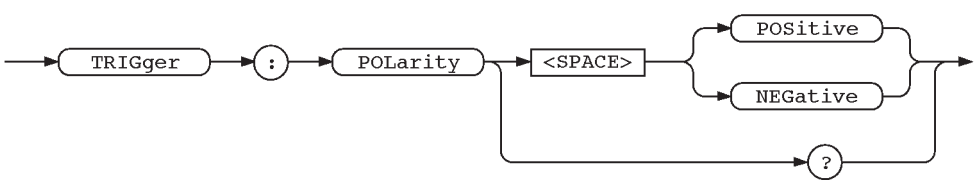
The polarity parameter is valid only when the mode is set to gated mode.

Group MODE

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

Syntax

```
TRIGger:POLarity {POSitive | NEGative}
TRIGger:POLarity?
```



Arguments

POSitive
selects positive polarity.

NEGative
selects negative polarity.

Examples

:TRIGGER:POLARITY NEGATIVE
selects negative polarity.

TRIGger:SLOpe (?)

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

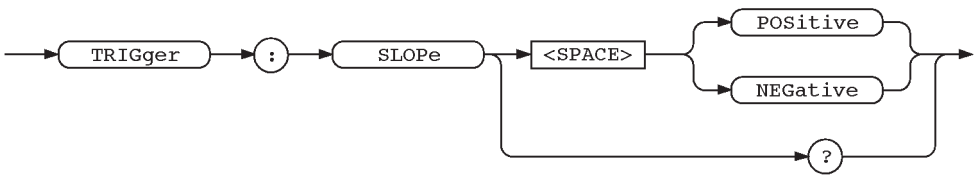
The slope parameter is valid only when the mode is set to other than gated or continuous mode.

Group MODE

Related Commands TRIGger:IMPedance, TRIGger:LEVel, TRIGger:POLarity

Syntax

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



Arguments

POSitive
selects rising edge.

NEGative
selects falling edge.

Examples

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

*TST?

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

Group

CALIBRATION and DIAGNOSTIC

Related Commands

DIAG:SELect, DIAG:STATe, DIAG:RESUlt?

Syntax

```
*TST?
```

Arguments

None

Responses

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

0	Terminated without error.
100	Detected an error in the CPU unit.

200 Detected an error in the clock unit.
300 Detected an error in the display unit.
400 Detected an error in the floating point processor unit.
500 Detected an error in the front panel unit.
600 Detected an error in the setup-related unit.
700 Detected an error in the waveform memory.
800 Detected an error in the trigger unit.

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

UNLock

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

Group SYSTEM

Related Commands

Syntax UNLOCK ALL

Arguments ALL
enables the front panel buttons and knob.

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

UPTime?

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

Group SYSTEM

Related Commands None

Syntax UPTime?

Arguments None

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

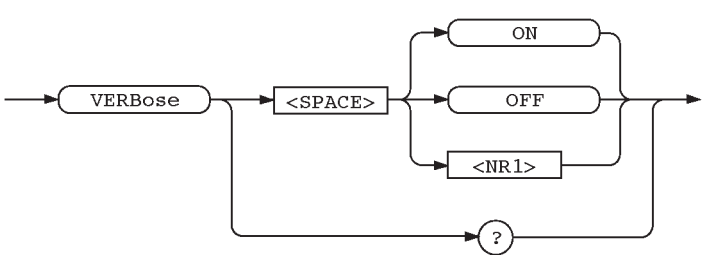
VERBose (?)

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

Group SYSTEM

Related Commands HEADer

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



Arguments ON or nonzero value
selects long response header.

OFF or zero value
selects short response header.

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

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

Examples

`:VERBOSE ON`
sets long header for query responses.

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

*WAI

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

Group SYNCHRONIZATION

Related Commands `*OPC`

Syntax `*WAI`

Arguments None

Responses None

Examples

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

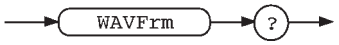
WAVFrm?

The `WAVFrm?` query transmits waveform preamble and waveform. This query is equivalent to the `WFMPre?` query, followed by the `CURVe?` query.

Group WAVEFORM

Related Commands `CURVe?`, `DATA:SOURce`, `DATA:ENCDG`, `WFMPre?`

Syntax `WAVFrm?`



Arguments None

Responses Returns the settings as a sequence of commands, suitable for sending as set commands to restore a setup (see Examples).

Examples
```
:WAVFRM?
```
might return the following response.
```
:WFMPRE:ENCDG BIN;BN_FMT RP;BYT_NR 2;BIT_NR 12;BYT_OR MSB;CRVCHK
NONE;WFID "WAVEFORM.WFM, 1000 points, clock: 100.0MHz, amplitude:
1.000V, offset: 0.000V";NR_PT 1000;PT_FMT Y;XUNIT "S"; XINCR
1.0000E-08;PT_OFF 0;XZERO 0.000;YUNIT "V";YMULT 2.442E-04; YZERO
0.000;YOFF 2.047E+03;:CURVE #42000<DAB><DAB> ... <DAB>
```

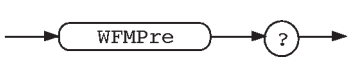
WFMPre?

The WFMPre? query returns all settings for the waveform preamble. The preamble information refered to by this query are for the information for DATA:SOURce (waveform source).

Group WAVEFORM

Related Commands All WFMPRE subgroup commands, DATA:SOURce

Syntax
```
WFMPre?
```



Arguments None

Responses Returns the settings as a sequence of commands, suitable for sending as set commands to restore a setup (see Examples).

Examples
```
:WFMPRE?
```
might return as follows:
```
:WFMPRE:ENCDG BIN;BN_FMT RP;BYT_NR 2;BIT_NR 12;BYT_OR MSB;CRVCHK
NONE;WFID "WAVEFORM.WFM, 1000 points, clock: 100.0MHz, amplitude:
1.000V, offset: 0.000V";NR_PT 1000;PT_FMT Y;XUNIT "S"; XINCR
```

```
1.0000E-08;PT_OFF 0;XZERO 0.000;YUNIT "V";YMULT 2.442E-04; YZERO
0.000;YOFF 2.047E+03
```

WFMPre:BIT_NR (?)

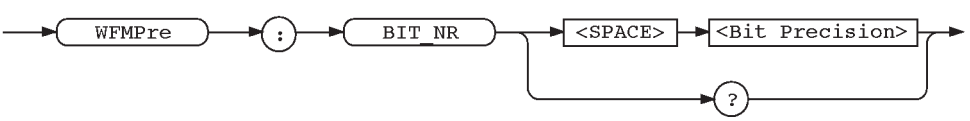
The WFMPre:BIT_NR command specifies the number of bits of precision for each binary data point. The WFMPre:BIT_NR? query returns the bits of precision currently specified.

Group WAVEFORM

Related Commands WFMPre:BN_FMT, WFMPre:BYT_NR, WFMPre:PT_FMT, WFMPre:BYT_OR, WFMPre:ENCDG, DATA:ENCDG, DATA:WIDTh

Syntax

```
WFMPre:BIT_NR <Bit Precision>
WFMPre:BIT_NR?
```



Arguments <Bit Precision>::=<NR1>
where the bit precision must be set to 8 for the 1 byte data width, while 12 for the 2 byte. Any argument other than 8 or 12 (default) is ignored.

Examples :WFMPRE:BIT_NR?
might return :WFMPRE:BIT_NR 12.

WFMPre:BN_FMT (?)

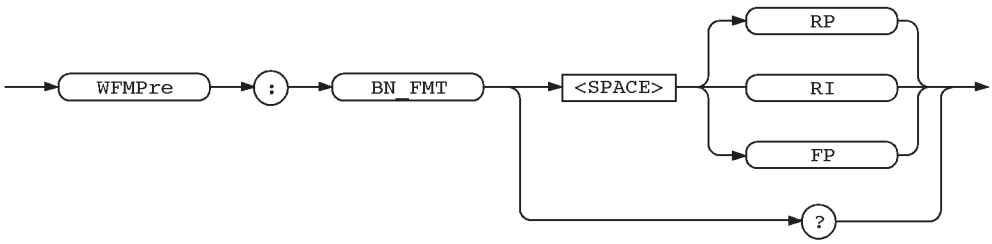
The WFMPre:BN_FMT command specifies format of binary data. The WFMPre:BN_FMT? query returns the binary data format currently specified.

Group WAVEFORM

Related Commands WFMPre:BYT_NR, WFMPre:BIT_NR, WFMPre:BYT_OR, WFMPre:ENCDG, DATA:ENCDG

Syntax

```
WFMPre:BN_FMT {RP | RI | FP}
WFMPre:BN_FMT?
```



Arguments

RP
binary unsigned integer code.

RI
binary integer code.

FP
single precision binary floating code.

The choice other than the RP (default) is ignored on input in this argument.

Examples

:WFMPRE:BN_FMT?
might return :WFMPRE:BN_FMT RP

WFMPre:BYT_NR (?)

The WFMPre:BYT_NR command specifies data field width (byte length) for each binary data point. The WFMPre:BYT_NR? query returns the data field width currently specified.

Group

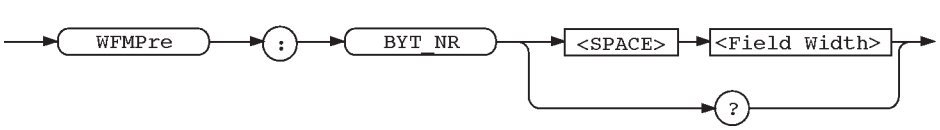
WAVEFORM

Related Commands

WFMPre:BN_FMT, WFMPre:BIT_NR, WFMPre:BYT_OR, WFMPre:ENCDG, DATA:ENCDG, DATA:WIDTh

Syntax

WFMPre:BYT_NR <Field Width>
WFMPre:BYT_NR?



Arguments

<Field Width>::=<NR1>
The field width must be 2 or 1. When transferring the data, the data width can

also be specified using the DATA:WIDTh command. When both this command and the DATA:WIDTh command are used, the most recently issued, i.e., the last, command takes effect. For example, if the byte width is set to 1 using this command (WFMPre:BYT_NR1), and then a DATA:WIDTh2 command is executed, the setting will be changed so that the data width of 2 bytes is transmitted.

Examples

WFMPRE:BYT_NR
might return :WFMPRE:BYT_NR 2.

WFMPre:BYT_OR (?)

The WFMPre:BYT_OR command specifies which byte of the binary data is send first when the data field width of the binary data is defined to be 2-byte. The WFMPre:BYT_OR? query returns the binary data byte order currently specified.

Group

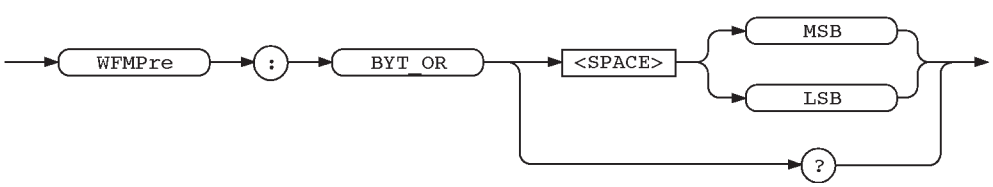
WAVEFORM

Related Commands

WFMPre:BN_FMT, WFMPre:BYT_NR, WFMPre:BIT_NR, WFMPre:ENCDG, DATA:ENCDG, DATA:WIDTh

Syntax

WFMPre:BYT_OR {MSB | LSB}
WFMPre:BYT_OR?



Arguments

MSB
sends upper byte first, then lower byte for each data word.

LSB
sends lower byte first, then upper byte for each data word.

The data transfer time byte order can also be specified using the DATA:ENCDG command. When both this command and the DATA:ENCDG command are used, the most recently issued, i.e., the last, command takes effect. For example, if the byte order is set to low order byte first using this command (WFMPre:BYT_OR LSB), and then a DATA:ENCDG RPBinary command is executed, the setting will be changed so that the high order byte is transmitted first.

Examples

```
:WFMPRE:BYT_OR?
```
might return `:WFMPRE:BYT_OR MSB`.

WFMPre:CRVCHK (?)

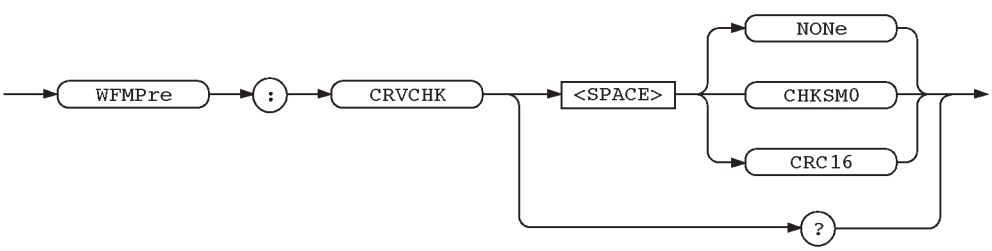
The `WFMPre:CRVCHK` command specifies the error check method for binary data. The `WFMPre:CRVCHK?` query returns the error check method currently in effect.

Group WAVEFORM

Related Commands `WFRPre:ENCDG`, `DATA:ENCDG`

Syntax

```
WFMPre:CRVCHK {NONe | CHKSM0 | CRC16}
WFMPre:CRVCHK?
```



Arguments While the following arguments may be used, all arguments except for `NONe` (default) is ignored.

`NONe`
no error checking. All binary block data represent data.

`CHKSM0`
last byte of the binary data is a checksum defined as the two's complement of the modulo 256 sum of the preceding binary data bytes and ASCII count bytes.

`CRC16`
last two bytes represent the 16-bit cyclic redundancy check code.

Examples

```
:WFMPRE:CRVCHK?
```
might return `:WFMPRE:CRVCHK NONE`.

WFMPre:ENCDG (?)

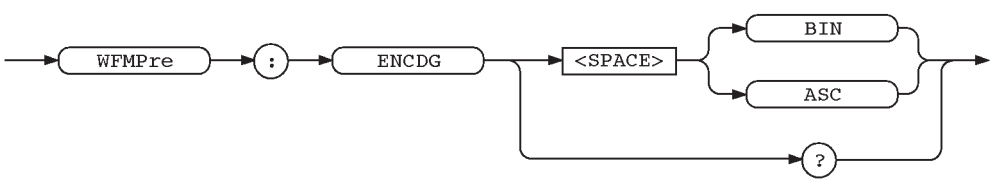
The WFMPre:ENCDG command sets the encoding type for the waveform transmitted with the CURVe command. The WFMPre:ENCDG? query returns the encoding type currently set.

Group WAVEFORM

Related Commands DATA:ENCDG

Syntax

```
WFMPre:ENCDG {BIN | ASC}
WFMPre:ENCDG?
```



Arguments While the following arguments may be used, any arguments except for BIN (default) is ignored. The choice other than the BIN (default) is ignored on input in this argument.

BIN
specifies binary encoding type.

ASC
specifies ASCII encoding type.

Examples

```
:WFMPre:ENCDG?
```

might return :WFMPRE:ENCDG BIN.

WFMPre:NR_PT (?)

The WFMPre:NR_PT command sets the size of the waveform in terms of sets of points. The WFMPre:NR_PT? query returns the waveform size currently set.

Group WAVEFORM

Related Commands DATA:SOURce, DATA:DESTination

Syntax

```
WFMPre:NR_PT <Data Size>
WFMPre:NR_PT?
```

Arguments

<Data Size>::=<NR1>
where <NR1> is ignored. The waveform generator sets the size of the waveform automatically and, therefore, ignores any value entered for <Data Size>.)

Examples

WFMPre:NR_PT?
might return :WFMPRE:NR_PT 131072.

WFMPre:PT_FMT (?)

The WFMPre:PT_FMT command selects the data point format of the waveform. The WFMPre:PT_FMT? query returns the data point format currently selected.

Group

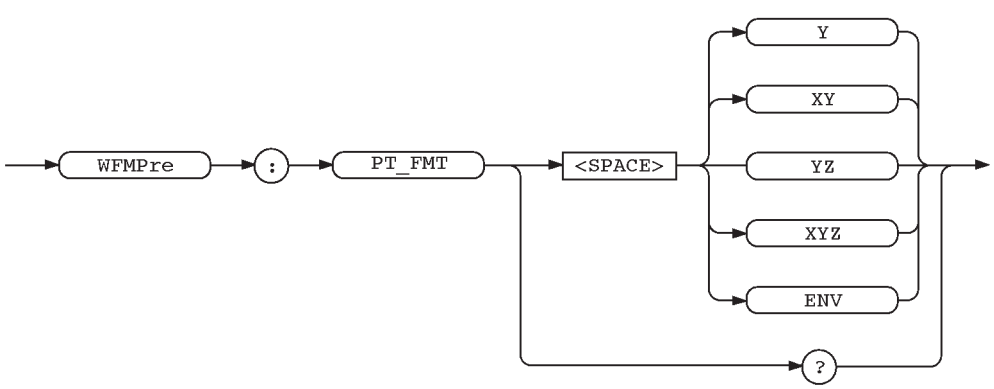
WAVEFORM

Related Commands

WFMPre:PT_OFF, WFMPre:XINCR, WFMPre:XMULT, WFMPre:XZERO, WFMPre:XOFF, WFMPre:YMULT, WFMPre:YZERO, WFMPre:YOFF

Syntax

```
WFMPre:PT_FMT {Y | XY | YZ | XYZ | ENV}
WFMPre:PT_FMT?
```



Arguments

While any of the following arguments may be transmitted to the waveform generator, it only recognizes the Y (default) argument. All others are ignored.

Y
explicitly transmits Y values, absolute X and Y component values are calculated for each data point using the the transmission sequence yn, yn+1, yn+2 ...
where
Xn = <XZERO-value> + <XINCR-value> (n – <PT_OFF-value>)
and
Yn = <YZERO-value> + <YMULT-value> (yn – <YOFF-value>).

XY
explicitly transmits XY values.

YZ
explicitly transmits YZ values.

XYZ
explicitly transmits XYZ values.

ENV
transmits two y values for each point: maximum and minimum.

Examples

:WFMPRE:PT_FMT?
might return :WFMPRE:PT_FMT Y.

WFMPre:PT_OFF (?)

The WFMPre:PT_OFF command defines the X axis point offset value. The WFMPre:PT_OFF? query returns the X axis point offset value currently set.

Group

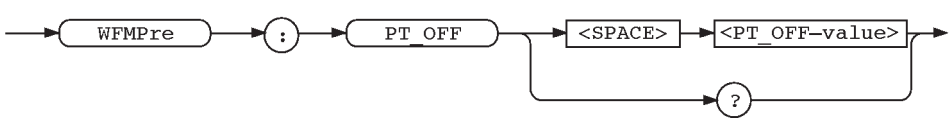
WAVEFORM

Related Commands

WFMPre:PT_FMT, WFMPre:XINCR, WFMPre:XZERO

Syntax

WFMPre:PT_OFF <PT_OFF-value>
WFMPre:PT_OFF?



Arguments

<PT_OFF–value>::=<NR1>
where <NR1> is a decimal integer. The waveform generator ignores all input for <NR1> except for zero, the default.

Examples `:WFMPRE:PT_OFF?`
might return `:WFMPRE:PT_OFF 0`.

WFMPre:XINCR (?)

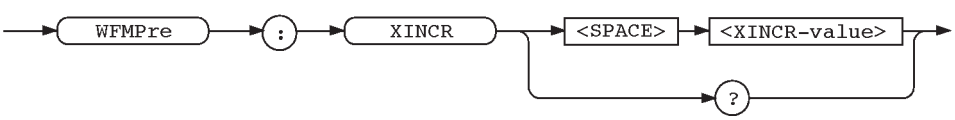
The `WFMPre:XINCR` command defines the X axis increment value. The `WFMPre:XINCR?` query returns the X axis increment value.

This increment value is effective for the destination of the waveform file defined by `DATA:DESTination` command.

Group WAVEFORM

Related Commands `WFMPre:PT_FMT`, `WFMPre:PT_OFF`, `WFMPre:XZERO`

Syntax
```
WFMPre:XINCR <XINCR-value>
WFMPre:XINCR?
```



Arguments `<XINCR-value>::=<NR3>`
where `<NR3>` is a decimal number that ranges from 5E–8 seconds to 1E–1 seconds (AWG2005), 4E–9 seconds to 1E–1 seconds (AWG2020/21), 9.7646E–10 seconds to 1E–3 seconds (AWG2040/41).

Examples `:WFMPRE:XINCR 0.01`
sets the X axis increment value to 0.01 second.

WFMPre:XUNIT (?)

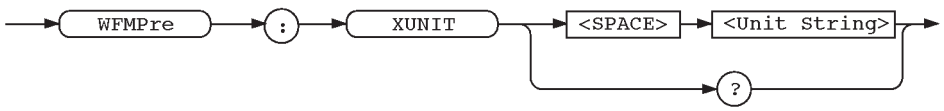
The `WFMPre:XUNIT` command defines the appropriate representation of the data unit for the X axis. The `WFMPre:XUNIT?` query returns the representation for the X axis data unit currently defined.

Group WAVEFORM

Related Commands `WFMPre:PT_OFF`, `WFMPRe:XINCR`, `WFMPre:XZERO`

Syntax

```
WFMPre:XUNIT <Unit String>
WFMPre:XUNIT?
```



Arguments

<Unit String>::=<string>
where <string> is either the default S, for second, or is ignored by the waveform generator.

Examples

```
:WFMPRE:XUNIT?
```
might return `:WFMPRE:XUNIT "S"`.

WFMPre:XZERO (?)

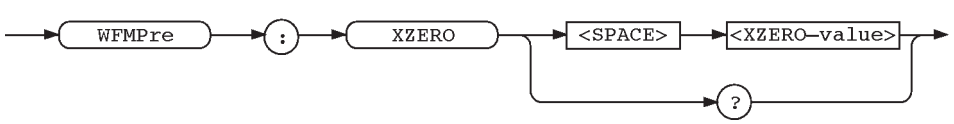
The WFMPre:XZERO command defines the X axis origin value.
The WFMPre:XZERO? query returns the X axis origin value currently defined.

Group WAVEFORM

Related Commands WFMPre:PT_OFF, WFMPre:XUNIT, WFMPre:XINCR

Syntax

```
WFMPre:XZERO <XZERO-value>
WFMPre:XZERO?
```



Arguments

<XZERO-value>::=<NR2>
where <NR1> is either the default value 0.0, or is ignored by the waveform generator.

Examples

```
:WFMPRE:XZERO?
```
might return `:WFMPRE:PT_OFF 0.0`.

WFMPre:YMULT (?)

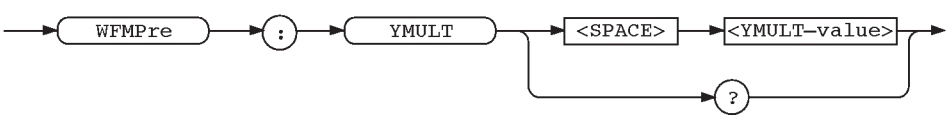
The WFMPre:YMULT command defines multiplier value of the data for the Y axis. The WFMPre:YMULT? query returns the Y axis multiplier value currently defined.

This value is effective for destination of the waveform file defined by DATA:DESTination command. And referring to the multiplier value is performed for the source of the waveform file defined by DATA:SOURce command.

Group WAVEFORM

Related Commands WFMPre:YOFF, WFMPre:YZERO, WFMPre:YUNIT, DATA:DESTination, DATA:SOURce

Syntax
```
WFMPre:YMULT <YMULT-value>
WFMPre:YMULT?
```



Arguments <YMULT-value>::=<NR3>

Examples
```
:WFMPRE:YMULT 0.0012
```
sets the multiplier value to 0.0012 V.

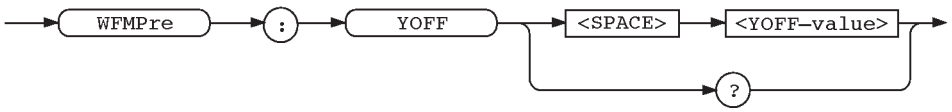
WFMPre:YOFF (?)

The WFMPre:YOFF command defines the Y axis offset value.
The WFMPre:YOFF? query returns the Y axis offset value currently defined.

Group WAVEFORM

Related Commands WFMPre:YMULT, WFMPre:YZERO, WFMPre:YUNIT

Syntax
```
WFMPre:YOFF <YOFF-value>
WFMPre:YOFF?
```



Arguments `<YOFF-value>::=<NR3>`
where `<NR3>` is either the default value 127 in 1 byte data width or 2047 in 2 byte data width, or is ignored by the waveform generator.

Examples `:WFMPRE:YOFF?`
might return `:WFMPRE:YOFF 2.047E+03`

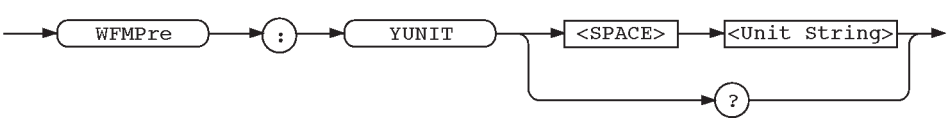
WFMPre:YUNIT (?)

The `WFMPre:YUNIT` command defines the appropriate representation of the data unit for the Y axis. The `WFMPre:YUNIT?` query returns the representation for the Y axis data unit currently defined.

Group WAVEFORM

Related Commands `WFMPre:YMULT`, `WFMPre:YZERO`, `WFMPre:YOFF`

Syntax
```
WFMPre:YUNIT <Unit String>
WFMPre:YUNIT?
```



Arguments `<Unit String>::=<string>`
where `<string>` is either the default V for voltage or is ignored by the waveform generator.

Examples `:WFMPRE:YUNIT?`
might return `:WFMPRE:YUNIT "V"`.

WFMPre:YZERO (?)

The WFMPre:YZERO command defines the Y axis origin value.
The WFMPre:YZERO? query returns the Y axis origin value currently defined.

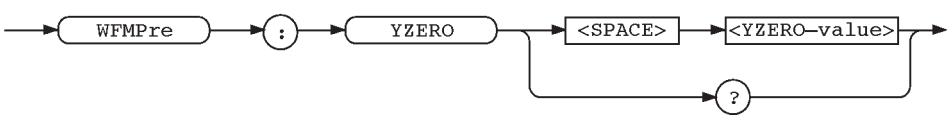
This value is effective for the destination of the waveform file defined by DATA:DESTination command. And referring to the origin value is performed for the source of the waveform file defined by DATA:SOURce command.

Group WAVEFORM

Related Commands WFMPre:PT_OFF, WFMPre:YMULT, WFMPre:YUNIT, WFMPre:YOFF, DATA:DESTination, DATA:SOURce

Syntax

```
WFMPre:YZERO <YZERO-value>
WFMPre:YZERO?
```



Arguments <YZERO-value>::=<NR2>
where <NR2> is a decimal number that ranges from –5.000 to 5.000 in steps 0.005 (AWG2005), –2.500 to 2.500 in steps 0.005 (AWG2020/21), and –1.000 to 1.000 in steps 0.001 (AWG2040/41). The unit volts is assumed.

Examples :WFMPRE:YZERO 0.225
sets the Y axis origin value to 0.225 V.

WFMPre:WFID (?)

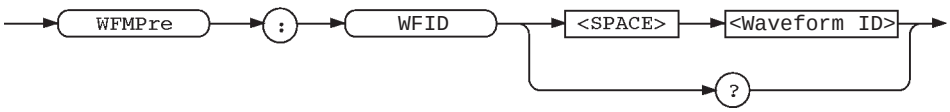
The WFMPre:WFID command sets comment and/or additional information as a waveform ID for the waveform preamble.

Group WAVEFORM

Related Commands

Syntax

```
WFMPre:WFID <Waveform ID>
WFMPre:WFID?
```



Arguments

<Waveform ID> is automatically set by the waveform generator, and arguments are ignored on input.

Examples

```
:WFMPRE:WFID?
```

might return the following response.

```
:WFMPRE:WFID "WAVEFORM.WFM, 1000 points, clock: 100.0MHz,ampli-
tude: 1.000V, offset: 0.000V"
```

Retrieving Response Messages

The method used for retrieving response messages differs depending on whether a GPIB interface or an RS-232-C interface is used. Figures 2-4 and 2-5 give an overview of these methods.

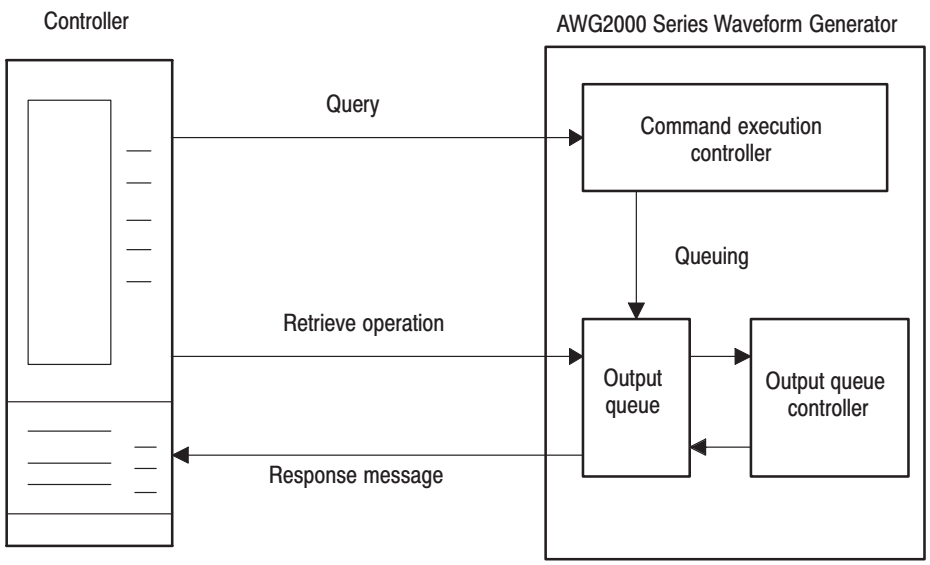


Figure 2-4: GPIB: Retrieving Response Messages

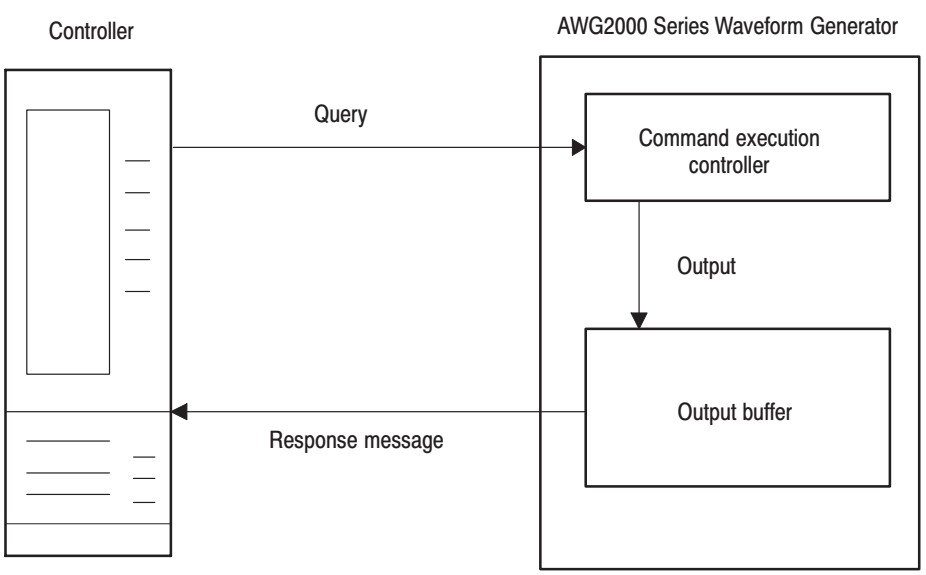


Figure 2-5: RS-232-C: Retrieving Response Messages

Figure 2-4 shows the response message retrieval operation when a GPIB interface is used. When a query command is sent from the external controller the waveform generator puts the response message for the query on the output queue. This response message cannot be retrieved unless the user performs a retrieval operation through the external controller. The response message retrieval operation is performed using the awgRead() support function in the programming examples in Section 4. See “Example 4” and “Support Functions” in Section 4 for more information on this retrieval operation.

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

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

Figure 2-5 shows the response message retrieval operation when an RS-232-C interface is used. When a query command is sent from the external controller, the waveform generator immediately sends the response message to the external controller through an output buffer. As a result, when either a dumb terminal or a terminal emulator program running on a PC is used as the external controller, the response message will be displayed on the CRT immediately after the query command is typed in.

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

Waveform Transfer

The waveform transfer function transfers waveforms between the waveform generator and an external controller. This function can be used to store waveforms created by the waveform generator in the external controller so that those waveforms can then be transferred to another unit, or to return to the waveform generator modified waveforms or waveforms that were created on the external controller.

Waveform transfer is performed under the Tektronix Std. Codes and Formats waveform format specifications. The following part describes the waveform transfer method between these waveform generators and external controllers.

These waveform generators are also equipped with direct waveform transfer functions to transfer waveforms directly with Tektronix digital oscilloscopes and other units using a GPIB interface. See the user manual for each waveform generator for details on the use of these functions.

Note that these waveform generators can also transfer equations and marker data with an external controller. See the EQUAtion:DEFine command description for details on equation transfer, and the MARKer:DATA command description for details on marker data transfer.

Source and Destination

The source and destination are specified prior to waveform transfer.

"Source" refers to the waveform transfer source when waveforms or marker data are transferred from the waveform generator to the external controller. Waveforms and marker data that the waveform generator can transfer to external equipment are limited to data loaded in waveform memory and data that is stored in waveform files in internal memory. Use the DATA:SOURce command to specify the source.

"Destination" refers to the destination for the waveform transfer when waveforms or marker data are transferred from the external controller to the waveform generator. The transfer destination must be a waveform file in internal memory. If the specified waveform file is not in internal memory, a new file is created. On the other hand if that file already exists it will be overwritten. Use the DATA:DESTination command to specify the destination.

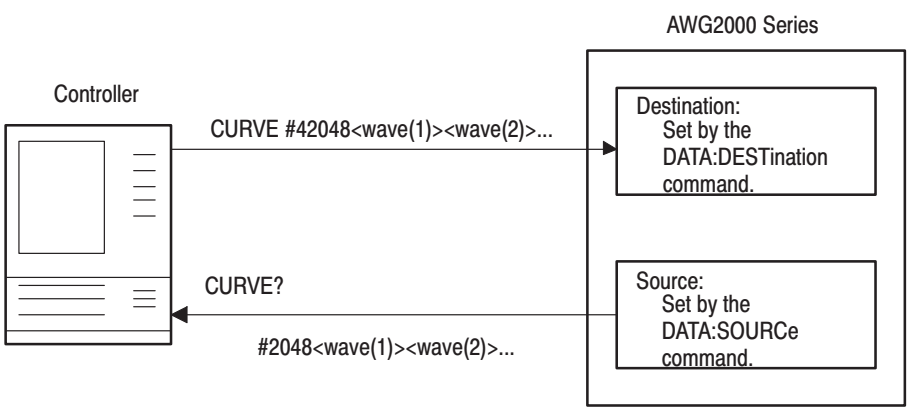


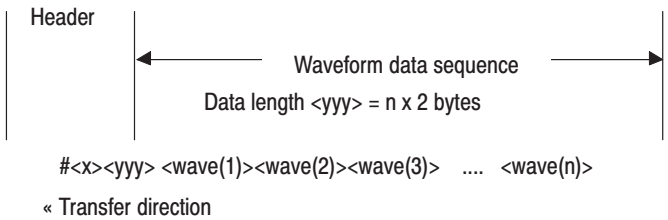
Figure 2-6: Source and Destination

Preamble and Curve

A transferred waveform consists of a preamble and a curve. The preamble consists of data including the size, scale, and format of the curve data, and supplementary data such as the waveform ID and units. The curve expresses the data that is to be stored in waveform memory as a sequence of unscaled waveform data. Complete scaled data can be derived from this unscaled data and the data in the preamble.

A curve is sent to the external controller from the waveform generator by the CURVE? query command as an arbitrary block format response message as shown below. (Note that the response header is turned off in this case.) Inversely,

unscaled waveform data can be transferred from the external controller to the waveform generator by specifying an arbitrary block in the format shown below as the argument to the CURVE command.



Here <yyy> is the byte count (in ASCII format) of the waveform data sequence that follows, <x> is the number of digits in <yyy> (in ASCII format), and <wave(i)> is the ith waveform datum. The ith data point (X(i), Y(i)) is converted to scaled waveform data according to the following formulas.

X(i)= i _<XINCR-value>
Y(i)= <YZERO-value>+(<wave(i)>–<YOFF–value>)_ <YMULT-value>

The <XINCR-value>, <YZERO-value>, <YOFF-value>, and <YMULT-value> are data that is included in the preamble as shown in the table below. When reading out waveform data this data can be retrieved from the waveform generator by using query commands. Inversely, when transferring waveform data to the waveform generator, this data can be set in the waveform generator using commands. However, note that the <YOFF-value> can only be set to 2047 when the data width is two or 127 when the data width is one.

Parameter	Command	Meaning
<XINCR-value>	WFMPre : XINCR	X-axis data point increment
<YZERO-value>	WFMPre : YZERO	Y-axis origin offset
<YOFF-value>	WFMPre : YOFF	Y-axis data point offset (2047 or 127)
<YMULT-value>	WFMPre : YMULT	Y-axis data point multiplier

Each data point <wave(i)> is transferred as an unsigned integer code of two bytes with 12 valid data bits (when the data width is two bytes or one byte with eight valid data bits when the data width is one byte. When data is transferred in the two byte width, the byte order (which of the upper and lower bytes is transferred first) can be specified using either the WFMPre:BYT_OR command or the DATA:ENCDG command.

Byte order specification allows data to be stored more easily in memory by specifying the appropriate order depending on whether the external controller CPU uses a Little-Endian or Big-Endian addressing scheme. For example, if an

NEC PC-9800 series or an IBM-PC compatible is used as the external controller, set data to be transferred with the low order byte first. See the detailed command descriptions for more information.

The X-axis and Y-axis are represented as time (S) and voltage (V) respectively. Note that the transferred waveform data format and related information (preamble) can be set and queried by the commands in the WAVEFORM command groups that have WFMPre as their root header mnemonic.

Data Transfer Procedures

The following two sections show examples of procedures for transferring waveforms from the waveform generator to an external controller and from an external controller to the waveform generator.

Transfer from the Waveform Generator to an External Controller.

1. Specify the source.

DATA : SOURCE “CH1”

This command specifies the waveform loaded in channel one. The following command specifies a waveform file.

DATA : SOURCE “SAMPLE-1.WFM”

2. Specify the waveform data points, data width, and byte order. The following command specifies a data width of two and that the low order byte be transferred first.

DATA : WIDTH 2 ; ENCDG SRPBINARY

Use RPBINARY in place of SRPBINARY to specify high order byte first transfers. This specification can also be performed using the WFMPRE:BYT_OR command.

3. Turn off the response header.

HEADER OFF

4. Read in the next preamble data, convert it to binary format, and then store it in memory. (See step 7.)

a. Read the number of data items <NR_PT>.

WFMPRE : NR_PT?

b. Read the Y-axis origin offset <YZERO-value>.

WFMPRE : YZERO?

c. Read the Y-axis data point multiplier <YMULT-value>.

WFMPRE : YMULT?

d. Read the X-axis data point increment value <XINCR-value>.

WFMPRE : XINCR?

5. Specify the start of the waveform data transfer.

CURVE?

6. Read the waveform data from the output queue. Note that when an RS-232-C interface is used the waveform data will be transferred immediately since there is no output queue.

a. Read the arbitrary block header section.

b. Read the waveform data into an array. Since the waveform data consists of <NR_PT> data items, a one dimensional array of <NR_PT> items each the size of the data width will be required as data memory.

When the data width setting is two, the byte order of the data points must be determined according to the CPU used in the external controller. However, we recommend using the technique in which data is read one byte at a time and then reconstructed into two byte objects to avoid being dependent on the CPU type.

7. Convert the data to scaled waveform data. Convert the ith data point (X(i), Y(i)) according to the following formulas. Note that wave(i) is the ith element in the unscaled waveform data.

X(i)= i _<XINCR-value>
Y(i)=(wave(i)-<YOFF-value>)_<YMULT-value>

8. Restore the response header state to on.

HEADER ON

This completes the transfer of a waveform file from the waveform generator to the external controller.

Transfer from an External Controller to the Waveform Generator.

1. Set the destination.

DATA : DESTINATION “SAMPLE-1.WFM”

This command specifies the waveform file “SAMPLE-1.WFM” in internal memory as the destination.

2. Specify the data width and byte order for the waveform data points.

DATA: WIDTH 2 ; ENCDG SRPBINARY

This command specifies transfer with a data width of 2 and with the low order byte first. To transfer the high order byte first specify RPBINARY in stead of SRPBINARY. The WFMPRE:BYT_OR command can also be used for this specification.

3. Set up the preamble data.

a. Set the Y-axis origin offset <YZERO-value>.

Example: WFMPRE : YZERO 0.0

b. Set the Y-axis data point multiplier <YMULT-value>.

Example: WFMPRE : YMULT 4.8E–04

c. Set the X-axis data point increment <XINCR-value>.

Example: WFMPRE : XINCR 5.0E–9

A default value will be used for any preamble data that is not set. (If a waveform file that exists in internal memory is specified when setting the source in step 1, the preamble data recorded in that file will be used as the default values.)

4. Transfer the waveform data.

CURVE #42048<wave(1)><wave(2) ... <wave(1024)>

This completes the transfer of a waveform file from the external controller to the waveform generator.

Status and Events

Status and Event Reporting

This section describes how the AWG2000 Series Arbitrary Waveform Generator reports its status and internal events for both the GPIB and RS-232-C interfaces. It describes the elements that comprise the status and events reporting system and explains how status and events are handled.

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

Registers

The registers fall into two functional groups:

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

Status Registers

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

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

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

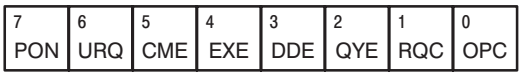


Figure 3-1: The Standard Event Status (SESR)

Table 3-1: SESR Bit Functions

Bit	Function
7 (MSB)	**PON** (Power On). Indicates that the waveform generator was powered on.
6	**URQ** (User Request). Indicates an event occurred and because of that event the waveform generator needs attention from the operator.
5	**CME** (Command Error). Indicates that an error occurred while the waveform generator was parsing a command or query. Command error messages are listed in Table 3-5 on page 3-10.
4	**EXE** (Execution Error). Indicates that an error occurred while the waveform generator was executing a command or query. An execution error occurs for either of the following reasons: ■ A value designated for the argument is out of the range allowed by the waveform generator, is not valid for the command, or is incorrect in some other sense. ■ Execution took place improperly under conditions different from those which should have been requested. Execution error messages are listed in Table 3-6 on page 3-12.
3	**DDE** (Device Dependent Error). Indicates that a device-specific error occurred. Device error messages are listed in Table 3-7 on page 3-13.
2	**QYE** (Query Error). Indicates that an error occurred upon attempting to read the output queue. Such an error occurs for one of the following two reasons. ■ An attempt was made to retrieve a message from the output queue even through it is empty or pending. ■ Output queue message was cleared while it was being retrieved from the output queue.
1	**RQC** (Request Control).The waveform generator does not use this bit. Request Control (RQC) is used to show that an instrument has requested to transfer bus control back to the controller. (This is the usage prescribed by the IEEE Std. 488.1.)
0 (LSB)	**OPC** (Operation Complete). Indicates that the operation is complete. This bit is set when all pending operations complete following a `*OPC` command.

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

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

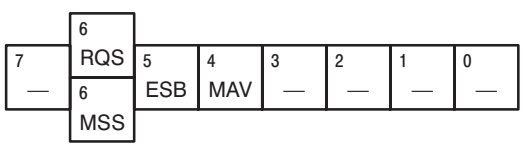


Figure 3-2: The Status Byte Register (SBR)

Table 3-2: SBR Bit Functions

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

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

Enable Registers

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

Each bit in an Enable Register corresponds to a bit in the Status Register it controls. In order for an event to be reported to its bit in the Status Register, the corresponding bit in the Enable Register must be set to one. If the bit in the Enable Register is set to zero, the event is not recorded.

Various commands set the bits in the Enable Registers. The Enable Registers and the commands used to set them are described below.

The Device Event Status Enable Register (DESER). Shown in Figure 3-3. This register controls which events of those shown are reported to the SESR and the Event Queue. The bits in the DESER correspond to those in the SESR, as was described earlier.

Use the DESE command to enable and disable the bits in the DESER. Use the DESE? query to read the DESER.

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

Figure 3-3: The Device Event Status Enable Register (DESER)

The Event Status Enable Register (ESER). Shown in Figure 3-4. It controls which events of those shown are allowed to be summarized by the Event Status Bit (ESB) in the SBR.

Use the *ESE command to set the bits in the ESER. Use the *ESE? query to read it.

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

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

The Service Request Enable Resgister (SRER). Shown in Figure 3-5. It controls which bits in the SBR generate a Service Request and are summarized by the Master Status Summary (MSS) bit.

Use the *SRE command to set the SRER. Use the *SRE? query to read it. The RQS bit remains set to one until either the Status Byte Register is read with a Serial Poll or the MSS bit changes back to a zero.

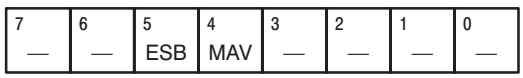


Figure 3-5: The Service Request Enable Register (SRER)

Queues

The status and event reporting system contains two queues, the Event Queue and the Output Queue. The Event Queue which is used when operating with either the GPIB and RS-232-C interface, while the Output Queue is used only when operating over the GPIB interface. (Instead of using an output queue, an output buffer buffers query-response messages for immediate transfer to the data transmission line for RS-232-C operation.)

Output Queue

The Output Queue is a FIFO (First In First Out) queue that hold response messages while until they are requested. When a message is put in the queue, the MAV bit of the Status Byte Register (SBR) is set.

The Output Queue empties each time the waveform generator receives a new command or query. Therefore the controller must read the output queue before it sends the next command or query command or it will lose responses to earlier queries. If a command or query command is given without taking it out, an error results and the Output Queue is emptied.

Event Queue

The Event Queue is a FIFO queue which can hold up to 20 waveform generator-generated events. When the number of events exceeds 20, the 20th event is replaced by the event code 350, “Queue overflow”.

To read out from the Event Queue, do the following steps.

1. Send *ESR? To read out the contents of SESR. When the contents of SESR are read out, SESR is cleared allowing you to take out events from the Event Queue.
2. Send one of the following queries:
   - ALLEv? To read out and returns all events made available by *ESR?. Returns both the event code and message text.
   - EVENT? To read out and return the oldest event of those made available by *ESR?. Returns only the event code.

- EVMsg? To read out and return the oldest event of those made available by *ESR?. Returns both the event code and message text.

Reading the SESR erases any events that were made available by previous *ESR? reads, but that were not read from the Event Queue. Events that occur after an *ESR? read are put in the Event Queue but are not available until *ESR? is used again.

Processing Sequence

Figure 3-6 shows the status and event processing flow.

1. An event occurs, which causes the DESR to be checked. Based on the state of the DESR, the following actions occur:

- If the control bit for that event is set in the DESER, the SESR bit that corresponds to this event becomes set to 1.
- The set control bit lets the event be placed into the Event Queue. Placing the event in the Event Queue sets the MAV bit in the SBR to one.
- If the control bit for that event is also set in the ESER, the ESB bit of SBR becomes set also.

2. When either bit of SBR has been set to 1 and the corresponding control bit of SRER is also set, the MSS bit of SBR becomes set and a service request is generated for use with GPIB interface operation.

As noted earlier, the RS-232-C interface does not use the output queue; therefore, the MAV bit would not become set in the sequence just described. Rather, response messages are sent to the output buffer for immediately transfer to the external controller on the output line. Message transfer is automatic and it is not necessary to use commands to retrieve these messages.

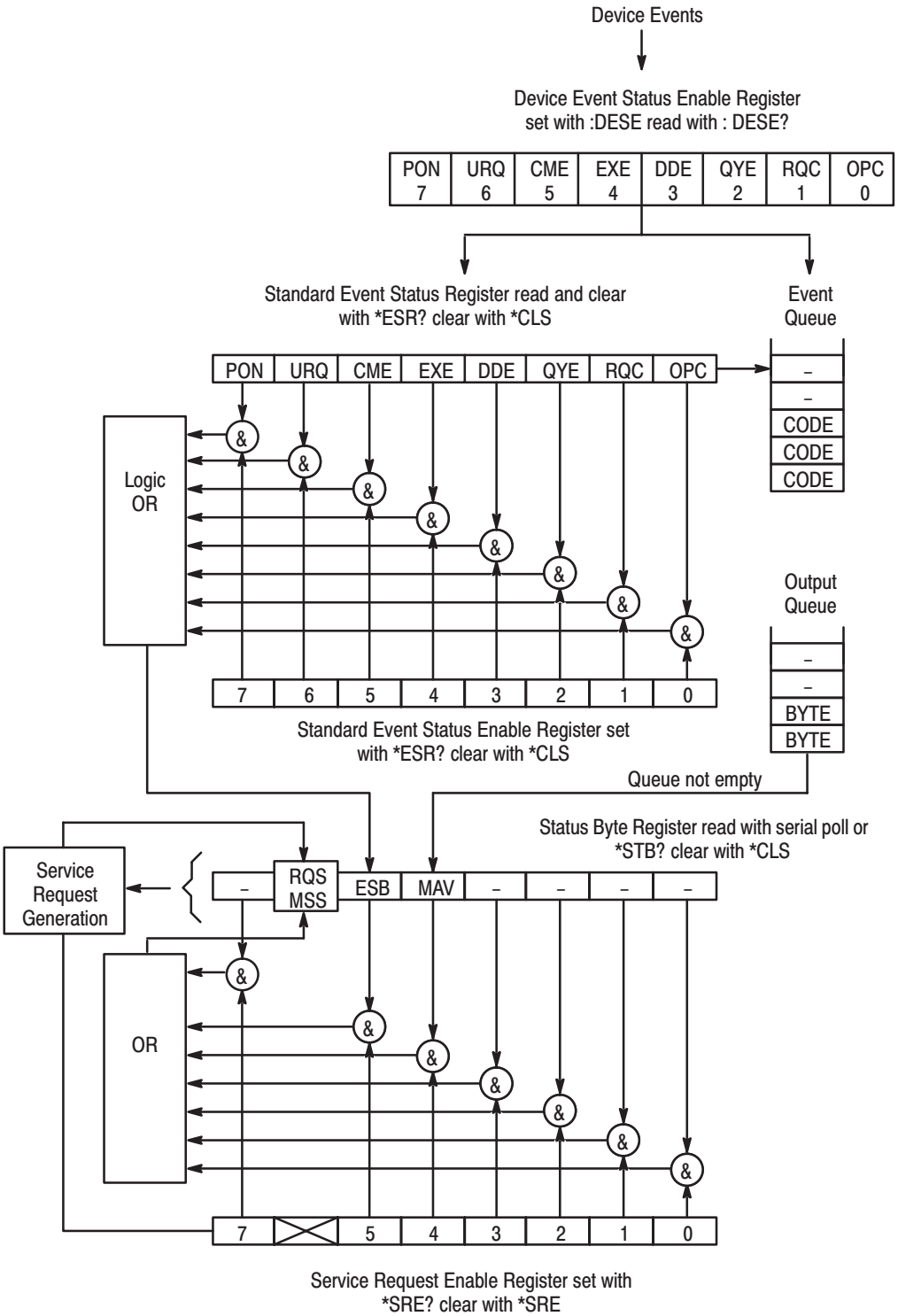


Figure 3-6: Status and Event Handling Process Overview

I/O Status and Event Screen

Figure 3-7 shows the contents of GPIB status and event reporting system displayed on the screen. Use the following procedure to display the status and event screen.

1. Press the UTILITY button in the MENU column to the right of the screen. The UTILITY button menu appears above the bottom menu buttons.
2. Press the Misc bottom menu button to display the Misc side menu.
3. Press the Status... side button to display the status submenu.
4. Press the I/O side button to display the I/O submenu.

The status and event screen displays the registers: DESER, SESR, ESER, SBR and SRER. Each of these registers is displayed with the decimal equivalent of its contents shown in brackets. Events which can be dequeued are indicated in the Avail column of the Events Queue part of the display. All events currently in the queue are indicated as pending in the Pend column of the display.

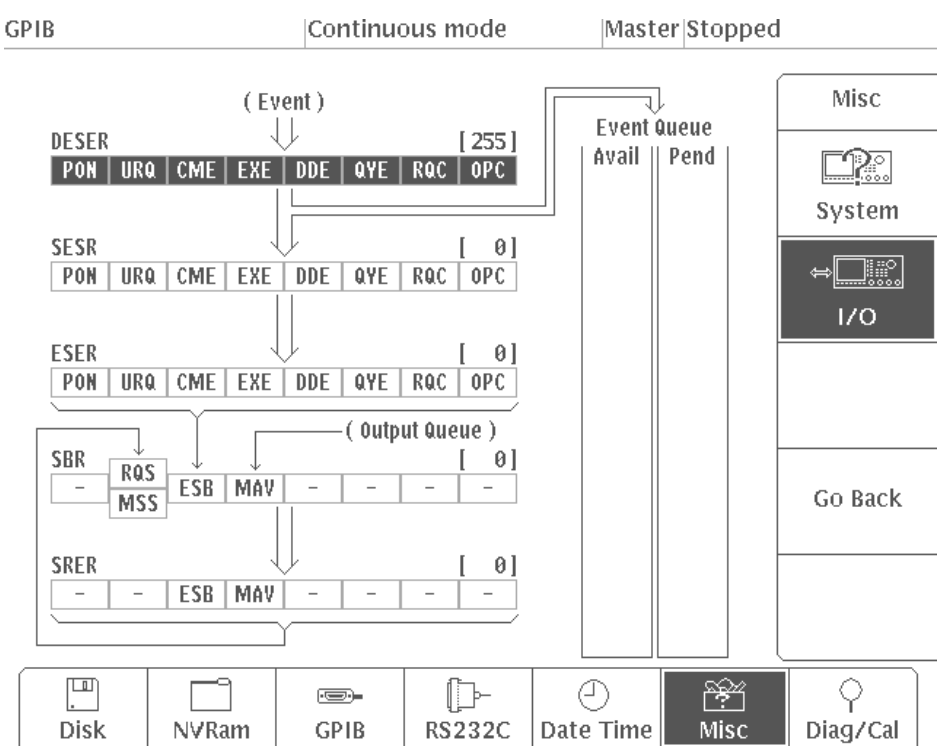


Figure 3-7: Status and Event Screen

Messages

Tables 3-3 through 3-12 list the status and event messages used in the GPIB/ RS-232-C status and event reporting system. You use the *ESR? query to make the messages available for dequeuing; you use the :EVENT?, EVMsg?, and ALLEv? queries to dequeue and return the messages. The messages return as follows:

- The :EVENT? query command returns the event code only. When using these query commands, use the *ESR? query to make the events available for return.
- The EVMsg?, and ALLEv? queries return both the event code and event message in the following format:

  <event code>, "<event message ; secondary message>"

Most messages returned have both an event message, followed by a semicolon (;), and a second message which contains more detailed information. Although these secondary messages are not listed in this manual, you can use the EVMsg? and ALLEv? queries to display them.

Table 3-3 lists the definition of event codes.

Table 3-3: Definition of Event Codes

Event Class	Event Code Ranges	Descriptions
No Events	0–1	No event nor status
Reserved	2–99	(unused)
Command Errors	100–199	Command errors
Execution Errors	200–299	Command execution errors
Device-Specific Errors	300–399	Internal device errors (Hardware errors)
Query Errors	400–499	System event and query errors
Execution Warnings	500–599	Execution warnings
Internal Warnings	600–699	Internal warnings
Reserved	700–1999	(unused)
Extended Execution Errors	2000–2999	Device dependent command execution errors
Extended Device-Specific Errors	3000–3999	Device dependent device errors
Reserved	4000–	(unused)

Table 3-4 lists the message when the system has no events nor status to report. These have no associated SESR bit.

Table 3-4: Normal Condition

Code	Description
0	No events to report — queue empty
1	No events to report — new events pending *ESR?

Table 3-5 lists the error messages generated due to improper command syntax. In this case, check that the command is properly formed and that it follows the syntax.

Table 3-5: Command Errors (CME Bit:5)

Code	Description
100	Command error
101	Invalid character
102	Syntax error
103	Invalid separator
104	Data type error
105	GET not allowed
106	Invalid program data separator
108	Parameter not allowed
109	Missing parameter
110	Command header error
111	Header separator error
112	Program mnemonic too long
113	Undefined header
114	Header suffix out of range
118	Query not allowed
120	Numeric data error
121	Invalid character in number
123	Exponent too large
124	Too many digits
128	Numeric data not allowed
130	Suffix error

Table 3-5: Command Errors (CME Bit:5) (Cont.)

Code	Description
131	Invalid suffix
134	Suffix too large
138	Suffix not allowed
140	Character data error
141	Invalid character data
144	Character data too long
148	Character data not allowed
150	String data error
151	Invalid string data
152	String data too long
158	String data not allowed
160	Block data error
161	Invalid block data
168	Block data not allowed
170	Expression error
171	Invalid expression
178	Expression data not allowed
180	Macro error
181	Invalid outside macro definition
183	Invalid inside macro definition
184	Macro parameter error

Table 3-6 lists the execution errors that are detected during execution of a command.

Table 3-6: Execution Errors (EXE Bit:4)

Code	Description
200	Execution error
201	Invalid while in local
202	Settings lost due to RTL
203	Invalid password
210	Trigger error
211	Trigger ignored
212	Armed ignored
213	Init ignored
214	Trigger deadlock
215	ARM deadlock
220	Parameter error
221	Settings conflict
222	Data out of range
223	Too much data
224	Illegal parameter value
225	Parameter under range
226	Parameter over range
227	Parameter rounded
230	Data corrupt or stale
231	Data questionable
240	Hardware error
241	Hardware missing
250	Mass storage error
251	Missing mass storage
252	Missing media
253	Corrupt media
254	Media full
255	Directory full
256	File name not found
257	File name error

Table 3-6: Execution Errors (EXE Bit:4) (Cont.)

Code	Description
258	Media protected
260	Expression error
261	Math error in expression
262	Expression syntax error
263	Expression execution error
270	Macro error
271	Macro syntax
272	Macro execution error
273	Illegal macro label
274	Macro parameter error
275	Macro definition too long
276	Macro recursion error
277	Macro redefinition not allowed
278	Macro header not found
280	Program error
281	Cannot create program
282	Illegal program name
283	Illegal variable name
284	Program currently running
285	Program syntax error
286	Program run time error

Table 3-7 lists the internal errors that can occur during operation of the waveform generator. These errors may indicate that the waveform generator needs repair.

Table 3-7: Execution Errors (EXE Bit:4)

Code	Description
300	Device-specific error
310	System error
311	Memory error
312	PUD memory lost
313	Calibration memory lost

Table 3-7: Execution Errors (EXE Bit:4) (Cont.)

Code	Description
314	Save/recall memory lost
315	Configuration memory lost
330	Self-test failed
350	Queue overflow (does not affect the DDE bit)

Table 3-8 lists the system event messages. These messages are generated whenever certain system conditions occur.

Table 3-8: System Event and Query Errors

Code	Description
401	Power on
402	Operation complete
403	User request
404	Power fail
405	Request control
410	Query INTERRUPTED
420	Query UNTERMINATED
430	Query DEADLOCKED
440	Query UNTERMINATED after indefinite response

Table 3-9 lists warning messages that do not interrupt the flow of command execution. These messages warn you that you may get unexpected results.

Table 3-9: Warnings (EXE Bit:4)

Code	Description
500	Execution warning

Table 3-10 lists internal errors that indicate an internal fault in the waveform generator.

Table 3-10: Internal Warnings (DDE Bit:3)

Code	Description
600	Internal warning
610	Data not multiple of 32 points

Table 3-11 lists status messages that are specific to the waveform generator. These messages appear when a operation starts, ends, or is in process. These messages have no associated SESR bit.

Table 3-11: Device-Dependent Command Execution Errors

Code	Description
2000	File error
2001	Directory not empty
2002	Too many files
2003	File locked
2004	File already exists
2005	File already opened
2006	Invalid file type
2007	File type mismatch
2008	Internal memory full
2009	Invalid file format
2010	Comment error
2012	Invalid data in comment string
2020	Waveform error
2021	Waveform request is invalid
2022	Too much curve data
2024	Curve data byte count error
2025	Waveform load error
2026	Internal waveform memory full
2027	Waveform size invalid
2028	Missing waveform data
2030	Marker error

Table 3-11: Device-Dependent Command Execution Errors (Cont.)

Code	Description
2031	Marker request is invalid
2032	Too much marker data
2040	Equation error
2042	Too much equations
2043	Equation too long
2044	Invalid equation syntax
2046	Equation compile error
2050	Sequence error
2052	Too much sequence data
2053	Invalid sequence repeat count
2054	Invalid sequence syntax
2055	Sequence load error
2056	Internal sequence memory full
2057	Recursive sequence
2058	Sequence in sub-sequence
2059	Sequence incomplete
2060	Autostep error
2062	Too much autostep data
2063	Invalid autostep data
2064	Invalid autostep syntax
2070	Data error
2071	Invalid data syntax
2072	Invalid data value
2080	Time error
2081	Invalid time syntax
2082	Invalid time value
2090	Message error
2100	Hardcopy error
2101	Hardcopy busy
2102	Hardcopy timeout error
2110	Clock sweep error
2112	Too much clock sweep data
2113	Internal clock sweep memory full

Table 3-11: Device-Dependent Command Execution Errors (Cont.)

Code	Description
2114	Clock sweep size invalid
2115	Invalid clock sweep dwell
2116	Invalid clock sweep frequency
2120	PLL lock timeout

Table 3-12 lists device error messages that are specific to the device.

Table 3-12: Extended Device Specific Errors

Code	Description
3001	RS-232-C input buffer overflow

Execution Synchronization

The GPIB commands used in these waveform generators are designed to be executed in the order in which they are sent from the external controller. However, since certain commands require a certain amount of time for their execution to complete, these waveform generators are designed to allow the execution of the command that is sent next at the same time. With these types of commands there are cases where the waveform generator must wait for the execution of the first command to complete before executing the next command.

The following commands allow the simultaneous execution of other commands before their execution has completed.

EQUAtion : COMPile[: STATe EXECute,]<Equation File>
HCOPy STARt

These waveform generators provide the following commands for performing synchronization control.

*WAI
*OPC
*OPC?

*WAI Command

In general, the *WAI command is the simplest method for execution synchronization. All that is required is to send a *WAI command before sending the next command as shown in the following example.

: EQUATION : COMPILE : STATE EXECUTE
“SAMPL.EQU” ; *WAI ;: CH1 : WAVEFORM “SAMPL.WFM”

Synchronization Using the *OPC Command

The *OPC command sets the OPC bit in the SESR (standard event status register) when all pending processing has completed. This command allows the most effective execution completion monitoring to be performed when used together with serial poling or the service request function. As shown in the following example, essentially identical processing sequences can be achieved by either method.

Enable the corresponding status register

: DESE 1
*ESE 1

*SRE 0 (If serial polling is used)

Or:

*SRE 32 (If the service request function is used)

For example, use the following commands to start the compilation of an equation file and then wait for the compilation to complete.

: EQUATION : COMPILE “SAMPL.EQU” ; *OPC

(This either waits while the serial poll function is 0, or waits for a service request to occur.)

Now use the following command to load the waveform file generated by the compilation into channel 1.

: CH1 : WAVEFORM “SAMPL.WFM”

See “Programming Example 3” in Section 4 for more explicit details on the use of this technique.

The *OPC Query

The *OPC? query returns the ASCII code for “1” as the response if all pending processing has completed. Execution completion monitoring can be performed as shown in the example below using this query.

For example, use the following commands to start the compilation of an equation file and then wait for the compilation to complete.

: EQUATION : COMPILE “SAMPL.EQU” ; *OPC?

(Now, wait for a “1” to be returned as the response. Note that when a GPIB interface is used, a timeout may occur before the data is written into the queue while waiting to data from the output queue.)

Now use the following command to load the waveform file generated by the compilation into channel 1.

: CH1 : WAVEFORM “SAMPL.WFM”

Examples

Programming Examples

This section describes the example programs that illustrate methods that you can use to control the Arbitrary Waveform Generator over the GPIB interface. The floppy disk supplied with the waveform generator contains source lists for these programs written in Microsoft QuickC 2.0 and Microsoft Quick BASIC 4.5.

The programs run on PC compatible system equipped with a National Instruments GPIB board and associated drivers.

All the Microsoft example programs assume that the GPIB system recognizes the instrument as DEV1 and the PC (external controller) as GPIB0.

The example software includes:

getwfm This program transfers a waveform and its preamble from the waveform generator to a file or displays the waveform in a scaled format.

putwfm This program transfers a waveform to the waveform generator.

equset This program sends equation expression data to the waveform generator, instructs the waveform generator to compile it into a waveform, then sets up the waveform generator so it outputs the compiled waveform from the CH1 output connector. This program demonstrates the use of the *OPC and serial poll synchronization method to determine when the compile completes in order to go on to next step.

intrv This program demonstrates interactive communication between the external controller and the waveform generator (QuickC only).

Compiling the Example Programs

The floppy disk *GPIB Programming Examples* contains the programs just described written in Microsoft QuickC 2.0 or Quick BASIC 4.5. Source program files and the MAKE file are placed in the directory of that disk. All files in the directory should be copied to a directory on the hard disk.

To create the executable program files, perform following steps:

In case of QuickC

1. Install QuickC. Select the SMALL memory model. Be sure to set up your path so DOS can access the directory where QuickC is installed.

2. Install the National Instruments PC2/PC2A GPIB board and drivers. Remember to identify the GPIB device as DEV1. This identifier is defined using the IBCONF.EXE program.

3. Copy the files from the supplied floppy disk to your hard disk. A special directory should be created to store them. For example, if you wish to store the example programs in hard disk C and you have placed the *AWG2000 Series GPIB Programming Examples* disk in floppy drive B, switch to drive C and type:

```
mkdir examples

cd examples

copy B:\C\*.* .
```

4. For this installation, the files: DECL.H and MCIBS.OBJ must be copied from your National Instruments PC2/PC2A GPIB drivers directory to this directory. Assuming you have installed these drivers in `gpib-pc`, you would type:

```
copy \gpib-pc\decl.h .

copy \gpib-pc\mcibs.obj .
```

5. To compile and link all sample programs, simply type:

```
nmake /F samp.mak
```

In case of Quick BASIC

1. Install Quick BASIC. Be sure to set up your path so DOS can access the directory where Quick BASIC is installed.

2. Install the National Instruments PC2/PC2A GPIB board and drivers. Remember to indentify the GPIB device as DEV 1. This identifier is defined using the IBCONF. EXE program.

3. Copy the files from the supplied floppy disk to your hard disk. A special directory should be created to store them. For example, if you wish to store the example programs in hard disk C and you have placed the GPIB Programming Examples disk in floppy drive B, switch to drive C and type :

```
mkdir example

cd example

copy B:\basic\*.*.
```

4. For this installation, the files GPIB. OBJ and GPDECL. BAS must be copied from your National Instruments PC2/PC2A GPIB drivers directory to this directory.

```
copy \gpib-pc\qbasic\gbib.obj .

copy \gbib-pc\qbasic\gpdecl.bas .
```

   And copy the files from Quick BASIC directory.

```
copy \gpib-pc\bin\bc.exe .

copy \gpib-pc\bin\link.exe .

copy \gpib-pc\lib\bcom45.lib .
```

5. To compile and link all sample program, simply type:

```
makeexe.bat
```

Executing the Example Programs

The programs can be executed as described below.

Getwfm

This command reads a waveform from the waveform generator and stores it with preamble into file, or it displays it in a scaled format. To run the `getwfm` program type:

```
getwfm <source> [<out-file>]
```

where:

`<source>` is either the number of the waveform generator channel or the name of a source waveform file in the internal memory of the waveform generator. In either case, it is the source from which the waveform data is transferred.

`<out-file>` is the name of a file in which the binary unscaled waveform data and the preamble are to be stored. If no `<out-file>` is specified, the waveform data displayed on the default video output device in scaled format.

The waveform data in the file can be returned to the waveform generator using putwfm command.

Putwfm

This command sends the waveform to the waveform generator. To run the `putwfm` program, type:

```
putwfm <destination> <waveform-file>
```

where

<destination> is the waveform file in the internal memory of the waveform generator to which the waveform data is to be transferred.

<waveform-file> is a file storing waveform data and preamble previously obtained using the getwfm command.

Equset This command sends equation expression data to the waveform generator, then has the waveform generator compile it and output it. To run the equset program, type:

```
equset
```

The equation expression data is contained in the C source program, which transfers it to the AWG2000 and gives it the file name EQUSAMPL.EQU. (If a file with this name already exists, the command is not executed; you must remove that file from the AWG2000 before you can execute equset.)

The synchronization method is used to hold off further operation until the compile operation is complete, which may take about 35 seconds. Compilation of the data creates the waveform file EQUSAMPL.WFM.

The waveform file is set to CH1 with the amplitude 5.0 V, frequency 25 MHz, and output in triggered mode.

Intrv This command sets up interactive communication between the external controller and the AWG2000. To run the intrv program, type:

```
Intrv
```

which in turn displays the prompt:

```
AWG2020 >>
```

This interactive prompt indicates the program is waiting for you to type a command. The following sorts of commands can be entered in response to this prompt:

GPIB commands. ll commands and queries defined in this programmers manual can be used. The response message to the query is immediately output on the standard output device.

Built-in commands. The following commands are built in to the `intrv` program.

`help`	Displays this message.
`view`	Displays the contents of the file in the external controller, which is specified by an argument. To use this command type `view <path-name>`.
`exec`	Reads command lines from the file specified by an argument instead of the standard input. When the EOF (end of file) is detected, command lines are read from the standard input again. To use, type `exec <description-file>`.
`status`	Reads the status byte from the AWG2000.
`resets`	Resets the registers in the event and status reporting system to the default values established for this program. This command must be used after you change the value of those registers using GPIB commands such as :DESE, *ESE, etc.

Redirection. The >,<, and >> redirection operators can be combined with `file names` and used in commands to redirect the standard input or output.

`< file name`	Uses the contents of `file name` as the standard input to the preceding command. If this redirection command is not preceded by a command, the contents of the file are directly transferred to the AWG2000.
`> file name`	Writes to `file name` as the standard output device, creating that file if it does not exist. If the file `file name` does already exist, the redirected standard output overwrites it, erasing its previous contents.
`>>file name`	Writes to `file name` as the standard output, creating that file if it does not exist. If the file `file name` already exists, its contents are not overwritten; instead, the standard output is added to the end of the existing contents.

Last command. The `!!` operator can be entered in a command line to reference a previous command. When typed in the current command line, the contents of the command line last entered replaces the `!!` operator.

Example 1: Waveform Transfer #1

The first example illustrates a simple waveform transfer from the instrument to the external controller.

In case of QuickC

```
/*
 * getwfm.c – a simple waveform transfer program that converts to
 * scaled waveform in ASCII format, or save raw waveform and preamble
 * to a file in the external controller.
 */
# include <stdio.h>
# include <stdlib.h>
# include "decl.h"
# include "exit.h"

# define        MAX_DATA2       4000
# define        MAX_DATA        (MAX_DATA2 / 2)
# define        CMD_LEN         80
# define        LEN12           12
# define        FILE_OUT        1
# define        STD_OUT         -1

typedef         float   FLOAT;
typedef         double  DOUBLE;
typedef         long    LONG;
typedef         short   SHORT;

void    checkarg();
char    *awgWR();

SHORT   wfm[MAX_DATA + 1];      /* Array for raw awg input              */
LONG    nr_pt;                  /* Preamble: number of data points      */
LONG    pt_off;                 /* Preamble: point offset               */
FLOAT   yoff;                   /* Preamble: Y offset                   */
FLOAT   ymult;                  /* Preamble: Y multiple                 */
FLOAT   xincr;                  /* Preamble: X increment                */
char    xunit[LEN12 + 1];       /* Preamble: X unit representation      */
char    yunit[LEN12 + 1];       /* Preamble: Y unit representation      */

char    *outfile;       /* Output file descriptor                       */
char    *source;        /* Source from which a waveform is transferred  */
int     fflag = STD_OUT;

main(argc, argv)
int     argc;
char    *argv[];
```

```
{
        printf("\n\n");
        printf("GETWFM  -  simple waveform transfer program.\n");
        printf("Copyright (c) 1993 Sony/Tektronix, Corp.");
        printf(" All Rights Reserved.\n\n");

        checkarg(argc, argv);   /* Check arguments and open output device */
        open_dev();             /* Find GPIB devices                      */
        SrcSetup();             /* Define source in the instrument        */
        if (fflag == STD_OUT)
                ReadandStdout();        /* Get waveform and convert to
                                           scaled waveform                */
        else
                ReadandFileout();       /* Get waveform and preamble, and
                                           then save into a file          */
        close_dev();
}

/*
 * Check if the arguments are valid.
 */

void    checkarg(argc, argv)
int     argc;
char    *argv[];
{

/*
 * Check command line argument count.
 */

        if ((argc < 2)  || (argc > 3)){
                fprintf(stderr, "usage: getwfm <source> [<out-file>]\n");
                fprintf(stderr, "\twhere:\n");
                fprintf(stderr, "\t\t<source>\t\tis the source channel or");
                fprintf(stderr, " file to read\n");
                fprintf(stderr, "\t\t[<outfile>]\tis the optional save");
                fprintf(stderr, " output file\n");
                exit(1);
        }

/*
 * Check for valid source channel, or waveform file name.
 */
```

```
        source = argv[1];
        if (strcmp(argv[1], "CH3") != 0 && strcmp(argv[1], "CH4") != 0 &&
           wfmfile(argv[1]) != 0)
        {
                fprintf(stderr, "ERROR: Invalid Source: \"%s\":", argv[1]);
                fprintf(stderr, " No Waveform Acquired\n");
                exit(1);
        }

/*
 * Open output file if specified otherwise use stdout.
 */

        if (argc == 3)
        {
                outfile = argv[2];
                fflag = FILE_OUT;
        }
}

/*
 * Check the file extension is '.wfm'.
 */

wfmfile(name)
char    *name;
{
        int     dlen = strlen(name);

        if (dlen < 4 || dlen > 12)
                return -1;
        if (strcmp(&name[dlen - 4], ".WFM") == 0 ||
            strcmp(&name[dlen - 4], ".wfm") == 0)
                return 0;
        return -1;
}

/*
 * Define source in the instrument, and set encoding format and byte order.
 *
 * WARNING — This program assumes a CPU with little Indian so that the
 * byte order in waveform transfer is set to LSB with :WFMPRE:BYT_OR command.
 * If the CPU with big Indian is used, byte order must be set to MSB.
 */
```

```
SrcSetup()
{
        char cmd[CMD_LEN + 1];
        sprintf(cmd,":DATA:SOURCE \"%s\";ENCDG RPBINARY;:WIDTH2",
                                                                source);
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Write Error: Unable to Setup waveform parameters");
                exit(1);
        }
}

/*
 * Read preamble and waveform, then save them into a file.
 */

ReadandFileout()
{
        if (awgWrite(":HEADER  ON") < 0)
        {
           gpiberr("Write Error: Unable to turn header on\n");
           exit(1);
        }
        if (awgWrite(":WAVFRM?") < 0)
        {
           gpiberr("Write Error: Unable to write :WAVFRM? query");
           exit(1);
        }
        if (WrtGtoF(outfile) < 0)
        {
           gpiberr("Read Error/File Open Error:");
           exit(1);
        }
}

/*
 * Read waveform data and convert to scaled waveform.
 *
 * The waveform is formatted as #<x><yyy><data> where
 *      <x> is the number of y bytes; for example if yyy = 500, then
 *           x = 3
 *    <yyy> is the number of bytes to transfer;
 *           if width is 1 then all bytes on bus are single data
 *           points; if width is 2 then bytes on bus are
```

```
 *          2-byte pairs; this program uses width of 2
 *   <data> is the curve data
 */

ReadandStdout()
{
        char cmd[CMD_LEN + 1];
        LONG    llen;                   /* data size                    */
        LONG    li;                     /* loop index                   */
        int     dlen;                   /* size                         */
        int     c;
        int     i;                      /* loop index                   */

/*
 * Get some parameters in preamble.
 */

        if (awgWrite(":HEADER  OFF") < 0)
        {
                gpiberr("Write Error: Unable to turn header off\n");
                exit(1);
        }
        nr_pt = atol(awgWR("WFMPRE:NR_PT?", cmd, CMD_LEN));
        yoff  = atof(awgWR("WFMPRE:YOFF?", cmd, CMD_LEN));
        ymult = atof(awgWR("WFMPRE:YMULT?", cmd, CMD_LEN));
        xincr = atof(awgWR("WFMPRE:XINCR?", cmd, CMD_LEN));
        pt_off = atol(awgWR("WFMPRE:PT_OFF?", cmd, CMD_LEN));
        awgWR("WFMPRE:XUNIT?", xunit, LEN12);
        awgWR("WFMPRE:YUNIT?", yunit, LEN12);

/*
 * Read the header information in <Arbitrary Block>.
 * The header includes #<x><yyy>.
 */

        if (awgWrite(":CURVE?") < 0)
        {
                gpiberr("Write Error: CURVE?");
                exit(1);
        }
        awgRead(cmd, 1);        /* Read the '#' symbol                          */
        awgRead(cmd, 1);        /* Read string length of num bytes to transfer */
        c = atoi(cmd);          /* Convert string to integer                    */
        awgRead(cmd, c);        /* Read string containing number of bytes
                                   to transfer                                  */
        llen = ldiv(atol(cmd), 2L).quot;  /* Two bytes per one data point    */
```

```
/*
 * Read the raw waveform data, process waveform data
 */
        fprintf(stdout, "%s,%s,\"%s\", max. number of data point (%ld)\n",
                                                xunit, yunit, source, nr_pt);
        for (li = 0; li < llen;)
        {
                if (awgRead(wfm, MAX_DATA2) < 0)
                {
                        gpiberr("Read Error: WAVEFORM");
                        exit(1);
}
/*

* Output scaled x, y values in (Sec, Volts)
* Time[li] = (li – PT_OFF) * XINCR
* Volts[li] = (point value – YOFF) * YMULT
*/
                dlen = ibcnt / 2; /* Two bytes per one data point */
                for(i = 0; i < dlen; i++)
                {
                        fprintf(stdout, "%.2e,%.2e\n\r",
                                (FLOAT)(li - pt_off)*(FLOAT)(xincr),
                                (FLOAT)(((FLOAT)wfm[i] - (FLOAT)yoff) * ymult));
                        li++;
                }
        }

/*
 * Cleanup
 */
        if (awgWrite(":HEADER ON") < 0)
        {
                gpiberr("Write Error: Unable to turn header on\n");
                exit(1);
        }
        fprintf(stdout, "\n");
        fprintf(stdout, "Waveform from %s successfully transferred!\n", source);
        return 0;
}
```

```
/*
 * Write GPIB query, and immediately read the response.
 */

char    *awgWR(cmd, resp, cnt)
char    *cmd, *resp;
int     cnt;
{
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Write Error: WFMPRE?");
                exit(1);
        }
        if (awgRead(resp, cnt) < 0)
        {
                gpiberr("Read Error: WFMPRE");
                exit(1);
        }
        resp[ibcnt -1] = '\0';  /* Replace '\n' at the end of response
                                   with '\0'.                          */
        return resp;
}
```

In case of Quick BASIC

```
DECLARE SUB GPIB2ASC (DEV%, FLNAME$)
DECLARE SUB GPIB2ISF (DEV%, FLNAME$)
DECLARE SUB CHKSTAT (DEV%, ESR%, EVENT$)
DECLARE SUB FINDDEV (KEYNAME$, DEV%)
DECLARE SUB EXTOPT (OPTION$, SOURCE$, FLNAME$)
DECLARE FUNCTION DISKERR$ ()
'$INCLUDE: 'QBDECL.BAS'
PRINT
PRINT "GETWFM Ver.1.0 "
PRINT "      Sample Progarm for AWG2000 series"
PRINT "            Copyright(C)1993,SONY/Tektronix Corp. Allright Reserved."
PRINT "           No warranty."
'
'Check COMMAND Arguments and extract source & filename
'
   OPTION$ = COMMAND$
   CALL EXTOPT(OPTION$, SOURCE$, FLNAME$)
'
'GPIB address search
'
```

```
   KEYNAME$ = ”SONY/TEK,AWG2”
   CALL FINDDEV(KEYNAME$, DEV%)
   PRINT KEYNAME$
   IF DEV% = 0 THEN BEEP: END
’
’Check DATA source
’
   WRT$ = ”HEADER ON;:DATA:SOURCE ’” + SOURCE$ + ”’;:WFMPRE?”
   CALL IBWRT(DEV%, WRT$)
   RD$ = SPACE$(500): CALL IBRD(DEV%, RD$)
   IF INSTR(RD$, ”WFID”) = 0 THEN
      BEEP
      PRINT ”ERROR. ”; SOURCE$; ” data is none.”
      END
   END IF
’
’Set DATA ENCDG to SRPBINARY. It’s a signed integer and transfer the LSB data fi’rst.
’
   CALL IBWRT(DEV%, ”data:encdg srpbin;width 2”)
’
’Choose saved data type with extention.
’
   IF INSTR(FLNAME$, ”.CSV”) OR INSTR(FLNAME$, ”CONS:”) THEN
      CALL GPIB2ASC(DEV%, FLNAME$)
   ELSE
      CALL GPIB2ISF(DEV%, FLNAME$)
   END IF
’
’Check GPIB Status.
’
   DO
      CALL CHKSTAT(DEV%, ESR%, EVENT$)
      IF ESR% <> 0 THEN
         BEEP
         PRINT ”Worning.”
         PRINT EVENT$
      END IF
   LOOP UNTIL ESR% = 0

END
’
’ERROR Trap routine.
’
ERRHANDLER:
```

```
   BEEP
   PRINT ”ERROR. ”; DISKERR$
   END
END
’————————————————————————————————End of Main procedure

SUB CHKSTAT (DEV%, ESR%, EVENT$)

   CALL IBRSP(DEV%, sta%)
   CALL IBWRT(DEV%, ”*esr?”)
   RD$ = SPACE$(16)
   CALL IBRD(DEV%, RD$)
   ESR% = VAL(RD$)

   CALL IBWRT(DEV%, ”allev?”)
   RD$ = SPACE$(500)
   CALL IBRD(DEV%, RD$)
   EVENT$ = LEFT$(RD$, IBCNT% – 1)

END SUB

FUNCTION DISKERR$
   SELECT CASE ERR
      CASE 54
         DISKERR$ = ”Bad file mode”
      CASE 64
         DISKERR$ = ”Bad file name”
      CASE 52
         DISKERR$ = ”Bad name or number”
      CASE 25
         DISKERR$ = ”Device fault”
      CASE 57
         DISKERR$ = ”Device I/O error”
      CASE 24
         DISKERR$ = ”Device timeout”
      CASE 68
         DISKERR$ = ”Device unavailable”
      CASE 61
         DISKERR$ = ”Disk full”
      CASE 72
         DISKERR$ = ”Disk-media error”
      CASE 71
         DISKERR$ = ”Disk not ready”
      CASE 53
         DISKERR$ = ”File not found”
```

```
      CASE 62
         DISKERR$ = ”Input past end of file”
      CASE 76
         DISKERR$ = ”Path not found”
      CASE 75
         DISKERR$ = ”Path/File sccess error”
      CASE 70
         DISKERR$ = ”Permission denied”
      CASE 67
         DISKERR$ = ”Too many files”
      CASE ELSE
         DISKERR$ = ”???”
      END SELECT
END FUNCTION

SUB EXTOPT (OPTION$, SOURCE$, FLNAME$)

  IF OPTION$ = ”” THEN GOTO DISPUSAGE
  OPTION$ = OPTION$ + ” ”
  SOURCE$ = ””
  FLNAME$ = ””
’
’Extract string between spaces.
’
   FOR I% = 1 TO LEN(OPTION$)
      A$ = MID$(OPTION$, I%, 1)
      IF A$ = ” ” THEN EXIT FOR
      SOURCE$ = SOURCE$ + A$
   NEXT I%

   FOR J% = I% + 1 TO LEN(OPTION$)
      A$ = MID$(OPTION$, J%, 1)
      IF A$ = ” ” THEN EXIT FOR
      FLNAME$ = FLNAME$ + A$
   NEXT J%
’
’clean up DATA source.
’
   IF FLNAME$ = ”” THEN FLNAME$ = ”CONS:”
   IF SOURCE$ = ”CH1” THEN EXIT SUB
   IF SOURCE$ = ”CH2” THEN EXIT SUB
   IF INSTR(SOURCE$, ”.WFM”) THEN EXIT SUB
   BEEP
   PRINT ”Invalid argument.”
```

```
DISPUSAGE:
   PRINT
   PRINT ”Usage:GETWFM <source> [<filename>]”
   PRINT
   PRINT ”        <source>:Waveform data to transfer”
   PRINT ”                CH1/CH2/Wafeform file.”
   PRINT ”                Wafeform file have a ’.WFM’ extention.”
   PRINT
   PRINT ”        [<filename>]:Output filename”
   PRINT ”                If no spec, dislay on the screen.”
   PRINT ”                Specially ’.CSV’ extention is given, convert to the ascii”
   PRINT ”                data for spread sheet software.”
END
END SUB

SUB FINDDEV (KEYNAME$, DEV%)

   CALL IBFIND(”GPIB0”, BD%)
   IF BD% < 0 THEN
     KEYNAME$ = ”’GPIB0’ not found.”
     DEV% = 0
     EXIT SUB
   END IF

   CALL IBFIND(”DEV1”, DEV%)
   IF DEV% <= 0 THEN
     KEYNAME$ = ”’DEV1’ not found, Please run IBCONF and define.”
     DEV% = 0
     EXIT SUB
   END IF
   CALL IBSRE(BD%, 0)
   CALL IBSRE(BD%, 1)

   V% = 11: CALL IBTMO(DEV%, V%)
   AD% = 0
’
’GPIB Address Search
’
DO
   CALL IBPAD(DEV%, AD%)
   CALL IBWRT(DEV%, ”*IDN?”)
   IF IBSTA% AND &H8000 THEN
     AD% = AD% + 1
   ELSE
     id$ = SPACE$(100): CALL IBRD(DEV%, id$)
```

```
      IF INSTR(id$, UCASE$(KEYNAME$)) THEN
         EXIT DO
      ELSE
         AD% = AD% + 1
         CALL IBCLR(DEV%)
      END IF
   END IF
   IF 30 < AD% THEN
      KEYNAME$ = "Specified instrument not found."
      DEV% = 0
      EXIT SUB
   END IF
LOOP

   V% = 13: CALL IBTMO(DEV%, V%)
   KEYNAME$ = LEFT$(id$, IBCNT% – 1) + "  (GPIB Address =" + STR$(AD%) + ")"

   CALL IBWRT(DEV%, ":DESE 255;*CLS")

END SUB

SUB GPIB2ASC (DEV%, FLNAME$)
'
'Request to send the waveform data.
'
   CALL IBWRT(DEV%, "CURVE?")
'
'Read the waveform data
'
'The waveform data is formatted as #<x><yyy><data><newline> where
'      <x> is the number of bytes of <yyy>, for example if yyy = 500, then x = 3.
'      <yyy> is the number of bytes to transfer inclulude checksum.
'           (The AWG don't send cheksum.)
'              The resolution in the AWG2000 is 12 bits/point, then the number of bytes at
'              one point data is two. The Length of waveform data is the harf of yyy.
'      <data> is the curve data.
'       <newline> End of data block.(=0AH(linefeed character))
'
  RD$ = SPACE$(1)              'define buffer to 1 byte
  DO
     CALL IBRD(DEV%, RD$)    'read and discard until '#' symbol
     LOOP UNTIL RD$ = "#"
     CALL IBRD(DEV%, RD$)          'read <x>
     RD$ = SPACE$(VAL(RD$))        'set buffer to x bytes
     CALL IBRD(DEV%, RD$)          'read <yyy>
```

```
    BYTCNT& = VAL(RD$)              'get the number of bytes to transfer
'
'Define an arrey for raw data. It's the twe bytes signed integer arrey.
'The length of the arrey is the harf of total bytes count.
'
    NRPT& = BYTCNT& / 2
'
'Limit the data length to 32k bytes.
'
    IF 32767 <= NRPT& THEN
       BEEP
       PRINT "Data length is too long. Set to till 32k words."
       DO
          CALL CHKSTAT(DEV%, ESR%, EVENT$)
       LOOP UNTIL ESR% = 0
       END
    END IF

    NRPT% = NRPT&
    BYTCNT% = BYTCNT&

    DIM WFM%(NRPT% – 1)     'Option base is 0.
'
'Read the waveform data at two bytes pair by IBRDI.
'
    CALL IBRDI(DEV%, WFM%(), BYTCNT%)
    IF IBSTA% < 0 THEN
       BEEP
       PRINT "Error on Reaf waveform data."
       END
    END IF
'
'Read the End charctor
'
    RD$ = SPACE$(2)
    CALL IBRD(DEV%, RD$)

'
'Read Scale data and Convert the raw data to voltae value.
'
    CALL IBWRT(DEV%, ":HEADER OFF;:WFMPRE:YOFF?")
    RD$ = SPACE$(40)
    CALL IBRD(DEV%, RD$)
    YOFF! = VAL(RD$)
```

```
    CALL IBWRT(DEV%, ”WFMPRE:YZERO?”)
    RD$ = SPACE$(40)
    CALL IBRD(DEV%, RD$)
    YZERO! = VAL(RD$)

    CALL IBWRT(DEV%, ”WFMPRE:YMULT?”)
    RD$ = SPACE$(40)
    CALL IBRD(DEV%, RD$)
    YMULT! = VAL(RD$)

    CALL IBWRT(DEV%, ”WFMPRE:XINCR?”)
    RD$ = SPACE$(40)
    CALL IBRD(DEV%, RD$)
    XINCR! = VAL(RD$)

    CALL IBWRT(DEV%, ”WFMPRE:PT_OFF?”)
    RD$ = SPACE$(40)
    CALL IBRD(DEV%, RD$)
    PTOFF! = VAL(RD$)

’
’    X axsis unit, Y axsis unit, date, time
’
    ON ERROR GOTO ERRHANDLER

    OPEN FLNAME$ FOR OUTPUT AS #1
      WRITE #1, ”sec”, ”Volts”, DATE$, TIME$
’
’Scaling method
’          Time[i] = (i – PT_OFF) * XINCR
’          Volts[i] = (point value – YOFF) * YMULT + YZERO
’
      FOR I% = 0 TO NRPT% – 1
        TTT! = (I% – PTOFF!) * XINCR!
        VOLTS! = (WFM%(I%) – YOFF!) * YMULT! + YZERO!
        PRINT #1, TTT!; ”,”; VOLTS!
      NEXT I%
    CLOSE #1

    ON ERROR GOTO 0
    PRINT NRPT%; ”points data is written to”; FLNAME$

END SUB
```

```
SUB GPIB2ISF (DEV%, FLNAME$)
'
'Request to send Preamble and waveorm data
'
   CALL IBWRT(DEV%, "WAVFRM?")
'
'Read the waveform data and write to file.
'The IBRDF transfer from GPIB to file directory.
'
'more file error check.(because IBRDF can't check the disk cache)
'
   ON ERROR GOTO ERRHANDLER
   OPEN FLNAME$ FOR OUTPUT AS #1
   CLOSE #1

   CALL IBRDF(DEV%, FLNAME$)
   IF IBSTA% AND &H8000 THEN
     BEEP
     PRINT "Error on writing data."
     END
   ELSE
     PRINT IBCNTL&; "bytes data is written to "; FLNAME$
   END IF

   ON ERROR GOTO 0

END SUB
```

Example 2: Waveform Transfer #2

The second example illustrates a simple waveform transfer from the external controller to the instrument.

In case of QuickC

```
/*
 * putwfm.c – a simple waveform transfer program that restores waveform
 * to the instrument. The waveform must be one obtained with getwfm
 * program.
 */
# include <stdio.h>
# include ”decl.h”
# include ”exit.h”

void    checkarg();

char    *infile;        /* Output file descriptor                        */
char    *destination;   /* Destination from which a waveform is transferred */

main(argc, argv)
int     argc;
char    *argv[];
{
        printf("\n\n");
        printf("PUTWFM -  simple waveform transfer program.\n");
        printf("Copyright (c) 1993 Sony/Tektronix, Corp.");
        printf(" All Rights Reserved.\n\n");

        checkarg(argc, argv);   /* Check if arguments are valid.        */
        open_dev();             /* Find GPIB devices                    */
        DestSetup();            /* Define destination in the instrument */
        FtoGPIBwrite();         /* Read preamble and waveform, and
                                   write them to the instrument.        */
        close_dev();
}

/*
 * Check if the arguments are valid.
 */

void    checkarg(argc, argv)
int     argc;
char    *argv[];
{
```

```
/*
* Check command line argument count.
*/

        if(argc != 3)
        {
                fprintf(stderr, "usage: putwfm <destination> <in-file>\n");
                fprintf(stderr, "\twhere:\n");
                fprintf(stderr, "\t\t<destination>\t\tis the destination");
                fprintf(stderr, " waveform file to be written\n");
                fprintf(stderr, "\t\t<in-file>\tis the input file\n");
                exit(1);
        }

/*
 * Check for valid destination.
 */

        destination = argv[1];
        if(wfmfile(argv[1]) != 0)
        {
                fprintf(stderr, "ERROR: Invalid Destination: \"%s\":", argv[1]);
                exit(1);
        }
        infile = argv[2];
}

/*
 * Check if the file extension is '.wfm'.
 */

wfmfile(name)
char    *name;
{
        int     dlen = strlen(name);

        if (dlen < 4 || dlen > 12)
                return -1;
        if (strcmp(&name[dlen - 4], ".WFM") == 0 ||
            strcmp(&name[dlen - 4], ".wfm") == 0)
                return 0;
        return -1;
}

/*
 * Define destination to be written in the instrument.
 */
```

```
DestSetup()
{
        char cmd[100];

        sprintf(cmd, ":DATA:DESTINATION \"%s\"", destination);
        if(awgWrite(cmd) < 0)
        {
                gpiberr("Write Error: Unable to Setup waveform parameters");
                exit(1);
        }
}

/*
 * Read waveform and preamble from a file, and then write them
 * to the instrument.
 */

FtoGPIBwrite()
{
        if (WrtFtoG(infile) < 0)
        {
           gpiberr("Read Error/File Open Error:");
           exit(1);
        }
}
```

In case of Quick BASIC

```
DECLARE SUB CHKSTAT (DEV%, ESR%, EVENT$)
DECLARE SUB FINDDEV (KEYNAME$, DEV%)
DECLARE SUB EXTOPT (OPTION$, FLNAME$, DESTINATION$)
DECLARE FUNCTION DISKERR$ ()
DECLARE SUB ISF2GPIB (DEV%, FLNAME$)
'$INCLUDE: 'QBDECL.BAS'
PRINT
PRINT "PUTWFM Ver.1.0 "
PRINT "       Sample Program for AWG2000 series"
PRINT "                Copyright(C)1993,SONY/Tektronix Corp. Allright Reserved."
PRINT "             No warranty."
'
'Check COMMAND Arguments and extract source & filename
'
   OPTION$ = COMMAND$
   CALL EXTOPT(OPTION$, FLNAME$, DESTINATION$)
'
'GPIB Adress search
```

```
'
   KEYNAME$ = "SONY/TEK,AWG2"
   CALL FINDDEV(KEYNAME$, DEV%)
   PRINT KEYNAME$
   IF DEV% = 0 THEN BEEP: END
'
'Check file name.
'
   WRT$ = ":DATA:DESTINATION '" + DESTINATION$ + "'"
   CALL IBWRT(DEV%, WRT$)
   CALL CHKSTAT(DEV%, ESR%, EVENT$)
   IF ESR% <> 0 THEN
      BEEP
      PRINT "Error on file name."
      PRINT EVENT$
      END
   END IF
'
'Data transfer.
'
   CALL ISF2GPIB(DEV%, FLNAME$)
'
'Check GPIB Status
'
   DO
      CALL CHKSTAT(DEV%, ESR%, EVENT$)
      IF ESR% <> 0 THEN
        BEEP
        PRINT "Worning."
        PRINT EVENT$
      END IF
   LOOP UNTIL ESR% = 0

END
'
'ERROR Trap routine
'
ERRHANDLER:
   BEEP
   PRINT "ERROR. "; DISKERR$
   END
END
'————————————————————————————————————End of Main procedure
```

```
SUB CHKSTAT (DEV%, ESR%, EVENT$)

   CALL IBRSP(DEV%, sta%)
   CALL IBWRT(DEV%, ”*esr?”)
   RD$ = SPACE$(16)
   CALL IBRD(DEV%, RD$)
   ESR% = VAL(RD$)

   CALL IBWRT(DEV%, ”allev?”)
   RD$ = SPACE$(500)
   CALL IBRD(DEV%, RD$)
   EVENT$ = LEFT$(RD$, IBCNT% – 1)

END SUB

FUNCTION DISKERR$
   SELECT CASE ERR
      CASE 54
         DISKERR$ = ”Bad file mode”
      CASE 64
         DISKERR$ = ”Bad file name”
      CASE 52
         DISKERR$ = ”Bad name or number”
      CASE 25
         DISKERR$ = ”Device fault”
      CASE 57
         DISKERR$ = ”Device I/O error”
      CASE 24
         DISKERR$ = ”Device timeout”
      CASE 68
         DISKERR$ = ”Device unavailable”
      CASE 61
         DISKERR$ = ”Disk full”
      CASE 72
         DISKERR$ = ”Disk-media error”
      CASE 71
         DISKERR$ = ”Disk not ready”
      CASE 53
         DISKERR$ = ”File not found”
      CASE 62
         DISKERR$ = ”Input past end of file”
      CASE 76
         DISKERR$ = ”Path not found”
      CASE 75
         DISKERR$ = ”Path/File sccess error”
```

```
      CASE 70
         DISKERR$ = ”Permission denied”
      CASE 67
         DISKERR$ = ”Too many files”
      CASE ELSE
         DISKERR$ = ”???”
      END SELECT
END FUNCTION

SUB EXTOPT (OPTION$, FLNAME$, DESTINATION$)

   IF OPTION$ = ”” THEN GOTO DISPUSAGE
   OPTION$ = OPTION$ + ” ”
   FLNAME$ = ””
   DESTINATION$ = ””
’
’Extract string between spaces.
’
   FOR I% = 1 TO LEN(OPTION$)
      A$ = MID$(OPTION$, I%, 1)
      IF A$ = ” ” THEN EXIT FOR
      DESTINATION$ = DESTINATION$ + A$
   NEXT I%

   FOR J% = I% + 1 TO LEN(OPTION$)
      A$ = MID$(OPTION$, J%, 1)
      IF A$ = ” ” THEN EXIT FOR
      FLNAME$ = FLNAME$ + A$
   NEXT J%
’
’Check arguments
’
   IF INSTR(DESTINATION$, ”.WFM”) AND FLNAME$ <> ”” THEN EXIT SUB
   BEEP
   PRINT ”Invalid argument.”
DISPUSAGE:
   PRINT
   PRINT ”Usage:PUTWFM <destination> <filename>”
   PRINT
   PRINT ”       <destination>:Waveform filename to distination”
   PRINT ”              extention is ’.WFM’”
   PRINT
   PRINT ”       <filename>:Waveform file to transfer”
   PRINT ”              must be instument specified format.”
```

```
    PRINT
    PRINT "  This program read the waveform file form disk and send to the AWG2000."
    PRINT "  If same as <destination> is already exist in the memory of the AWG2000,"
    PRINT "  and if the file isn't locked, It's overwrited."

END

END SUB

SUB FINDDEV (KEYNAME$, DEV%)

    CALL IBFIND("GPIB0", BD%)
    IF BD% < 0 THEN
      KEYNAME$ = "'GPIB0' not found."
      DEV% = 0
      EXIT SUB
    END IF

    CALL IBFIND("DEV1", DEV%)
    IF DEV% <= 0 THEN
      KEYNAME$ = "'DEV1' not found."
      DEV% = 0
      EXIT SUB
    END IF
    CALL IBSRE(BD%, 0)
    CALL IBSRE(BD%, 1)

    V% = 11: CALL IBTMO(DEV%, V%)
    AD% = 0
'
'GPIB Address search
'
    DO
      CALL IBPAD(DEV%, AD%)
      CALL IBWRT(DEV%, "*IDN?")
      IF IBSTA% AND &H8000 THEN
         AD% = AD% + 1
      ELSE
         id$ = SPACE$(100): CALL IBRD(DEV%, id$)
         IF INSTR(id$, UCASE$(KEYNAME$)) THEN
            EXIT DO
         ELSE
            AD% = AD% + 1
            CALL IBCLR(DEV%)
         END IF
```

```
      END IF
      IF 30 < AD% THEN
         KEYNAME$ = ”Specified instrument not found.”
         DEV% = 0
         EXIT SUB
      END IF
   LOOP

   V% = 13: CALL IBTMO(DEV%, V%)
   KEYNAME$ = LEFT$(id$, IBCNT% – 1) + ”  (GPIB Address =” + STR$(AD%) + ”)”
   CALL IBWRT(DEV%, ”:DESE 255;*CLS”)

END SUB

SUB ISF2GPIB (DEV%, FLNAME$)
’
’Read waveform from file and transfer to GPIB.
’The IBWRTF function transfer from file to GPIB directry
’No problem on binary file.
’
’more file error check
’
   ON ERROR GOTO ERRHANDLER
   OPEN FLNAME$ FOR INPUT AS #1
   CLOSE #1

   CALL IBWRTF(DEV%, FLNAME$)
   IF IBSTA% < 0 THEN
     BEEP
     PRINT ”Error when transfer data.”
     END
   ELSE
     PRINT FLNAME$; ”is tarnsfered (at”; IBCNTL&; ”bytes).”
   END IF

   ON ERROR GOTO 0

END SUB
```

Example 3: Equation Transfer and Setting Up

The third example illustrates how to transfer and compile equation data, how to synchronize its termination, how to setup and turn on the output.

In case of QuickC

```
/*
 * equset.c – equation data processing program that writes and compiles
 * equation data, sets the instrument up, and turns the output on.
 */
# include     <stdio.h>
# include     <stdlib.h>
# include     ”decl.h”
# include     ”exit.h”

# define     CMD_LEN      100

typedef          long    LONG;

LONG    wpoints = 8000;
char    *equfile = "equsampl.equ";      /* Equation file to be created  */
char    *wfmfile = "equsampl.wfm";      /* Waveform file to be created  */
char    *equation =                     /* Equation data                */
"range(0,50ms)\n\
K0=100e-3\n\
K1=63.3e-9\n\
K2=K0*K1\n\
K3=10e-3\n\
exp(-t/K3)*sin(1/sqrt(K2)*t)\n\
range(51ms,100ms)\n\
exp(-t/K3)*sin(1/sqrt(K2)*t)\n\
range(101ms,150ms)\n\
exp(-t/K3)*sin(1/sqrt(K2)*t)\n\
range(151ms,200ms)\n\
exp(-t/K3)*sin(1/sqrt(K2)*t)\n\
range(201ms,250ms)\n\
exp(-t/K3)*sin(1/sqrt(K2)*t)\n\
norm()\n";

main()
{
        char    cmd[CMD_LEN + 1];

        open_dev();                     /*  Find GPIB devices           */
```

```
        printf("\n\n");
        printf("EQUSET  -  equation data processing program.\n");
        printf("Copyright (c) 1993 Sony/Tektronix Corp.");
        printf(" All Rights Reserved.\n\n");

        printf("Start processing ...\n\n");
        printf("Lock front panel controls\n");
        awgWrite(":LOCK  ALL");

/*
 * Process equation data
 */

        WriteCompEqu(); /* Write equation data and number of waveform
                           points and compile                          */
        WaveOutput();   /* Setup for output and turns output on        */

    /*
     * Clean up
     */

        printf("Recover front panel controls\n\n");
        awgWrite(":LOCK  NONE");
        close_dev();
}

/*
 * Write equation data and number of waveform points and compile
 */

WriteCompEqu()
{
        int     l;                      /* Size                        */
        char    cmd[CMD_LEN + 1];       /* Command buffer              */

        awgWrite("ABSTOUCH  EDIT");     /* Display EDIT screen         */

        /*
         * Check whether the file exists
         */
        if (awgWrite("*CLS ;:DESE 255 ;*ESE 16 ;*SRE 0") < 0)
        {                                       /* Set for serial poll  */
                gpiberr("Write Error:");
                exit(3);
        }
        sprintf(cmd, ":MEMORY:LOCK? \"%s\"", equfile);
        if (awgWrite(cmd) < 0)
```

```
        {
                gpiberr("Write Error:\n");
                exit(3);
        }
        if (!(serialp() & 0x20))        /* Check ESB bit in SRB */
        {
                awgtmo(T1s);            /* Wait further 100us   */
                awgwait(TIMO | SRQI | RQS | END);
                awgtmo(T10s);                   /* Reset to 10s                 */
                if (!(serialp() & 0x20))
                {
                        fprintf(stderr,
                                "Equation file (%s) already exists\n", equfile);
                        exit(3);
                }
        }

/*
 * Write Equation Data
 */
        if (awgWrite("*CLS ;:DESE 255 ;*ESE 1 ;*SRE 0") < 0)
        {                                       /* Set for serial poll  */
                gpiberr("Write Error:");
                exit(3);
        }
        printf("Write equation data\n");
        l = strlen(equation);
        eotcont(0);             /* Turns off sending terminator                 */
        sprintf(cmd, ":EQUATION:DEFINE \"%s\", #%d%d",
                                        equfile, DigitCount((LONG)l), l);
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Equation Definition Error:");
                exit(3);
        }
        eotcont(1);             /* Turns on sending terminator          */
        if (awgWrite(equation) < 0)
        {
                gpiberr("Equation Definition Error:");
                exit(3);
        }
```

```
/*
 * Write number of waveform points
 */

        printf("Write number of waveform points\n");
        sprintf(cmd, ":EQUATION:WPOINTS \"%s\",%ld", equfile, wpoints);
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Waveform Point Write Error:");
                exit(3);
        }

/*
 * Compile
 */

        printf("Start compiling...\n");
        sprintf(cmd, ":EQUATION:COMPILE \"%s\" ;*OPC", equfile);
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Equation Compile Command Write Error:");
                exit(3);
        }

/*
 * Wait termination by checking status byte.
 */

        printf("Wait its termination\n\n");
        while (!serialp())      /* Keep looping while serial_poll = 0    */
                ;

        awgWrite(":DESE 255 ;*ESE 0 ;*SRE 0");  /* Set back             */
}

/*
 * Set the instrument up for output, and turns output on.
 */

WaveOutput()
{
        int     l;                      /* Size                         */
        char    cmd[CMD_LEN + 1];       /* Command buffer               */

        awgWrite("ABSTOUCH  SETUP");            /* Display SETUP screen */
        awgWrite(":OUTPUT:CH1:STATE OFF");      /* Turns output off     */
```

```
/*
 * Set waveform to CH1
 */

        printf("Set waveform file (%s) to CH1\n", wfmfile);
        sprintf(cmd, "CH1:WAVEFORM  \"%s\"", wfmfile);
        if (awgWrite(cmd) < 0)
        {
                gpiberr("Write Error:");
                exit(4);
        }

/*
 * Set mode to triggered
 */

        printf("Set mode to triggered\n");
        if (awgWrite("MODE  TRIGGERED") < 0)
        {
                gpiberr("Write Error:");
                exit(4);
        }

/*
 * Set output parameters and ready to start output by trigger
 */

        printf("Set amplitude to 2.0V, and frequency to 20MHz\n\n");
        if (awgWrite(":CH1:AMPLITUDE 2.0V") < 0 ||
            awgWrite(":CLOCK:FREQUENCY  20MHz") < 0 ||
            awgWrite(":OUTPUT:CH1:STATE  ON") < 0)
        {
                gpiberr("Write Error:");
                exit(4);
        }
}

/*
 * Count digits
 */

DigitCount(n)
LONG    n;
{
        int     cc = 1;
```

```
        while (n = ldiv(n, 10L).quot)
                cc++;
        return cc;
}
```

In case of Quick BASIC

```
DECLARE SUB WAVEOUTPUT (DEV%, WFMFILE$)
DECLARE SUB WRITECOMPEQU (DEV%, WPOINTS&, EQUFILE$, EQUATION$)
DECLARE SUB CHKSTAT (DEV%, ESR%, EVENT$)
DECLARE SUB FINDDEV (KEYNAME$, DEV%)
'$INCLUDE: 'QBDECL.BAS'
PRINT
PRINT "EQUSET Ver.1.0 "
PRINT "       Sample Program for AWG2000 series"
PRINT "            Copyright(C)1993,SONY/Tektronix Corp. Allright Reserved."
PRINT "          No warranty."
'
'Define equation data and file names
'
   WPOINTS& = 8000          'number of waveform points
   EQUFILE$ = "EQUSAMPL.EQU"   'Equation file to created
   WFMFILE$ = EQUFILE$    'Waveform file to Created
   MID$(WFMFILE$, INSTR(WFMFILE$, ".EQU")) = ".WFM"
   EQUATION$ = ""          'Equation data

   EQUATION$ = EQUATION$ + "range(0,50ms)" + CHR$(10)
   EQUATION$ = EQUATION$ + "K0=100e–3" + CHR$(10)
   EQUATION$ = EQUATION$ + "K1=63.3e–9" + CHR$(10)
   EQUATION$ = EQUATION$ + "K2=K0*K1" + CHR$(10)
   EQUATION$ = EQUATION$ + "K3=10e–3" + CHR$(10)
   EQUATION$ = EQUATION$ + "exp(–t/K3)*sin(1/sqrt(K2)*t)" + CHR$(10)
   EQUATION$ = EQUATION$ + "range(51ms,100ms)" + CHR$(10)
   EQUATION$ = EQUATION$ + "exp(–t/K3)*sin(1/sqrt(K2)*t)" + CHR$(10)
   EQUATION$ = EQUATION$ + "range(101ms,150ms)" + CHR$(10)
   EQUATION$ = EQUATION$ + "exp(–t/K3)*sin(1/sqrt(K2)*t)" + CHR$(10)
   EQUATION$ = EQUATION$ + "range(151ms,200ms)" + CHR$(10)
   EQUATION$ = EQUATION$ + "exp(–t/K3)*sin(1/sqrt(K2)*t)" + CHR$(10)
   EQUATION$ = EQUATION$ + "range(201ms,250ms)" + CHR$(10)
   EQUATION$ = EQUATION$ + "exp(–t/K3)*sin(1/sqrt(K2)*t)" + CHR$(10)
   EQUATION$ = EQUATION$ + "norm()"
'
'Search GPIB Address
'
   KEYNAME$ = "SONY/TEK,AWG2"
```

```
   CALL FINDDEV(KEYNAME$, DEV%)
   PRINT KEYNAME$
   IF DEV% = 0 THEN BEEP: END
'
'Lock the front pannel controls
'
   PRINT "Processing..."
   PRINT "Lock front pannel controls."
   CALL IBWRT(DEV%, ":LOCK ALL")
'
'Write the equation data and number of points and compile
'
   CALL WRITECOMPEQU(DEV%, WPOINTS&, EQUFILE$, EQUATION$)
'
'Setup fot output and turns output on
'
   CALL WAVEOUTPUT(DEV%, WFMFILE$)
'
'Check GPIB Status
'
   DO
      CALL CHKSTAT(DEV%, ESR%, EVENT$)
      IF ESR% <> 0 THEN
        BEEP
        PRINT "Worning."
        PRINT EVENT$
      END IF
   LOOP UNTIL ESR% = 0
'
'UNLock front pannel controls
'
   PRINT "Recover front panel controls."
   CALL IBWRT(DEV%, ":LOCK NONE")
END
'————————————————————————————————End of Main procedure

SUB CHKSTAT (DEV%, ESR%, EVENT$)

   CALL IBRSP(DEV%, STB%)
   CALL IBWRT(DEV%, "*ESR?")
   RD$ = SPACE$(16)
   CALL IBRD(DEV%, RD$)
   ESR% = VAL(RD$)
```

```
   CALL IBWRT(DEV%, ”ALLEV?”)
   RD$ = SPACE$(500)
   CALL IBRD(DEV%, RD$)
   EVENT$ = LEFT$(RD$, IBCNT% – 1)

END SUB

SUB FINDDEV (KEYNAME$, DEV%)

   CALL IBFIND(”GPIB0”, BD%)
   IF BD% < 0 THEN
      KEYNAME$ = ”’GPIB0’ not found.”
      DEV% = 0
      EXIT SUB
   END IF

   CALL IBFIND(”DEV1”, DEV%)
   IF DEV% <= 0 THEN
      KEYNAME$ = ”’DEV1’ not found, Please run IBCONF and define.”
      DEV% = 0
      EXIT SUB
   END IF
   CALL IBSRE(BD%, 0)
   CALL IBSRE(BD%, 1)
’
’GPIB Address search
’
   V% = 11: CALL IBTMO(DEV%, V%)
   AD% = 0
   DO
      CALL IBPAD(DEV%, AD%)
      CALL IBWRT(DEV%, ”*IDN?”)
      IF IBSTA% AND &H8000 THEN
        AD% = AD% + 1
      ELSE
        ID$ = SPACE$(100): CALL IBRD(DEV%, ID$)
        IF INSTR(ID$, UCASE$(KEYNAME$)) THEN
            EXIT DO
        ELSE
            AD% = AD% + 1
            CALL IBCLR(DEV%)
        END IF
      END IF
      IF 30 < AD% THEN
```

```
        KEYNAME$ = ”Specified instrument not found.”
        DEV% = 0
        EXIT SUB
      END IF
   LOOP

   V% = 13: CALL IBTMO(DEV%, V%)
   KEYNAME$ = LEFT$(ID$, IBCNT% – 1) + ”  (GPIB¿ Address =” + STR$(AD%) + ”)”
   CALL IBWRT(DEV%, ”:DESE 255;*CLS”)

END SUB

SUB WAVEOUTPUT (DEV%, WFMFILE$)
’
’Dispaly SETUP screen to see the operation
’
   CALL IBWRT(DEV%, ”ABSTOUCH SETUP”)
’
’Turns output off
’
   CALL IBWRT(DEV%, ”OUTPUT:CH1:STATE OFF”)
’
’Set waveform file to CH1
’
   PRINT ”Set the ”; WFMFILE$; ” to CH1.”
   WRT$ = ”CH1:WAVEFORM ’” + WFMFILE$ + ”’”
   CALL IBWRT(DEV%, WRT$)
’
’Set mode to triggered
’
   PRINT ”Set mode to triggered.”
   CALL IBWRT(DEV%, ”MODE TRIGGERED”)
’
’Set output parameters and turns output on
’
   PRINT ”Set amplitude to 2.0V, and frequency to 20MHz.”
   CALL IBWRT(DEV%, ”CH1:AMPLITUDE 2.0V”)
   CALL IBWRT(DEV%, ”CLOCK:FREQUENCY 20MHz”)
   CALL IBWRT(DEV%, ”OUTPUT:CH1:STATE ON”)

END SUB

SUB WRITECOMPEQU (DEV%, WPOINTS&, EQUFILE$, EQUATION$)
’
’Check file name to transfer
’
```

```
   CALL IBWRT(DEV%, ”ABSTOUCH EDIT”)’Display EDIT screen to see operation

   CALL IBWRT(DEV%, ”:MEMORY:CATALOG:EQU?”)
   RD$ = SPACE$(5000)
   CALL IBRD(DEV%, RD$)
   IF INSTR(RD$, UCASE$(EQUFILE$)) THEN
      BEEP
      PRINT EQUFILE$; ”is already exist.”
      INPUT ”overwrite(y/[n])”; SURE$
      IF UCASE$(SURE$) <> ”Y” THEN
        CALL IBWRT(DEV%, ”UNLOCK ALL”)
        END
      END IF
   END IF
’
’Write the equation data
’
   EQULENGTH$ = LTRIM$(RTRIM$(STR$(LEN(EQUATION$))))
   DIGCOUNT$ = ”#” + LTRIM$(RTRIM$(STR$(LEN(EQULENGTH$)))) + EQULENGTH$
   WRT$ = ”:EQUATION:DEFINE ’” + EQUFILE$ + ”’,” + DIGCOUNT$ + EQUATION$
   CALL IBWRT(DEV%, WRT$)
   CALL CHKSTAT(DEV%, ESR%, EVENT$)
   IF ESR% <> 0 THEN
      BEEP
      PRINT ”Error on write the equation data.”
      PRINT EVENT$
      CALL IBWRT(DEV%, ”UNLOCK ALL”)
      END
   END IF
’
’Write the number of points
’
   WRT$ = ”:EQUATION:WPOINTS ’” + EQUFILE$ + ”’,” + STR$(WPOINTS&)
   CALL IBWRT(DEV%, WRT$)
   CALL CHKSTAT(DEV%, ESR%, EVENT$)
   IF ESR% <> 0 THEN
      BEEP
      PRINT ”Error on write the number of points.”
      PRINT EVENT$
      CALL IBWRT(DEV%, ”UNLOCK ALL”)
      END
   END IF
’
’Set the event report resisters to know the operation complete
```

```
’Not use the SRQ and check by serial polling.
’
   CALL IBWRT(DEV%, ”:DESE 255;*ESE 1;*SRE 0;*CLS”)
’
’Compile
’
   PRINT ”Compile in progress...”
   WRT$ = ”EQUATION:COMPILE ’” + EQUFILE$ + ”’;*OPC”
   CALL IBWRT(DEV%, WRT$)
’
’Wait to the operation complete
’
   DO
      CALL IBRSP(DEV%, STB%)
   LOOP UNTIL STB%
’
’Check the standard event status resister, if it’s 1, that’s OK but...
’
   CALL CHKSTAT(DEV%, ESR%, EVENT$)
   IF ESR% <> 1 AND ESR% <> 0 THEN
      BEEP
      PRINT ”Error at Compiling.”
      PRINT EVENT$
      CALL IBWRT(DEV%, ”UNLOCK ALL”)
      END
   END IF

END SUB
```

Example 4: Interactive Communication

The fourth example illustrates interactive communication method between the external controller and the instrument. In this program, sending GPIB commands, reading from the output queue, controlling event/status, and etc. are shown.

In case of QuickC

```
/*
 * intrv.c – interactive communication program between the external
 * controller and the instrument.
 */
# include     <stdio.h>
# include     ”decl.h”

# define      MAX_BUF       128
# define      MAX_ARG       10
# define      FILE_LEN      15
# define      RD0           0
# define      RD1           1
# define      RD2           2
# define      RD3           3
# define      ON            1
# define      OFF           –1
# define      ERROR         –1
# define      NORMAL        1
# define      QUERY         ’?’
# define      NOQUERY       ’N’

extern  int     iostatus;             /* Same as ibsta                */

void    viewfile();
void    execfile();
void    helpmessg();
void    process();
void    bnull();
void    redirect();
void    chkstatus();
void    ReadOutputbuf();
char    *getfile();

int     argc;
char    *argv[MAX_ARG];
char    readbuf[MAX_BUF + 1];
char    replace[MAX_BUF + 1];
```

```
char    assmble[MAX_BUF + 1];
char    stack[MAX_BUF + 1];

FILE    *ifd;           /* Input file descriptor                        */
FILE    *ofd;           /* Output file descriptor                       */
char    *rfile;
int     rflag;          /* > : RD1, >> : RD2,  < : RD3, none : RD0      */
int     ropnum;
int     sflag;

main()
{
        open_dev();                      /* Fing GPIB devices            */

        printf("\n\n");
        printf("INTRV  -  interactive communication program.\n");
        printf("Copyright (c) 1993 Sony/Tektronix Corp.");
        printf(" All Rights Reserved.\n");
        printf("Type 'help' to display help messages.\n");
        printf("Type 'q' or 'quit' to exit from the program.\n\n");

        printf("Start interactive communication\n\n");
        process();                      /* Communication process        */
        close_dev();
}

/*
 * Communication process.
 *
 */

void    process()
{
        /*
         * Initialize
         */
        stack[0] = '\0';
        replace[0] = '\0';
        ofd = stdout;           /* Output to standard output            */
        ifd = stdin;            /* Input from standard input            */
        resets(1);             /* Set enable registers                         */

/*
 * Start interactive communication
 */
```

```
        for (;;)
        {
                chkstatus();    /* Check status byte, and dequeue events
                                                        if exists */
                tcerider();     /* Reset ofd to stdout                  */

                /*
                 * Get and parse command line
                 */

                inputline();

                /*
                 * Terminate interactive process
                 */

                if (strcmp(argv[0], "q") == 0 || strcmp(argv[0], "quit") == 0)
                        break;  /* Terminate interactive process        */

                /*
                 * Redirect
                 */

                if (rflag == RD1 || rflag == RD2)
                        redirect();

/*
 * Execute built-in command
 */

                if (strcmp(argv[0], "exec") == 0)
                        execfile(argc, argv[1]);
                else if (strcmp(argv[0], "view") == 0)
                        viewfile(argc, argv[1]);
                else if (strcmp(argv[0], "help") == 0)
                        helpmessg(argc);
                else if (strcmp(argv[0], "status") == 0)
                        statusbyte(argc);
                else if (strcmp(argv[0], "resetes") == 0)
                        resetes(argc);

/*
 * Transmit GPIB command or query. If transmitting a query, the output
 * queue is immediately read.
 */
```

```
                else if (argc > 0)
                {
                        if (rflag == RD3)
                        {
                                fprintf(stderr,
                                         "syntax error: (%s)\n", assmble);
                        }
                        else if (strcmp(assmble, "?") == 0)
                        {
                                ReadOutputbuf();
                        }
                        else
                        {
                                if (awgWrite(assmble) < 0)
                                {
                                        gpiberr("Write Error:");
                                        continue;
                                }
                                if (chkquery(assmble) == QUERY)
                                        ReadOutputbuf();
                        }
                }
                else if (rflag == RD3)
                {
                        WrtFtoG(rfile);
                }
        }
}

/*
 * Input command line is parsed and stored into following memory.
 *  *argv[]   – command and arguments delimited by space.
 *  argc      – divided count.
 *  assemble  – concatenation of *argv[].
 *  rflag     – '>', '>>', '<', or none.
 *  rfile     – input or output file to be redirected.
 *
 * If the string '!!' exists in command line, it is replaced with
 * previous command line.
 */

inputline()
{
        int     i;              /* loop index           */
```

```
        for (;;)
        {
                printf("AWG2020 >> ");
                if (fgets(readbuf, MAX_BUF, ifd) == NULL) /* Read one line */
                {
                        if (ifd == stdin)
                        {
                                fprintf(stderr, "Detected system error:");
                                fprintf(stderr, "restart the program\n");
                                exit (1);
                        }
                        fclose(ifd);
                        ifd = stdin;
                        continue;
                }
                if (ifd != stdin)
                        printf("%s", readbuf);
                chkreplace(readbuf);
                setarg(replace);                        /* Parse input line */
                if (!(argc < 1 && rflag == RDO))        /* Check input      */
                        break;
        }

        assmble[0] = '\0';
        for (i = 0; i < argc; i++)                      /* Assemble line    */
        {
                strcat(assmble, argv[i]);
                if ((i+1) < argc)
                        strcat(assmble, " ");
        }
        strcpy(stack, assmble);
        if (rflag != RDO)
        {
                strcat(stack, " ");
                strcat(stack, (rflag == RD1)?">":(rflag == RD2)?">>":"<");
                strcat(stack, " ");
                strcat(stack, rfile);
        }
}

/*
 * Check '!!' and replace if exists
 */
```

```
chkreplace(s)
char    *s;
{
        char    *p = stack;
        char    *r = replace;
        int     cc = 0;                 /* flag: replaced or not        */

/*
 * replace !! with previous input line
 */

        while (*s)
        {
                if (strncmp(s, "!!", 2) == 0)
                {
                        for (p = stack; *p;)
                                *r++ = *p++;
                        s++; s++; p= stack;
                        cc++;
                }
                else
                        *r++ = *s++;
        }
        *r = '\0';

        if (cc != 0)
                printf("%s", replace);
}

/*
 * Check if '?' is included in input line
 */

chkquery(s)
char    *s;
{
        for (; *s; s++)
                if (*s == '?')
                        return QUERY;   /* May be query or query is
                                                   included in a line       */
        return NOQUERY;                         /* Set command(s) only      */
}

/*
 * Read from output queue, and write stdin or file
 */
```

```
void    ReadOutputbuf()
{
        FILE    *tfd;
        char    *p;
        char    sc;             /* Store one character              */
        int     i;              /* Loop index                       */

/*
 * Check MAV bit in SBR
 */

        if (!(serialp() & 0x10))
        {
                awgtmo(T1s);                    /* Wait further 100us   */
                awgwait(TIMO | SRQI | RQS | END);
                awgtmo(T10s);                   /* Reset to 10s                 */
                if (!(serialp() & 0x10))
                {
                        fprintf(stderr,
                                 "Nothing to take out in Output Queue!!\n");
                        return;
                }
        }

/*
 * Take out
 */

        if (rflag == RD1 || rflag == RD2)
        {
                for (;;)
                {
                        if (awgRead(readbuf, MAX_BUF) < 0)
                        {
                                gpiberr("Read Error:");
                                break;
                        }
                        for (p = readbuf, i = 0; i < ibcnt; i++)
                                putc(*p++, ofd);
                        if (iostatus & (ERR | TIMO | END))
                                break;
                }
        }
        else                    /* Read Output Queue, Print to stdout   */
        {
```

```
                sc = ' ';
                for (;;)
                {
                        if (awgRead(readbuf, MAX_BUF) < 0)
                        {
                                gpiberr("Read Error:");
                                break;
                        }
                        for (p = readbuf, i = 0; i < ibcnt; i++, p++)
                        {
                                if (*p == ';')
                                {
                                        sc = *p;
                                }
                                else
                                {
                                        if (*p == ':' && sc == ';')
                                                putc('\n', ofd);
                                        else if (sc == ';')
                                                putc(sc, ofd);
                                        putc(*p, ofd);
                                        sc = *p;
                                }
                        }
                        if (iostatus & (ERR | TIMO | END))
                                break;
                }
        }
}

/*
 * Parse command line
 */

setarg(str)
char    *str;
{
        char    *s = str;
        char    *p = str;

        sflag = OFF;
        rflag = RDO;
        argc = 0;
        for (; *s; s++)
        {
```

```
switch ((int)*s)
{
case ' ' :
        sflag = OFF;
        bnull(p, s);
        break;
case '\n' :
        sflag = OFF;
        bnull(p, s);
        --s;
        break;
case '>' :
        ropnum = argc - 1;
        bnull(p, s);
        if ( *(s+1) == '>')
        {
                rflag = RD2;
                *++s = '\0';
        }
        else
        {
                rflag = RD1;
        }
        sflag = OFF;
        s = getfile(++s);
        s--;
        break;
case '<' :
        ropnum = argc - 1;
        bnull(p, s);
        if ( *(s+1) == '<')
                *++s = '\0';
        rflag = RD3;
        sflag = OFF;
        s = getfile(++s);
        s--;
        break;
default :
        if (sflag == OFF)
        {
                sflag = ON;
                argv[argc] = p = s;
                ++argc;
        }
```

```
                }
        }
}

/*
 * Put '\0' at the end of the command and arguments
 */


void    bnull(p, s)
char    *p, *s;
{
        *s-- = '\0';
        for (; p < s; s--)
        {
                if (*s == ' ')
                        *s = '\0';
                else
                        return;
        }
}

/*
 * Extract file name placed after '<', '>', or '>>'.
 */


char    *getfile(s)
char    *s;
{
        for (; *s == ' '; s++)
                ;
        rfile = s;
        for (;*s && *s != '\n' && *s != ' ' && *s != '>' && *s != '<'; s++)
                ;
        *s = '\0';
        return ++s;
}

/*
 * 'view' built-in command.
 */

void    viewfile(n, name)
int     n;
char    *name;
```

```
{
        FILE    *tfd;
        int     c;

        if (n != 2)
        {
                fprintf(stderr, "usage:  view  file-name\n");
                return;
        }

        if ((tfd = fopen(name, "r")) == NULL)
        {
                fprintf(stderr, "can't open file (%s)\n", name);
                return;
        }
        while ((c = getc(tfd)) != EOF)
                putc(c, ofd);
        fclose(tfd);
}

/*
 * 'exec' built-in command.
 */

void    execfile(n, name)
int     n;
char    *name;
{
        FILE    *tfd;

        if (n != 2)
        {
                fprintf(stderr, "usage: exec  file-name\n");
                return;
        }

        if ((tfd = fopen(name, "r")) == NULL)
        {
                fprintf(stderr, "can't open file (%s)\n", name);
                return;
        }
        ifd = tfd;
}
```

```
/*
 * 'status' built-in command
 */

statusbyte(n)
int     n;
{
        if (n != 1)
                fprintf(stderr, "Arguments are neglected!!\n\n");
        fprintf(ofd, "Status byte: (%X)h\n", serialp());
}

static  char    hmessg[] = {"\n\

/*
 * 'help' built-in command.
 */

void    helpmessg(n)
int     n;
{
        char    *p = hmessg;

        if (n != 1)
        {
                fprintf(stderr, "Arguments are neglected!!\n\n");
        }
        while (*p)
                putc(*p++, ofd);
}

/*
 * 'resetes' built-in command
 */

resetes(n)
int     n;
{
        if (n != 1)
                fprintf(stderr, "Arguments are neglected!!\n\n");
        if (awgWrite("*CLS;:DESE 59;*ESE 58;*SRE 48") < 0)
        {
                gpiberr("Write Error: can't set enable registers");
        }
}
```

```
/*
 * Read event queue if event is being stacked.
 *
 * '\n' is placed after event code and event message.
 */

void    chkstatus()
{
        int     ccount = 0;
        int     i;
        char    *p;

/*
 * Check ESB bit in SBR
 */

        if (!(serialp() & 0x20))
        {
                awgtmo(T1s);                    /* Wait further 100us   */
                awgwait(TIMO | SRQI | RQS | END);
                awgtmo(T10s);                   /* Reset to 10s                 */
                if (!(serialp() & 0x20))
                        return;
        }

/*
 * Prepare to Take out
 */

        if (awgWrite("HEADER OFF;*ESR?") < 0 || awgWrite(":ALLEV?") < 0)
        {
                gpiberr("Write Error:");
                return;
        }

/*
 * Read Event Queue
 */

        for (;;)
        {
                if (awgRead(readbuf, MAX_BUF) < 0)
                {
                        gpiberr("Read Error:");
                        break;
                }
```

```
                for (p = readbuf, i = 0; i < ibcnt; i++, p++)
                {
                        putc(*p, ofd);
                        if (*p == ',' && ccount == 1)
                        {
                                putc('\n', ofd);
                                ccount = 0;
                        }
                        else if (*p == ',')
                                ccount++;
                }
                if (iostatus & (ERR | TIMO | END))
                        break;
        }
        if (awgWrite("HEADER ON") < 0)
        {
                gpiberr("Write Error:");
                return;
        }
}

/*
 * Open file for output.
 */

void    redirect()
{
        FILE    *tfd;           /* temporary file descriptor    */

        if (rflag == RD1)
        {
                if ((tfd = fopen(rfile, "w")) == NULL)
                {
                        fprintf(stderr, "can't open file (%s)\n", rfile);
                        return;
                }
        }
        else if (rflag == RD2)
        {
                if ((tfd = fopen(rfile, "a")) == NULL)
                {
                        fprintf(stderr, "can't open file (%s)\n", rfile);
                        return;
                }
        }
```

```
        ofd = tfd;
}

/*
 * Close file after redirection.
 */

tcerider()
{
        if (ofd != stdout)
        {
                fclose(ofd);
                ofd = stdout;
        }
}
```

Support Functions

The examples in this section use the support functions listed below.

```
/*
 * awglib.c  –  libraries of GPIB interfaces.
 */
# include      ”decl.h”

int     awgdev;         /* gpib descriptor of AWG              */
int     extcdev;        /* gpib descriptor of GPIB0            */
int     iostatus;       /* save a value of ibsta               */

/*
 * Find GPIB devices
 */

open_dev()
{

/*
 * Assign unique identifiers to the device DEV1 and to the board GPIB0,
 * store them in the variables “awgdev” and “extcdev”, respectively, and
 * check for errors. If DEV1 or GPIB0 is not defined, ibfind returns –1.
 */

        if((awgdev = ibfind("DEV1")) < 0 || (extcdev = ibfind("GPIB0")) < 0)
        {
                gpiberr("Ibfind Error: Unable to find device/board!");
                exit(0);
        }

/*
 *  Clear the device and check for errors.
 */

        if(ibclr(awgdev) < 0 || ibsre(extcdev, 0) < 0)
        {
                gpiberr("ibclr/ibsre Error: Unable to clear device/board!");
                exit(0);
        }

/*
 * Set up the Device Event Status Enable Register, Event Status Enable
 * Register, and Service Request Enable Register to enable status
 * events.
 */
```

```
        if (awgWrite("DESE 255") < 0 || awgWrite("*ESE 255") < 0 ||
            awgWrite("*SRE 48") < 0)
        {
                gpiberr("GPIBWRITE Error: Unable to Initialize Device!");
                exit(0);
        }
}

close_dev()
{
}

/*
 *  Read into the string from the device and wait for the
 *  read to finish.
 */

awgRead(resp, cnt)
char    *resp;
int     cnt;
{

/*
 * Set the timeout for 10 seconds, send the command, and
 * wait for the scope to finish processing the command.
 */

        ibtmo(awgdev, T10s);
        ibrd(awgdev, resp, cnt);
        iostatus = ibsta;
        resp[ibcnt] = '\0';

    /*
     * If ibwrt was successful, wait for completion.
     */

        if(ibsta >=0)
                ibwait(awgdev, CMPL);

        return ibsta;
}

/*
 *  Send the contents of the string to the device and wait
 *  for the write to finish.
 */
```

```
awgWrite(cmd)
char    *cmd;
{
        int     cnt = strlen(cmd);

/*
 * Set the timeout for 10 seconds, send the command
 * wait for the instrument to finish processing the command.
 */

        ibtmo(awgdev, T10s);
        ibwrt (awgdev, cmd, cnt);

/*
 * If ibwrt was successful, wait for completion.
 */

        if(ibsta >=0)
                ibwait(awgdev, CMPL);

        return ibsta;
}

/*
 * Read from GPIB device, and Write into a file.
 */

WrtGtoF(name)
char    *name;
{
        return ibrdf(awgdev, name);
}

/*
 * Read from a file, and write into GPIB device.
 */

WrtFtoG(name)
char    *name;
{
        return ibwrtf(awgdev, name);
}

/*
 * Get status byte.
 */
```

```
serialp()
{
        char    serial_poll = 0;

        ibrsp(awgdev, &serial_poll);
        return serial_poll & 0xff;
}

/*
 * Set time out
 */


awgtmo(tm)
int     tm;
{
        ibtmo(awgdev, tm);
}

/*
 * Wait
 */

awgwait(wt)
int     wt;
{
        ibwait(awgdev, wt);
}

/*
 * EOI control
 */

eotcont(status)
{
        ibeot(awgdev, status);
}

/*
 *  gpiberr.c – display error from defined error codes based on what
 *  is contained in ibsta. This routine would notify you that an IB
 *  call failed.
 */
#include ”decl.h”
#include <stdio.h>
```

```
void gpiberr(msg)
char    *msg;
{
        fprintf(stderr, "%s\n", msg);
        fprintf(stderr,  "ibsta=(%X)h  <", ibsta);

        if (ibsta & ERR ) fprintf(stderr, " ERR");
        if (ibsta & TIMO) fprintf(stderr, " TIMO");
        if (ibsta & END ) fprintf(stderr, " END");
        if (ibsta & SRQI) fprintf(stderr, " SRQI");
        if (ibsta & RQS ) fprintf(stderr, " RQS");
        if (ibsta & CMPL) fprintf(stderr, " CMPL");
        if (ibsta & LOK ) fprintf(stderr, " LOK");
        if (ibsta & REM ) fprintf(stderr, " REM");
        if (ibsta & CIC ) fprintf(stderr, " CIC");
        if (ibsta & ATN ) fprintf(stderr, " ATN");
        if (ibsta & TACS) fprintf(stderr, " TACS");
        if (ibsta & LACS) fprintf(stderr, " LACS");
        if (ibsta & DTAS) fprintf(stderr, " DTAS");
        if (ibsta & DCAS) fprintf(stderr, " DCAS");
        fprintf(stderr, " >\n");

        fprintf(stderr, "iberr= %d", iberr);
        if (iberr == EDVR) fprintf(stderr, " EDVR <DOS Error>\n");
        if (iberr == ECIC) fprintf(stderr, " ECIC <Not CIC>\n");
        if (iberr == ENOL) fprintf(stderr, " ENOL <No Listener>\n");
        if (iberr == EADR) fprintf(stderr, " EADR <Address error>\n");
        if (iberr == EARG) fprintf(stderr, " EARG <Invalid argument>\n");
        if (iberr == ESAC) fprintf(stderr, " ESAC <Not Sys Ctrlr>\n");
        if (iberr == EABO) fprintf(stderr, " EABO <Op. aborted>\n");

        if (iberr == ENEB) fprintf(stderr, " ENEB <No GPIB board>\n");
        if (iberr == EOIP) fprintf(stderr, " EOIP <Async I/O in prg>\n");
        if (iberr == ECAP) fprintf(stderr, " ECAP <No capability>\n");        if (iberr ==
EFSO) fprintf(stderr, " EFSO <File sys. error>\n");
        if (iberr == EBUS) fprintf(stderr, " EBUS <Command error>\n");
        if (iberr == ESTB) fprintf(stderr, " ESTB <Status byte lost>\n");
        if (iberr == ESRQ) fprintf(stderr, " ESRQ <SRQ stuck on>\n");


/*
if (iberr == ETAB) fprintf(stderr, " ETAB <Table Overflow>\n");
*/

}
```

```
/*
 * exit.h
 */
# define        exit(x)     {awgWrite(":HEADER ON;:UNLOCK ALL");exit(x);}
```

Appendices

Appendix A: Character Charts

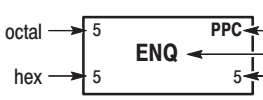
Table A–1: The AWG2000 Character Set

	0	1	2	3	4	5	6	7
0	NUL 0	16	space 32	0 48	@ 64	P 80	‘ 96	p 112
1	1	Ω 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	Δ 18	” 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	µ 22	& 38	6 54	F 70	V 86	f 102	v 118
7	‘ 7	23	’ 39	7 55	G 71	W 87	g 103	w 119
8	8	— 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	— 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	LF 10	∞ 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	± 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	CR 13	≠ 29	— 45	= 61	M 77	] 93	m 109	} 125
E	14	~ 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	● 15	31	/ 47	? 63	O 79	_ 95	o 111	rubout 127

Table A–2: ASCII & GPIB Code Chart

B7 B6 B5 BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	20 DLE 10 16	40 LA0 SP 20 32	60 LA16 0 30 48	100 TA0 @ 40 64	120 TA16 P 50 80	140 SA0 ` 60 96	160 SA16 p 70 112
0 0 0 1	1 GTL SOH 1 1	21 LL0 DC1 11 17	41 LA1 ! 21 33	61 LA17 1 31 49	101 TA1 A 41 65	121 TA17 Q 51 81	141 SA1 a 61 97	161 SA17 q 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 LA2 " 22 34	62 LA18 2 32 50	102 TA2 B 42 66	122 TA18 R 52 82	142 SA2 b 62 98	162 SA18 r 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 LA3 # 23 35	63 LA19 3 33 51	103 TA3 C 43 67	123 TA19 S 53 83	143 SA3 c 63 99	163 SA19 s 73 115
0 1 0 0	4 SDC EOT 4 4	24 DCL DC4 14 20	44 LA4 $ 24 36	64 LA20 4 34 52	104 TA4 D 44 68	124 TA20 T 54 84	144 SA4 d 64 100	164 SA20 t 74 116
0 1 0 1	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 LA5 % 25 37	65 LA21 5 35 53	105 TA5 E 45 69	125 TA21 U 55 85	145 SA5 e 65 101	165 SA21 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 LA6 & 26 38	66 LA22 6 36 54	106 TA6 F 46 70	126 TA22 V 56 86	146 SA6 f 66 102	166 SA22 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 LA7 ' 27 39	67 LA23 7 37 55	107 TA7 G 47 71	127 TA23 W 57 87	147 SA7 g 67 103	167 SA23 w 77 119
1 0 0 0	10 GET BS 8 8	30 SPE CAN 18 24	50 LA8 (28 40	70 LA24 8 38 56	110 TA8 H 48 72	130 TA24 X 58 88	150 SA8 h 68 104	170 SA24 x 78 120
1 0 0 1	11 TCT HT 9 9	31 SPD EM 19 25	51 LA9) 29 41	71 LA25 9 39 57	111 TA9 I 49 73	131 TA25 Y 59 89	151 SA9 i 69 105	171 SA25 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 LA10 * 2A 42	72 LA26 : 3A 58	112 TA10 J 4A 74	132 TA26 Z 5A 90	152 SA10 j 6A 106	172 SA26 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 LA11 + 2B 43	73 LA27 ; 3B 59	113 TA11 K 4B 75	133 TA27 [5B 91	153 SA11 k 6B 107	173 SA27 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 LA12 , 2C 44	74 LA28 < 3C 60	114 TA12 L 4C 76	134 TA28 \ 5C 92	154 SA12 l 6C 108	174 SA28 ¦ 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 LA13 – 2D 45	75 LA29 = 3D 61	115 TA13 M 4D 77	135 TA29] 5D 93	155 SA13 m 6D 109	175 SA29 } 7D 125
1 1 1 0	16 SO E 14	36 RS 1E 30	56 LA14 . 2E 46	76 LA30 > 3E 62	116 TA14 N 4E 78	136 TA30 ^ 5E 94	156 SA14 n 6E 110	176 SA30 ~ 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 LA15 / 2F 47	77 UNL ? 3F 63	117 TA15 O 4F 79	137 UNT – 5F 95	157 SA15 o 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY



Tektronix
REF: ANSI STD X3.4-1977
IEEE STD 488.1-1987
ISO STD 646-2973

Appendix B: Reserved Words

The words in the following list are reserved words for use with the AWG2000 Series Arbitrary Waveform Generator.

*CAL
*CLS
*ESE
*ESR
*IDN
*LRN
*OPC
*OPT
*PSC
*RST
*SRE
*STB
*TRG
*TST
*WAI
ABSTouch
ALL
ALLEv
ALOad
AMPLitude
AOFF
AST
AUTOStep
BIT_NR
BN_FMT
BRIGhtness
BYT__NR
BYT_OR
CATalog
CDIRectory
CH<x>
CH1
CH2
CLK
CLOCk
COMMent
COMPile
CONFigure
COPY
CRVCHK
CURVe
DATA
DATE
DEBug
DEFine
DELAy
DELete
DESE
DESTination
DIAG
DIRectory
DISK
DISPlay
DIVider
DWELl
ENCDG
EQU
EQUATion
EQUAtion
EVENT
EVMsg
EVQty
EXPAnd
FACTory
FG
FILTer
FORMat
FREE
FREQuency
HCOPy
HEADer
HIGH
ID
IMPedance
INVerted
LEVel
LOAD
LOCk
LOW
MARKer
MARKER<x>
MARKERLEVEL1
MARKERLEVEL2
MDIRectory
MEMory
MENU
MESSage
MMEMory
MODE
MSIS
NORMal
NR_PT
OFFSet
OPERation
ORDer
OUTPut
POINt
POLarity
PORT
PT_FMT
PT_OFF
REName
RESULt
RESUlt
RUNNing
SAVE
SECUre
SELect
SELFcal
SEQ
SEQUence
SETUp
SHAPe
SHOW
SLOpe
SNOop
SOURce
STARt
STATe
STOP
SWEep
SYNC
TIME
TRACk
TRIGger
TYPE
UNLock
UPTime
VERBose
WAVeform
WAVFrm
WFID
WFM
WFMPre
WIDTh
WPOints
XINCR
XUNIT
XZERO
YMULT
YOFF
YUNIT
YZERO

Appendix C: Interface Specification

This appendix lists and describes the GPIB functions and messages that the AWG2000 Series Arbitrary Waveform Generator implements.

Interface Functions

Table C–1 shows which GPIB interface functions are implemented in this instrument. Following the table is a brief description of each function.

Table C–1: GPIB Interface Function Implementation

Interface Function	Implemented Subset	Capability
Acceptor Handshake (AH)	AH1	Complete
Source Handshake (SH)	SH1	Complete
Listener (L)	L4	Basic Listener Unaddress if my talk address (MTA) No talk only mode
Talker (T)	T5	Basic Talker, Serial Poll Unaddress if my-listen-address (MLA)
Device Clear (DC)	DC1	Complete
Remote/Local (RL)	RL1	Complete
Service Request (SR)	SR1	Complete
Parallel Poll (PP)	PP0	None
Device Trigger (DT)	DT1	Complete
Controller (C)	C0	None
Electrical Interface	E2	Three-state driver

- Accepter Handshake (AH). Allows a listening device to help coordinate the the proper reception of data. The AH function holds off initiation or termination of a data transfer until the listening device is ready to receive the next data byte.
- Source Handshake (SH). Allows a talking device to help coordinate the proper transfer of data. The SH function controls the initiation and termination of the transfer of data bytes.

- Listener (L). Allows a device to receive device-dependent data over the interface. This capability exists only when the device is addressed to listen. This function uses a one-byte address.
- Talker (T). Allows a device to send device-dependent data over the interface. This capability exists only when the device is addressed to talk. The function uses a one-byte address.
- Device Clear (DC). Allows a device to be cleared or initialized, either individually or as part of a group of devices.
- Remote/Local (RL). Allows a device to select between two sources for operating control. This function determines whether input information from the front panel controls (local) or GPIB commands (remote) control the waveform generator.
- Service Request (SR). Allows a device to request service from the controller.
- Controller (C). Allows a device with the capability to send the device address, universal commands, and addressed commands to other device over the interface to do so.
- Electrical Interface (E) Identifies the type of the electrical interface. The notation E1 indicates the electrical interface uses open collector drivers, while E2 indicates the electrical interface uses three-state drivers.

Interface Messages

Table C–2 lists the GPIB Universal and Addressed commands that the AWG2000 Series Arbitrary Waveform Generator implements. A brief description of each function follows the table.

Table C–2: GPIB Interface Messages

Interface Message	Implemented
Device Clear (DC)	Yes
Local Lockout (LLO)	Yes
Serial Poll Disable (SPD)	Yes
Serial Poll Enable (SPE)	Yes
Parallel Poll Unconfigure (PPU)	No
Go To Local (GTL)	Yes
Selected Device Clear (SDC)	Yes
Group Execute Trigger (GET)	Yes

Table C–2: GPIB Interface Messages (Cont.)

Interface Message	Implemented
Take Control (TCT)	No
Parallel Poll Configure (PPC)	No

- Device Clear (DCL). Clears (initializes) all devices on the bus that have a device clear function, whether the controller has addressed them or not.
- Local Lockout (LLO). Disables the return to local function.
- Serial Poll Enable (SPE). Puts all devices on the bus, that have a service request function, into the serial poll enabled state. In this state, each device sends the controller its status byte, instead of the its normal output, after the device receives its talk address on the data lines. This function may be used to determine which device sent a service request.
- Serial Poll Disable (SPD). Changes all devices on the bus from the serial poll state to the normal operating state.
- Go To Local (GTL). Causes the listen-addressed device to switch from remote to local (front-panel) control.
- Select Device Clear (SDC). Clears or initializes all listen-addressed devices.
- Group Execute Trigger (GET). Triggers all applicable devices and causes them to initiate their programmed actions.
- Take Control (TCT).Allows controller in charge to pass control of the bus to another controller on the bus.
- Parallel Poll Configure (PPC). Causes the listen-addressed device to respond to the secondary commands Parallel Poll Enable (PPE) and Parallel Poll Disable (PPD), which are placed on the bus following the PPC command. PPE enables a device with parallel poll capability to respond on a particular data line. PPD disables the device from responding to the parallel poll.

Appendix D: Factory Initialization Settings

The following table lists the commands affected by a factory initialization and their factory initialization settings.

Table D–1: Factory Initialized Settings

Header	Default Settings
Calibration & Diagnostic Commands	
DIAG:SELect	ALL
SELFcal:SELect	ALL
Display Commands	
DISPlay:BRIGhtness	70
DISPlay:CATalog:ORDER	NAME 1
DISPlay:CLOCk	0
DISPlay:MENU:SETUP:FORMat	GRAPHICS
DISPlay:MESSAGE:SHOW	GRAPHICS
FG Commands	
FG:CH<x>:AMPLitude	1.000
FG:CH<x>:OFFSet	0.000
FG:CH<x>:POLarity	NORMAL
FG:CH<x>:SHAPe	SINUSOID
FG:STATe	0
FG:FREQuency	2.500E+06(AWG2020/21) 200.0E+3(AWG2005) /200.000E+03(AWG2005 opt.05) /10.00000E+6(AWG2040)
Hardcopy Commands	
HCOPy:FORMat	BMP
HCOPy:PORT	DISK
Memory Commands	
DISK:FORMat:TYPE	HD3
MMEMory:MSIS	DISK
MMEMory:ALOad:MSIS	DISK
MMEMory:ALOad:STATe	0

Table D–1: Factory Initialized Settings (Cont.)

Header	Default Settings
Mode Commands	
CONFigure	MASTER(AWG2005)
MODE	CONTINUOUS
TRIGger:IMPedance	HIGH
TRIGger:LEVel	1.4
TRIGger:POLarity	POSITIVE
TRIGger:SLOPe	POSITIVE
Output Commands	
OUTPut:CH<x>:STATe	0(AWG2005/20/21)
OUTPut:CH1:INVerted:STATE	0(AWG2040)
OUTPut:CH1:NORMal:STATE	0(AWG2040)
OUTPut:SYNC	END
Setup Commands	
CLOCk:FREQuency	100.0E+06 (AWG2020/21)/10.00E+06(AWG2005) /10.00000E+06 (AWG2005 opt.05) /1.000000E+09(AWG2040)
CLOCk:SOURce	INTERNAL
CLOCk:CH2:DIVider	1(AWG2020/21)
CLOCk:SWEep:DWELl	1.000E-03(AWG2005 opt.05)
CLOCk:SWEep:MODE	SCONTINUOUS(AWG2005 opt.05)
CLOCk:SWEep:STATe	0(AWG2005 opt.05)
CLOCk:SWEep:TIME	1.000E+00(AWG2005 opt.05)
CLOCk:SWEep:TYPE	LINEAR(AWG2005 opt.05)
CLOCk:SWEep:FREQuency:STARt	1.00000E+06(AWG2005 opt.05)
CLOCk:SWEep:FREQuency:STOP	20.0000E+06 (AWG2005 opt.05)
CH1:MARKERLEVEL1:HIGH	2.0(AWG2040)
CH1:MARKERLEVEL1:LOW	0.0(AWG2040)
CH1:MARKERLEVEL2:HIGH	2.0(AWG2040)
CH1:MARKERLEVEL2:LOW	0.0(AWG2040)
CH:OPERation	NORMAL(AWG2005/20/21)
CH<x>:AMPLitude	1.000
CH<x>:FILTer	THRU

Table D–1: Factory Initialized Settings (Cont.)

Header	Default Settings
CH<x>:OFFSet	0.000
CH<x>:TRACk:AMPLitude	OFF(AWG2005)
CH<x>:TRACk:OFFSet	OFF(AWG2005)
CH<x>:WAVeform	" "
Status & Event Commands	
DESE	256
*ESE	0
*PSC	1
*SRE	0
System Commands	
DEBug:SNOop:STATe	0
DEBug:SNOop:DELAy:TIME	0.2
HEADer	1
VERBose	1
Waveform Commands	
DATA:DESTination	"GPIB.WFM"
DATA:ENCDG	RPBINARY
DATA:SOURce	"CH1"
DATA:WIDTh	2
WFMPre:ENCDG	BIN
WFMPre:BN_FMT	RP
WFMPre:BYT_NR	2
WFMPre:BIT_NR	12
WFMPre:BYT_OR	MSB
WFMPre:CRVCHK	NONE

Glossary & Index

Glossary

ASCII
Acronym for the American Standard Code for Information Interchange. Controllers transmit commands to the instrument using ASCII character encoding.

Address
A 7-bit code that identifies an instrument on the communication bus. The instrument must have a unique address for the controller to recognize and transmit commands to it.

BNF (Backus-Naur Form)
A standard notation system for command syntax diagrams. The syntax diagrams in this manual use BNF notation.

Controller
A computer or other device that sends commands to and accepts responses from the digitizing oscilloscope.

EOI
A mnemonic referring to the control line "End or Identify" on the GPIB interface bus. One of the two possible end-of-message terminators.

EOM
A generic acronym referring to the end-of-message terminator. The end-of-message terminator can be either an EOI or the ASCII code for line feed (LF).

GPIB
Acronym for General Purpose Interface Bus, the common name for the communications interface system defined in IEEE Std 488.

IEEE
Acronym for the Institute for Electrical and Electronic Engineers.

QuickC
A computer language (distributed by Microsoft) that is based on C.

Index

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W