AWG70000A Series Arbitrary Waveform Generators

Programmer Manual

077-0782-02

Tektronix

AWG70000A Series
Arbitrary Waveform Generators

Programmer Manual

www.tektronix.com

077-0782-02

Tektronix

Supports product software version 3.0 and above.

Contacting Tektronix

Tektronix, Inc.
14150 SW Karl Braun Drive
P.O. Box 500
Beaverton, OR 97077
USA

For product information, sales, service, and technical support:
- In North America, call 1-800-833-9200.
- Worldwide, visit www.tektronix.com to find contacts in your area.

Table of Contents

Getting started

Introduction

This programmer guide provides you with the information required to use Programmable Interface (PI) commands for remotely controlling your AWG. With this information you can write computer programs that perform functions such as setting the front-panel controls, selecting clock source, setting sampling rate, and exporting data for use in other programs.

In addition to the LAN electronic interface, your AWG is provided with a *TekVISA* GPIB-compatible interface, (referred to as the virtual GPIB interface).

The programmer guide is divided into the following major sections:

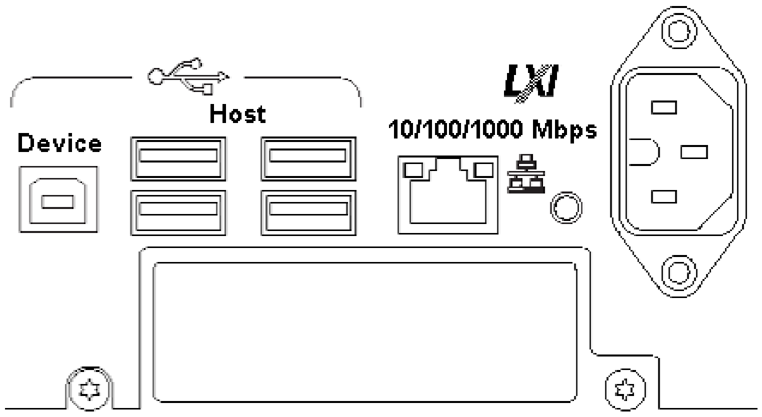
- Getting Started: provides basic information about setting up your AWG for remote control.
- Command Syntax: provides an overview of the command syntax that you will use to communicate with the AWG and other general information about commands, such as how commands and queries are constructed, how to enter commands, constructed mnemonics, and argument types.
- Command Groups: contains all the commands listed in functional groups. Each group consists of an overview of the commands in that group and a table that lists all the commands and queries for that group.
- Status and Events: discusses the status and event reporting system for the LAN interface. This system informs you of certain significant events that occur within the AWG.
- Appendices: contains miscellaneous information, such as LAN interface specifications that may be helpful when using remote commands to control the AWG.

Remote control

You can remotely control communications between your instrument and a PC via Ethernet or GPIB cables. Refer to the following sections describing the setups and connections required.

Ethernet control

If you are using Ethernet, start by connecting an appropriate Ethernet cable to the Ethernet port (RJ-45 connector) on the rear panel of the instrument. This connects the instrument to a 10BASE-T/100BASE-TX/1000BASE-T local area network.



The AWG accepts two types of Ethernet LAN connections:

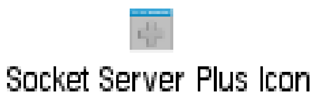
- VXI-11 Server: VXI-11 protocol is used through TekVISA. TekVISA is preinstalled on the instrument, but to use this protocol, TekVISA must also be installed on the remote controller (PC).
- Simple Raw Socket: Raw socket server using the Socket Server Plus application. The Socket Server Plus application is preinstalled on the instrument.

Raw Socket setup. The following steps to configure the Socket Server Plus application.

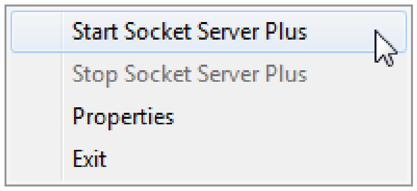
1. Enable the Socket Server Plus application.

   a. In the Windows notification area, click the arrow to show the hidden icons.

   b. Locate the icon representing the Socket Server Plus application.

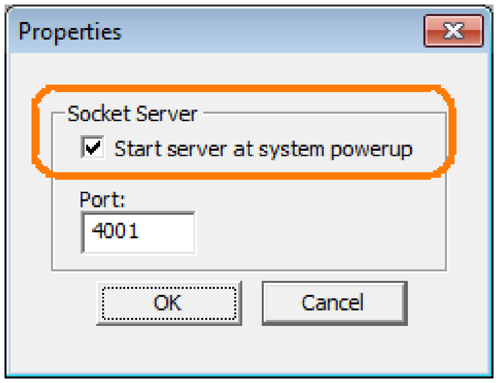
   

   c. Right click on the icon and select Start Socket Server Plus.

2. Set the properties for the Socket Server Plus application.

   a. Right click on the icon again and select Properties.

   
   Properties

   Socket Server

   ☑ Start server at system powerup

   Port: 4001

   b. In the Properties dialog box, set the Socket Server to enable (start) at system power up.

   c. The default Port is set to 4001. Typically, this does not need to change.

NOTE. *You can change the Port number as long as it doesn't conflict with other applications. For instance, TekVISA defaults to Port 4000 for communication.*

IP address. By default, the AWGs are specified to automatically acquire an IP address by DHCP. Refer to Windows documentation regarding network-related parameters.

GPIB control

The AWG has a USB 2.0 high-speed (HS) device port to control the instrument through USBTMC or GPIB with a TEK-USB-488 Adapter. The USBTMC protocol allows USB devices to communicate using IEEE488 style messages. This lets you run your GPIB software applications on USB hardware.

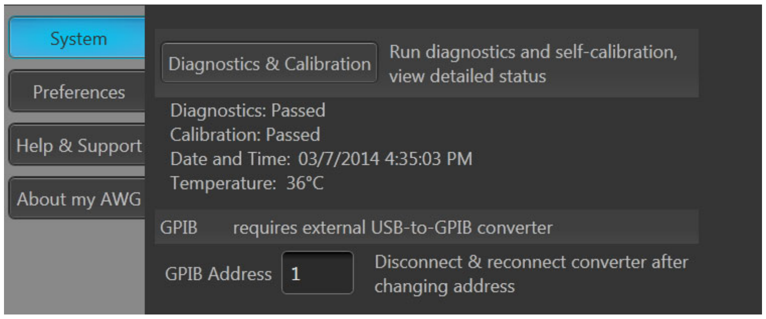
To use GPIB (General Purpose Interface Bus), start by connecting an appropriate USB cable to the USB 2.0 high-speed (HS) device port on the rear panel of the AWG. Connect the other end to the TEK-USB-488 Adapter host port. Then connect a GPIB cable from the TEK-USB-488 Adapter to your PC.

Before setting up the instrument for remote communication using the electronic (physical) GPIB interface, you should familiarize yourself with the following GPIB requirements:

- A unique device address must be assigned to each device on the bus. No two devices can share the same device address.
- No more than 15 devices can be connected to any one line.
- One device should be connected for every 6 feet (2 meters) of cable used.
- No more than 65 feet (20 meters) of cable should be used to connect devices to a bus.
- At least two-thirds of the devices on the network should be powered on while using the network.
- Connect the devices on the network in a star or linear configuration. Do not use loop or parallel configurations.

The default setting for the GPIB configuration is GPIB Address 1. If you need to change the GPIB address, do the following:

1. Display the Utilities screen and select System.
2. Set the GPIB address.
3. If the TEK-USB-488 adapter is connected to the instrument, disconnect and reconnect the adapter to ensure the new address is acquired.



Run diagnostics and self-calibration, view detailed status

Diagnostics: Passed
Calibration: Passed
Date and Time: 03/7/2014 4:35:03 PM
Temperature: 36°C

GPIB requires external USB-to-GPIB converter

GPIB Address 1

Disconnect & reconnect converter after changing address

Documentation

Review the following table to locate more information about this product.

To read about	Use these documents
Safety and Installation	Read the Safety and Installation manual for general safety information and proper instrument installation.
Operation and User Interface Help	Access the user help from the Help menu for information on controls and screen elements.
Programmer commands	Read the Programmer manual to learn the proper syntax of remote commands. This manual is available on the Tektronix Web site (www.Tektronix.com/manuals).
Specifications and Performance Verification procedures	Read the Technical Reference document for specifications and the performance verification procedures. This manual is available on the Tektronix Web site (www.Tektronix.com/manuals).
Service Procedures	Read the Service Manual to service the AWG to the module level. This manual is available on the Tektronix Web site (www.Tektronix.com/manuals).

Syntax and commands

Command syntax

Syntax overview

Control the operations and functions of the AWG through the LAN interface using commands and queries. The related topics listed below describe the syntax of these commands and queries. The topics also describe the conventions that the AWG uses to process them. See the Command Groups topic for a listing of the commands by command group or use the index to locate a specific command.

Refer to the following table for the symbols that are used.

Table 2-1: Syntax symbols and their meanings

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

Command and query structure

Overview

Commands consist of set commands and query commands (usually called commands and queries). Commands modify instrument settings or tell the instrument to perform a specific action. Queries cause the instrument to return data and status information.

Most commands have both a set form and a query form. The query form of the command differs from the set form by its question mark on the end. For example, the set command `AWGControl:RSTate` has a query form `AWGControl:RSTate?`. Not all commands have both a set and a query form. Some commands have only set and some have only query.

Messages

A command message is a command or query name followed by any information the instrument needs to execute the command or query. Command messages may contain five element types, defined in the following table.

Table 2-2: Message symbols and their meanings

Symbol	Meaning
<Header>	This is the basic command name. If the header ends with a question mark, the command is a query. The header may begin with a colon (:) character. If the command is concatenated with other commands, the beginning colon is required. Never use the beginning colon with command headers beginning with a star (*).
<Mnemonic>	This is a header subfunction. Some command headers have only one mnemonic. If a command header has multiple mnemonics, a colon (:) character always separates them from each other.
<Argument>	This is a quantity, quality, restriction, or limit associated with the header. Some commands have no arguments while others have multiple arguments. A <space> separates arguments from the header. A <comma> separates arguments from each other.
<Comma>	A single comma is used between arguments of multiple-argument commands. Optionally, there may be white space characters before and after the comma.
<Space>	A white space character is used between a command header and the related argument. Optionally, a white space may consist of multiple white space characters.

Commands

Commands cause the instrument to perform a specific function or change one of the settings. Commands have the structure:

```
[:]<Header>[<Space><Argument>[<Comma><Argument>]...]
```

A command header consists of one or more mnemonics arranged in a hierarchical or tree structure. The first mnemonic is the base or root of the tree and each subsequent mnemonic is a level or branch off the previous one. Commands at a higher level in the tree may affect those at a lower level. The leading colon (:) always returns you to the base of the command tree.

Queries

Queries cause the instrument to return status or setting information. Queries have the structure:

```
[:]<Header>?
```

```
[:]<Header>?[<Space><Argument>[<Comma><Argument>]...]
```

Clearing the instrument

Use the Device Clear (DCL) or Selected Device Clear (SDC) functions to clear the Output Queue and reset the instrument to accept a new command or query.

Command entry

Rules

The following rules apply when entering commands:

- You can enter commands in upper or lower case.
- You can precede any command with white space characters. White space characters include any combination of the ASCII control characters 00 through 09 and 0B through 20 hexadecimal (0 through 9 and 11 through 32 decimal).
- The instrument ignores commands consisting of any combination of white space characters and line feeds.

Abbreviating

You can abbreviate many instrument commands. Each command in this documentation shows the abbreviations in capitals. For example, enter the command `TRIGger:LEVel` simply as `TRIG:LEV`.

Concatenating

Use a semicolon (;) to concatenate any combination of set commands and queries. The instrument executes concatenated commands in the order received. When concatenating commands and queries, follow these rules:

1. Separate completely different headers by a semicolon and by the beginning colon on all commands except the first one. For example, the commands `TRIGger:IMPedance 50` and `SOURce:RMODe TRIGgered`, can be concatenated into the following single command:

   ```
   TRIGger:IMPedance 50;:RMODe TRIGgered
   ```

2. If concatenated commands have headers that differ by only the last mnemonic, you can abbreviate the second command and eliminate the beginning colon. For example, you can concatenate the commands `TRIGger:SOURCE EXTernal` and `TRIGger:SLOPe NEGative` into a single command:

   ```
   TRIGger:SOURce EXTernal; SLOPe NEGative
   ```

   The longer version works equally well:

   ```
   TRIGger:SOURCE EXTernal;:TRIGger:SLOPe NEGative
   ```

3. Never precede a star (*) command with a semicolon (;) or colon (:).

4. When you concatenate queries, the responses to all the queries are concatenated into a single response message. For example, if the high level of marker one of channel one is 1.0 V and the low level is 0.0 V, the concatenated query `SOURce1:MARKer1:VOLTage:HIGH?;` `SOURce1:MARKer1:VOLTage:LOW?` will return the following:

   1.0;0.0

5. Set commands and queries may be concatenated in the same message. For example, `TRIGger:SOURce EXTernal; SLOPe?` is a valid message that sets the trigger source to External. The message then queries the external trigger slope. Concatenated commands and queries are executed in the order received.

Terminating

This documentation uses <EOM> (end of message) to represent a message terminator.

Table 2-3: Message terminator and meaning

Symbol	Meaning
<EOM>	Message terminator

For messages sent to the instrument, the end-of-message terminator must be the END message (EOI asserted concurrently with the last data byte). The instrument always terminates messages with LF and EOI. It allows white space before the terminator. For example, it allows CR LF.

Parameter types

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

Table 2-4: Parameter types, their descriptions, and examples

Parameter type	Description	Example
Arbitrary block	A block of data bytes	#512234xxxxx... where 5 indicates that the following 5 digits (12234) specify the length of the data in bytes; xxxxx... indicates actual data or #0xxxxx...<LF><&EOI>
Boolean	Boolean numbers or values	ON or ≠ 0 OFF or 0
Discrete	A list of specific values	MINimum, MAXimum
NaN	Not a Number	9.91 ^{37}

Table 2-4: Parameter types, their descriptions, and examples (cont.)

Parameter type	Description	Example
NR1 numeric	Integers	0, 1, 15, –1
NR2 numeric	Decimal numbers	1.2, 3.141,–6.5
NR3 numeric	Floating point numbers	3.1415E+9
NRf numeric	Flexible decimal numbers that may be type NR1, NR2, or NR3	See NR1, NR2, and NR3 examples in this table
String	Alphanumeric characters (must be within quotation marks)	"Testing 1, 2, 3"

About MIN, MAX

You can also use MINimum and MAXimum keywords in the commands with the "Numeric" parameter. Set the minimum value or the maximum value using these keywords and query these values.

Block

Several instrument commands use a block argument form (see the following table).

Table 2-5: Block symbols and their meanings

Symbol	Meaning
<NZDig>	A nonzero digit character in the range of 1–9
<Dig>	<Dig> A digit character, in the range of 0–9
<DChar>	A character with the hexadecimal equivalent of 00 through FF (0 through 255 decimal) that represents actual data
<Block>	A block of data bytes defined as: <Block> ::={#<NZDig><Dig>[<Dig>...][<DChar>...] |#0[<DChar>...]<terminator>}

Arbitrary block

An arbitrary block argument is defined as:

#<NZDig><Dig>[<Dig>...][<DChar>...]

or

#0[<DChar>...]<terminator>

<NZDig> specifies the number of <Dig> elements that follow. Taken together, the <NZDig> and <Dig> elements form a decimal integer that specifies how many <DChar> elements follow.

#0 means that the <Block> is an indefinite length block. The <terminator> ends the block.

NOTE. *The AWGs do not support the indefinite format (a block starts with #0).*

Quoted string

Some commands accept or return data in the form of a quoted string, which is simply a group of ASCII characters enclosed by a single quote (') or double quote ("). For example: "this is a quoted string". This documentation represents these arguments as follows:

Table 2-6: String symbol and meaning

Symbol	Meaning
<QString >	Quoted string of ASCII text

A quoted string can include any character defined in the 7-bit ASCII character set. Follow these rules when you use quoted strings:

1. Use the same type of quote character to open and close the string. For example: "this is a valid string".
2. You can mix quotation marks within a string as long as you follow the previous rule. For example, "this is an 'acceptable' string".
3. You can include a quote character within a string simply by repeating the quote.

   For example: "here is a "" mark".
4. Strings can have upper or lower case characters.
5. A carriage return or line feed embedded in a quoted string does not terminate the string, but is treated as just another character in the string.
6. The maximum length of a quoted string returned from a query is 1000 characters.

Here are some invalid strings:

- "Invalid string argument' (quotes are not of the same type)
- "test<EOI>" (termination character is embedded in the string)

Units and SI prefix

If the decimal numeric argument refers to voltage, frequency, impedance, or time, express it using SI units instead of using the scaled explicit point input value format <NR3>. (SI prefixes are standardized for use in the International System of Units by the International Bureau of Weights and Measures.) For example, use the input format 200 mV or 1.0 MHz instead of 200.0E-3 or 1.0E+6, respectively, to specify voltage or frequency.

Omit the unit when you describe commands, but include the SI unit prefix. Enter both uppercase and lowercase characters. The following list shows examples of units you can use with the commands.

- V for voltage (V).
- HZ for frequency (Hz).
- OHM for impedance (ohm).
- S for time (s).
- DBM for power ratio.
- PCT for %.
- VPP for Peak-to-Peak Voltage (V p-p).
- UIPP for Peak-to-Peak, Unit is UI (UI p-p).
- UIRMS for RMS, Unit is UI (UIrms).
- SPP for Peak-to-Peak, Unit is second (s p-p).
- SRMS for RMS, Unit is second (srms).
- V/NS for SLEW's unit (V/ns).

The SI prefixes, which must be included, are shown in the following table. You can enter both uppercase and lowercase characters.

Table 2-7: SI prefixes and their indexes

SI prefix [1]	Corresponding power
EX	10^{18}
PE	10^{15}
T	10^{12}
G	10^{9}
MA	10^{6}
K	10^{3}

Table 2-7: SI prefixes and their indexes (cont.)

SI prefix [1]	Corresponding power
M	10^{-3}
U [2]	10^{-6}
N	10^{-9}
P	10^{-12}
F	10^{-15}
A	10^{-18}

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

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

Since M (m) can be interpreted as 1E-3 or 1E6 depending on the units, use mV for V, and MHz for Hz.

The SI prefixes need units.

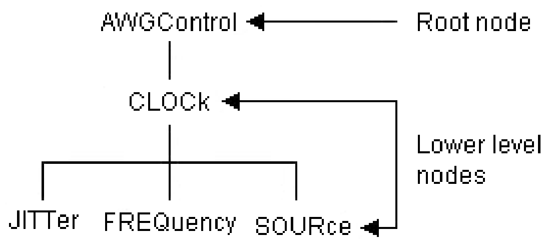
correct: 10MHz, 10E+6Hz, 10E+6

incorrect: 10M

SCPI commands and queries

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

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



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

Sequential, blocking, and overlapping commands

Programming commands (and queries) fall into three command type categories:

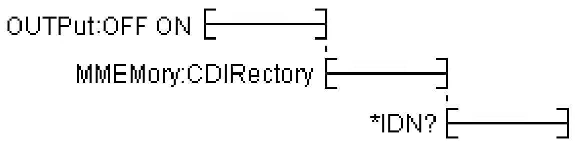
- Sequential
- Blocking
- Overlapping

The type of command is important to consider when programming since they could cause unexpected results if not handled correctly. See the following explanations and examples.

Sequential commands

Most of the programming commands for the AWG70000A Series are sequential type commands. This simply means a command will not start until the previous command has finished.

Following is an example of a series of sequential commands.



In normal operation, these commands could all be sent at once and they would be queued up and executed sequentially.

Blocking commands

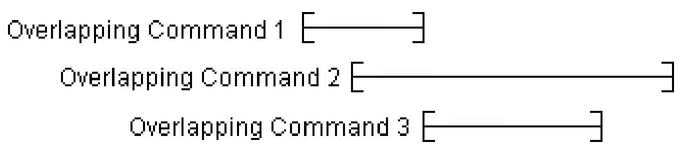
The AWG70000A Series instruments have several commands that are blocking. A blocking command does not allow any further commands to be executed until it is finished performing its task, such as a command that changes a hardware setting.

Blocking commands perform similar to sequential commands, but they tend to take a longer amount of time to complete. Because of the time for a blocking command to complete, if a number of blocking commands are run in a sequence followed by a query, the query could time out because the previous blocking commands have not finished.

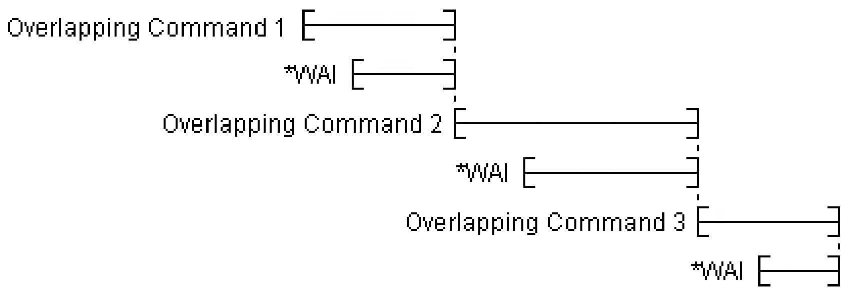
Blocking commands are noted in their command descriptions.

Overlapping commands

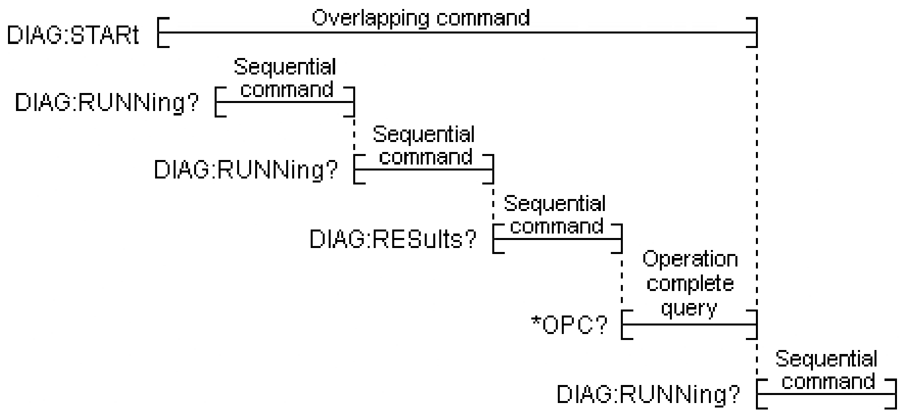
Overlapping commands run concurrently with other commands, allowing additional commands to start before the overlapping command has finished. The illustration below shows how a series of overlapping commands might start and end.



In some instances, you may want to make an overlapping command perform similarly to a sequential command. This is simply done by placing a *WAI command after the overlapping command as illustrated below.

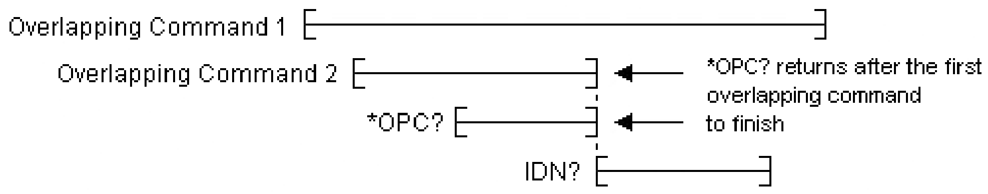


You always want to ensure the overlapping command has completed. This is done by using the *OPC? command. When an overlapping command starts, the operation complete status event is cleared. When the overlapping command completes, the operation complete status event is set. The *OPC? command requirement is to return a 1 when the operation complete status event is set. In the illustration below, the OPC? command blocks any further commands from being executed until the operation complete status event is set.



NOTE. *Always ensure overlapping commands have completed by placing an *OPC? command after the overlapping command.*

The AWG70000A Series instruments are limited to one outstanding overlapping command per *OPC?. If two or more overlapping commands are sent and followed by an *OPC?, then the first overlapped command to finish will set the operation complete status event and *OPC? will return 1. This early return may produce undesirable results. The following illustration shows this behavior.



NOTE. *For AWG70000A Series instruments, the *OPC? query only supports one overlapping command, not the two or more overlapping commands as defined in the IEEE Std 488.2 standard.*

Overlapping commands are noted in their command descriptions.

Command groups

Calibration group commands

Table 2-8: Calibration group commands and their descriptions

Command	Description
CALibration:ABORt	Stops the self calibration process and restores the previous calibration constants.
CALibration[:ALL]	Performs a full calibration of the AWG. The query form performs a full calibration and returns a status of the operation.
CALibration:CATalog?	Returns the list of subsystems, areas, or procedures.
CALibration:LOG?	Returns a string of continuous concatenated calibration results.
CALibration:LOG:CLEar	Clears the results log.
CALibration:LOG:DETails	Sets or returns the flag that controls the amount of result information saved into the log.
CALibration:LOG:FAILuresonly	Sets and returns the flag that controls the amount of result information saved into the log.
CALibration:RESTore	Restores the calibration constants from the factory non-volatile memory and copied to user storage.
CALibration:RESult?	Returns the status about the results of the last start of a set of selected calibration procedures.
CALibration:RESult:TEMPerature?	Returns the temperature from the results of the last start of a set of selected procedures.
CALibration:RESult:TIME?	Returns the time from the results of the last start of a set of selected procedures.
CALibration:RUNNing?	Returns the name of the subsystem, area, and procedure in progress.
CALibration:STARt	Starts the selected set of calibrations.
CALibration:STATe:FACTory?	Returns the current factory state of the calibration for the AWG.
CALibration:STATe:USER?	Returns the current factory state of the calibration for the AWG.
CALibration:STOP:STATe?	Returns the state of the calibration procedure.

Clock group commands

Table 2-9: Clock group commands and their descriptions

Command	Description
CLOCk:ECLock:DIVider	Sets or returns the divider rate of the external clock.
CLOCk:ECLock:FREQuency	Sets or returns the expected frequency being provided by the external clock.
CLOCk:ECLock:FREQuency:ADJust	Adjusts the external clock to the frequency specified by the user or set by the external clock frequency detect.
CLOCk:ECLock:FREQuency:DETect	Detects the frequency of the signal applied to the Clock In connector and adjusts the system to use the signal.
CLOCk:ECLock:MULTiplier	Sets or returns the multiplier rate of the external clock.
CLOCk:EREFerence:DIVider	Sets or returns the divider rate of the external reference oscillator.
CLOCk:EREFerence:FREQuency	Sets or returns the expected frequency of the signal applied to the EXT REF input connector.
CLOCk:EREFerence:FREQuency:DETect	Detects the frequency of the signal applied to the EXT REF input connector and adjusts the system to use the signal.
CLOCk:EREFerence:MULTiplier	Sets or returns the multiplier rate of the variable external reference signal.
CLOCk:JITTer	Sets or returns whether or not low jitter (Jitter Reduction) is enabled on the internal system clock or the clock signal applied to the Reference In connector.
CLOCk:OUTPut:FREQuency?	Returns the frequency of the output clock.
CLOCk:OUTPut[:STATe]	Sets or returns the output state of the clock output.
CLOCk:PHASe[:ADJust]	Sets or returns the internal clock phase adjustment of the AWG.
CLOCk:SOURce	Sets or returns the source of the clock.
CLOCk:SOUT[:STATe]	Sets or returns the state of the Sync Clock Out output.
CLOCk:SRATe	Sets or returns the sample rate for the clock.

Control group commands

Table 2-10: Control group commands and their descriptions

Command	Description
AWGControl[:CLOCk]:DRATe	***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:ECLock:DIVider.* Sets or returns the divider rate for the external clock.
AWGControl:CLOCk:PHASe[:ADJust]	***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:PHASe[:ADJust].* Sets or returns the phase of the internal clock.
AWGControl[:CLOCk]:SOURce	***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:SOURce.* Sets or returns the clock source.
AWGControl:CONFigure:CNUMber?	Returns the number of channels available on the AWG.
AWGControl:INTerleave:ADJustment:AMPLitude	Sets or returns the interleave adjustment amplitude percentage.
AWGControl:INTerleave:ADJustment:PHASe	Sets or returns the interleave adjustment phase.
AWGControl:RMODe	***NOTE.** This command exists for backwards compatibility. Use the command [SOURce[n]]:RMODe.* Sets or returns the run mode of the AWG.
AWGControl:RSTate?	Returns the state of the AWG.
AWGControl:RUN[:IMMediate]	Initiates the output of a waveform or sequence.
AWGControl:SNAMe?	Returns the most recently saved setup location.
AWGControl:SREStore	***NOTE.** This command exists for backwards compatibility. Use the command MMEMory:OPEN:SETup.* Opens a setup file into the AWG's setup memory.

Table 2-10: Control group commands and their descriptions (cont.)

Command	Description
AWGControl:SSAVe	***NOTE.*** *This command exists for backwards compatibility. Use the command MMEMory:SAVE:SETup* Saves the AWG's setup with waveforms.
AWGControl:STOP[:IMMediate]	Stops the output of a waveform or sequence.

Diagnostic group commands

Table 2-11: Diagnostic group commands and their descriptions

Command	Description
ACTive:MODE	Enables and disables access to diagnostics or calibration.
DIAGnostic:ABORt	Stops the current diagnostic test.
DIAGnostic:CATalog?	Returns the list of all diagnostic tests per selected type.
DIAGnostic:CONTrol:COUNt	Sets or returns the number of loop counts used when the selected loop mode is "COUNt".
DIAGnostic:CONTrol:HALT	Determines or returns whether the next execution of diagnostics looping stops on the first diagnostic failure that occurs or continues to loop on the selected set of diagnostic functions.
DIAGnostic:CONTrol:LOOP	Determines or queries whether the next start of diagnostics runs once, runs continuous loops, or loops for a number times for the selected set of tests.
DIAGnostic:DATA?	Returns the results of last executed tests for the NORMal diagnostic type.
DIAGnostic[:IMMediate]	Executes all of the NORMal diagnostic tests. The query form executes the selected tests and returns the results.
DIAGnostic:LOG?	Returns a string of continuous concatenated test results.
DIAGnostic:LOG:CLEar	Clears the diagnostics results log.
DIAGnostic:LOG:FAILuresonly	Sets or returns the flag that controls the amount of result information saved into the log.

Table 2-11: Diagnostic group commands and their descriptions (cont.)

Command	Description
DIAGnostic:LOOPs?	Returns the number of times that the selected diagnostics set was completed during the current running or the last diagnostic running of the set.
DIAGnostic:RESult?	Returns the status about the results of the last start of a set of selected tests.
DIAGnostic:RESult:TEMPerature?	Returns the temperature from the results of the last start of a set of selected tests.
DIAGnostic:RESult:TIME?	Returns the time from the results of the last start of a set of selected tests.
DIAGnostic:RUNNing?	Returns the name of the subsystem, area, and test of the current diagnostic test.
DIAGnostic:SELect	Selects one or more tests of the current test list.
DIAGnostic:SELect:VERify?	Returns selection status of one specific test.
DIAGnostic:STARt	This command starts the execution of the selected set of diagnostic tests.
DIAGnostic:STOP	Stops the diagnostic tests from running, after the diagnostic test currently in progress completes.
DIAGnostic:STOP:STATe?	Returns the current state of diagnostic testing.
DIAGnostic:TYPE	Sets or returns the diagnostic type.
DIAGnostic:TYPE:CATalog?	Returns a list of diagnostic types available depending on the end user.
DIAGnostic:UNSelect	Unselects one or more tests of the current test list.

Display group commands

Table 2-12: Display group commands and their descriptions

Command	Description
DISPlay[:PLOT][:STATe]	Minimizes or restores the plot’s display area on the Home screen's channel window of the AWG.

Function generator group commands

Table 2-13: Function generator group commands and their descriptions

Command	Description
FGEN[:CHANnel[n]]:AMPLitude	Sets or returns the amplitude of the generated waveform of the selected channel.
FGEN[:CHANnel[n]]:FREQuency	Sets or returns the frequency of the generated waveform.
FGEN[:CHANnel[n]]:DCLevel	Sets or returns the DC level of the generated waveform of the selected channel.
FGEN[:CHANnel[n]]:HIGH	Sets or returns the generated waveform's high voltage value of the selected channel.
FGEN[:CHANnel[n]]:LOW	Sets or returns the generated waveform's low voltage value of the selected channel.
FGEN[:CHANnel[n]]:OFFSet	Sets or returns the offset of the generated waveform of the selected channel.
FGEN:PERiod?	Returns the generated waveform's period.
FGEN[:CHANnel[n]]:PHASe	Sets or returns the generated waveform's phase of the selected channel.
FGEN[:CHANnel[n]]:SYMMetry	Sets or returns the generated waveform's symmetry value of the selected channel.
FGEN[:CHANnel[n]]:TYPE	Sets or returns the waveform type (shape) of the selected channel.
FGEN:COUPle:AMPLitude	Sets or returns the coupling of amplitude controls between channel 1 and channel 2 of a two channel instrument.

IEEE mandated and optional group commands

All AWG IEEE mandated and optional command implementations are based on the SCPI standard and the specifications for devices in IEEE 488.2.

Table 2-14: IEEE mandated and optional group commands and their descriptions

Command	Description
*CAL?	Runs all self calibrations. Same as CALibration[:ALL].
*CLS	Clears all event registers and queues.
*ESE	Sets the Event Status Enable Register (ESER).
*ESE?	Returns the contents of the Event Status Enable Register (ESER).
*ESR?	Returns the current contents of the Event Status Register (ESR).

Table 2-14: IEEE mandated and optional group commands and their descriptions (cont.)

Command	Description
*IDN?	Returns identification information for the AWG.
*OPC	Causes the AWG to sense the internal flag referred to as the “No-Operation-Pending” flag. When the pending operation has completed, the Operation Complete (OPC) bit in the Event Status Register (ESR) is set.
*OPC?	Causes the AWG to sense the internal flag referred to as the “No-Operation-Pending flag. When the pending operation has completed, a “1” will be returned to the client.
*OPT?	Returns the implemented options for the AWG.
*RST	Resets the AWG to its default state.
*SRE	Sets the bits in the Service Request Enable Register (SRER).
*SRE?	Returns the contents of the Service Request Enable Register (SRER).
*STB?	Returns the contents of Status Byte Register (SBR).
*TRG	Generates a trigger event for Trigger A only.
*TST?	Executes a power-on self test and returns the results.
*WAI	Ensures the completion of the previous command before the next command is issued.

Instrument group commands

Table 2-15: Instrument group commands and their descriptions

Command	Description
INSTrument:COUPle:SOURce	Sets or returns the coupled state of the channels (two channel AWGs).
INSTrument:MODE	Sets or returns the AWG mode.

Mass memory group commands

Table 2-16: Mass Memory group commands and their descriptions

Command	Description
MMEMory:CATalog?	Returns the current contents and state of the mass storage media.
MMEMory:CDIRectory	Sets or returns the current directory of the file system on the AWG.
MMEMory:DATA	Sets or returns block data to/from file in the current mass storage device.
MMEMory:DATA:SIZE?	Returns the size in bytes of a selected file.
MMEMory:DELete	Deletes a file or directory from the AWG's hard disk.
MMEMory:IMPort	***NOTE.*** *This command exists for backwards compatibility. Use the command MMEMory:OPEN.* Loads a file into the AWG waveform list.
MMEMory:IMPort:PARameter:NORMalize	***NOTE.*** *This command exists for backwards compatibility. Use the command MMEMory:OPEN:PARameter:NORMalize.* Sets or returns if the imported data is normalized during select file format import operations. The imported waveform data (for select file formats) is normalized based on the option set in this command.
MMEMory:MDIRectory	Creates a new directory in the current path on the mass storage system.
MMEMory:MSIS	Sets or returns a mass storage device used by all MMEMory commands.
MMEMory:OPEN	Loads a file into the AWG waveform list.
MMEMory:OPEN:PARameter:NORMalize	Sets or returns if the imported data is normalized during select file format import operations.
MMEMory:OPEN:SASSet[:WAVeform]	Loads all waveforms or a single desired waveform from a file into the AWG's waveforms list.
MMEMory:OPEN:SASSet:SEQuence	Loads all sequences or a single desired sequence from a file into the AWG's sequences list.
MMEMory:OPEN:SETup	Opens a setup file into the AWG's setup memory.
MMEMory:OPEN:TXT	Loads a file into the AWG's waveform list.

Table 2-16: Mass Memory group commands and their descriptions (cont.)

Command	Description
MMEMory:OPEN:PARameter:NORMalize	Sets or returns if the imported data is normalized during select file format import operations.
MMEMory:SAVE:SETup	Saves the AWG's setup and optionally includes the waveforms.
MMEMory:SAVE:SEQuence	Exports a sequence given a unique name to an eligible storage location as a .SEQX file type.
MMEMory:SAVE[:WAVeform]:TXT	Exports a waveform given a unique waveform name to an eligible storage location from the AWG's waveforms as a .TXT file type.
MMEMory:SAVE[:WAVeform][:WFMX]	Exports a waveform given a unique waveform name to an eligible storage location from the AWG's waveforms as a .WFMX file type.

Output group commands

Table 2-17: Output group commands and their descriptions

Command	Description
OUTPut:OFF	Sets or returns whether or not 'All Outputs Off' has been enabled.
OUTPut[n][:STATe]	Sets or returns the output state of the channel.
OUTPut[n]:SVALue[:ANALog][:STATe]	Sets or returns the output condition of a waveform of the specified channel while the instrument is in the stopped state.
OUTPut[n]:SVALue:MARKer[1|2]	Sets or returns the output condition of the specified marker of the specified channel while the instrument is in the stopped state.
OUTPut[n]:WVALue[:ANALog][:STATe]	Sets or returns the output condition of a waveform of the specified channel while the instrument is in the waiting-for-trigger state.
OUTPut[n]:WVALue:MARKer[1|2]	Sets or returns the output condition of the specified marker of the specified channel while the instrument is in the waiting-for-trigger state.

Sequence group commands

Table 2-18: Sequence group commands and their descriptions

Command	Description
SLISt:NAME?	Returns the name of the sequence corresponding to the specified index in the sequence list.
SLISt:SEQuence:DELete	Deletes a specific sequence or all sequences from the sequence list.
SLISt:SEQuence:EVENt:JTIMing	Sets or returns the jump timing of a sequence.
SLISt:SEQuence:EVENt:PJUMp:ENABle	Sets or returns the pattern jump for a sequence.
SLISt:SEQuence:EVENt:PJUMp:DEFine	Sets or returns the pattern jump targets in the pattern jump table.
SLISt:SEQuence:EVENt:PJUMp:SIZE?	Returns number of patterns in the pattern table.
SLISt:SEQuence:LENGth?	Returns the total number of steps in the named sequence.
SLISt:SEQuence:NEW	Creates a new sequence.
SLISt:SEQuence:RFLag	Sets or returns the Enable Flag Repeat value for the sequence.
SLISt:SEQuence:STEP[n]:EJINput	Sets or returns weather the sequence of play will jump when it receives Trigger A, Trigger B, or not jump at all.
SLISt:SEQuence:STEP[n]:EJUMp	Sets or returns the step that the sequence of play will jump to on a trigger event.
SLISt:SEQuence:STEP[n]:GOTO	Sets or returns the Goto target for a step.
SLISt:SEQuence:STEP[n]:RCOunt	Sets or returns the repeat count.
SLISt:SEQuence:STEP:RCOunt:MAX?	Returns the maximum number of repeats allowed for a step in a sequence.
SLISt:SEQuence:STEP[n]:TASSet[m]?	Returns the name of the waveform assigned to a step.
SLISt:SEQuence:STEP[n]:TASSet:SEQuence	Assigns a subsequence for a specific sequence's step and track.
SLISt:SEQuence:STEP[n]:TASSet[m]:TYPE?	Returns the type of asset assigned at the step and track for a specified sequence.
SLISt:SEQuence:STEP[n]:TASSet[m]:WAVeform	Assigns a waveform to the specified track of a step.
SLISt:SEQuence:STEP[n]:TFLag[m]:AFLag	Sets or returns the Flag A value of the track in a sequence step.
SLISt:SEQuence:STEP[n]:TFLag[m]:BFLag	Sets or returns the Flag B value of the track in a sequence step.
SLISt:SEQuence:STEP[n]:TFLag[m]:CFLag	Sets or returns the Flag C value of the track in a sequence step.

Table 2-18: Sequence group commands and their descriptions (cont.)

Command	Description
SLISt:SEQuence:STEP[n]:TFLag[m]:DFLag	Sets or returns the Flag D value of the track in a sequence step.
SLISt:SEQuence:STEP[n]:WINPut	Sets or returns the wait input for a step.
SLISt:SEQuence:TSTamp?	Returns the timestamp of the named sequence.
SLISt:SEQuence:STEP:MAX?	Returns the maximum number of steps allowed in a sequence.
SLISt:SEQuence:TRACk?	Returns the total number of tracks in the named sequence.
SLISt:SEQuence:TRACk:MAX?	Returns the maximum number of tracks allowed in a sequence.
SLISt:SIZE?	Returns the number of sequences in the sequence list.

Source group commands

Table 2-19: Source group commands and their descriptions

Command	Description
[SOURce[n]]:CASSet?	Returns the waveform or sequence assigned to a channel.
[SOURce[n]]:CASSet:SEQuence	Assigns a sequence to a channel.
[SOURce[n]]:CASSet:TYPE?	Returns the type of the asset (waveform or sequence) assigned to a channel.
[SOURce[n]]:CASSet:WAVeform	Assigns a waveform to a channel.
[SOURce[n]]:DAC:RESolution	Sets or returns the DAC resolution.
[SOURce]:FREQuency[:CW]][:FIXed]	Sets or returns the clock sample rate of the AWG. ***NOTE.*** *This command exists for backwards compatibility. Use the command CLOCk:SRATe.*
[SOURce[n]]:JUMP:FORCe	Forces the sequencer to jump to the specified step for the specified channel.
[SOURce[n]]:JUMP:PATTern:FORCe	Generates an event forcing the sequencer to the step specified by pattern in the pattern jump table.
[SOURce[n]]:MARKer[1|2]:DELay	Sets or returns the marker delay.
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]	Sets or returns the marker amplitude.
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH	Sets or returns the marker high level.

Table 2-19: Source group commands and their descriptions (cont.)

Command	Description
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW	Sets or returns the marker low level.
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet	Sets or returns the marker offset.
[SOURce]:RCCouple	Sets or returns the coupled state of the channel's run controls of a two channel instrument.
[SOURce[n]]:RMODe	Sets or returns the run mode of the AWG.
[SOURce]:ROSCillator:MULTiplier	Sets or returns the multiplier of the external reference oscillator. **NOTE.** *This command exists for backwards compatibility. Use the command CLOCk:EREFerence:MULTiplier.*
[SOURce[n]]:SCSTep?	Returns the current step of the sequence while system is running.
[SOURce[n]]:SKEW	Sets or returns the skew for the waveform associated with a channel in a two channel configuration.
[SOURce[n]]:TINPut	Sets or returns the trigger input source.
[SOURce[n]]:VOLTage[:LEVel][:IMMediate][:AMPLitude]	Sets or returns the amplitude for the waveform associated with a channel.
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH	Sets or returns the high voltage level for the waveform associated with a channel.
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW	Sets or returns the low voltage level for the waveform associated with a channel.
[SOURce[n]]:WAVeform	Sets or returns the name of the waveform assigned to a channel.

Status group command

The external controller uses the status commands to coordinate operation between the AWG and other devices on the bus. The status commands set and query the registers/queues of the AWG event/status reporting system. For more information about registers and queues, see Status and Event reporting section.

Table 2-20: Status group commands and their descriptions

Command	Description
STATus:OPERation:CONDition?	Returns the contents of the Operation Condition Register (OCR).
STATus:OPERation:ENABle	Sets or returns the mask for the Operation Enable Register (OENR).
STATus:OPERation[:EVENt]?	Returns the contents of Operation Event Register (OEVR).
STATus:OPERation:NTRansition	Sets or returns the negative transition filter value of the Operation Transition Register (OTR).
STATus:OPERation:PTRansition	Sets or returns the positive transition filter value of the Operation Transition Register (OTR).
STATus:PRESet	Sets the OENR and QENR registers.
STATus:QUEStionable:CONDition?	Returns the status of the Questionable Condition Register (QCR).
STATus:QUEStionable:ENABle	Sets or returns the mask for Questionable Enable Register (QENR).
STATus:QUEStionable[:EVENt]?	Returns the contents of the Questionable Event (QEVR) Register and clears it.
STATus:QUEStionable:NTRansition	Sets or returns the negative transition filter value of the Questionable Transition Register (QTR).
STATus:QUEStionable:PTRansition	Sets or returns the positive transition filter value of the Questionable Transition Register (QTR).

Synchronization group commands

The external controller uses these commands and an AWGSYNC01 to synchronize two to four AWG70000 instruments.

Table 2-21: Synchronization group commands and their descriptions

Command	Description
SYNChronize:ADJust:[STARt]	Performs a system sample rate calibration.
SYNChronize:CONFigure	Configures the ports in a synchronized system.
SYNChronize:DESKew:ABORt	Cancels a system deskew calibration.
SYNChronize:DESKew:[STARt]	Performs a system deskew calibration.
SYNChronize:DESKew:STATe?	Returns the state of the system deskew condition.
SYNChronize:ENABle	Enables or disables synchronization in the instrument.
SYNChronize:TYPE?	Returns the instrument type (master or slave) in the synchronized system.

System group commands

Table 2-22: System group commands and their descriptions

Command	Description
SYSTem:DATE	Sets or returns the system date.
SYSTem:ERRor:ALL?	Returns the error and event queue for all the unread items and removes them from the queue.
SYSTem:ERRor:CODE:ALL?	Returns the error and event queue for the codes of all the unread items and removes them from the queue.
SYSTem:ERRor:CODE[:NEXT]?	Returns the error and event queue for the next item and removes it from the queue.
SYSTem:ERRor:COUNt?	Returns the error and event queue for the number of unread items.
SYSTem:ERRor:DIALog	Sets or returns the error dialog display status.
SYSTem:ERRor[:NEXT]?	Returns data from the error and event queue.
SYSTem:TIME	Sets or returns the system time.
SYSTem:VERSion?	Returns the SCPI version number to which the command conforms.

Trigger group commands

Table 2-23: Trigger group commands and their descriptions

Command	Description
TRIGger[:SEQuence][:IMMediate]	Generates a trigger event.
TRIGger[:SEQuence]:IMPedance	Sets or returns the impedance of the external triggers.
TRIGger[:SEQuence]:INTerval	Sets or returns the internal trigger interval.
TRIGger[:SEQuence]:LEVel	Sets or returns the external trigger input levels (threshold).
TRIGger[:SEQuence]:MODE	Sets or returns the trigger timing used when an external trigger source is being used.
TRIGger[:SEQuence]:SLOPe	Sets or returns the external trigger slopes.
TRIGger[:SEQuence]:SOURce	Sets or returns the trigger source. ***NOTE.*** *This command exists for backwards compatibility. Use the command [SOURce[n]]:TINPut.*
TRIGger[:SEQuence]:WVALue	Sets or returns the output data position of a waveform while the instrument is in the waiting-for-trigger state ***NOTE.*** *This command exists for backwards compatibility. Use the commands OUTPut[n]:WVALue[:ANALog][:STATe] and OUTPut[n]:WVALue:MARKer[1|2].*

Waveform group commands

Table 2-24: Waveform group commands and their descriptions

Command	Description
WLISt:LAST?	Returns the name of the most recently added waveform in the waveform list.
WLISt:NAME?	Returns the waveform name of an element in the waveform list.
WLISt:SIZE?	Returns the size of the waveform list.
WLISt:WAVeform:DATA	Transfers waveform data from the external controller into the specified waveform or from a waveform to the external control program.
WLISt:WAVeform:DELete	Deletes the waveform from the currently loaded setup.
WLISt:WAVeform:GRANularity?	Returns the granularity of sample points required for the waveform to be valid.

Table 2-24: Waveform group commands and their descriptions (cont.)

Command	Description
WLISt:WAVeform:LMAXimum?	Returns the maximum number of waveform sample points allowed.
WLISt:WAVeform:LMINimum?	Returns the minimum number of waveform sample points required for a valid waveform.
WLISt:WAVeform:LENGth?	Returns the size of the waveform.
WLISt:WAVeform:MARKer:DATA	Sets or returns the waveform marker data.
WLISt:WAVeform:NEW	Creates a new empty waveform in the waveform list of current setup.
WLISt:WAVeform:NORMalize	Normalizes a waveform that exists in the waveform list.
WLISt:WAVeform:RESample	Resamples a waveform that exists in the waveform list.
WLISt:WAVeform:SHIFt	Shifts the phase of a waveform that exists in the waveform list.
WLISt:WAVeform:TSTamp?	Returns the timestamp of the waveform.
WLISt:WAVeform:TYPE?	This command returns the type of the waveform. ***NOTE.*** *This command exists for backwards compatibility.*

Waveform data format

The AWG supports the Floating Point format of waveform data.

Floating data format is the same as the IEEE 754 single precision format. It occupies 4 bytes per waveform data point. It stores normalized data without any scaling. When the waveform in real data format is output, the data is rounded off to the nearest integer value and clipped to fit the DAC range.

The waveforms in the real format retains normalized values. The format for the waveform analog data in the real format is IEEE754 single precision.

The real data format is shown in the following table.

Table 2-25: Real data format

Byte offset 3	Byte offset 2	Byte offset 1	Byte offset 0
IEEE754 single precision format (32 bits)			

DAC resolution affects the way hardware interprets the bits in the waveform. Therefore it is necessary to reload waveforms once the DAC resolution is modified. To understand how to change the DAC resolution, see the [SOURce[n]]:DAC:RESolution command. To understand how to load a waveform into hardware memory see the [SOURce[n]]:WAVeform command.

Byte order during transfer

Waveform data is always transferred in LSB first format.

Command descriptions

ACTive:MODE

This command enables and disables access to diagnostics or calibration. When the active mode is DIAGnostic or CALibration, all other non-diagnostic and non-calibration commands are ignored and no action occurs.

If a test or procedure is in progress, errors are not returned; they are added to the system error queue, which can be accessed with SYSTem:ERRor[:NEXT]?. For example:

- -200, "[D|C] are still running;"
- -300,"Device-specific error; Diagnostics tests still in progress - act:mode diag"
- -300,"Device-specific error; Calibration procedures still in progress - act:mode cal"

To avoid this error, use the command DIAGnostic:STOP:STATe? or CALibration:STOP:STATe? to test for this condition.

This command blocks when changing any state. Changing the state to NORMal causes a hardware initialization process and any related system settings are restored.

If any diagnostic tests are in progress, then the request to change the active mode fails and the mode will not change.

When changing the active mode, it's recommended to follow the action with an operation complete command (*OPC) to ensure the command has finished before other commands are processed.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Diagnostic

Syntax

```
ACTive:MODE {NORMal|CALibration|DIAGnostic}
ACTive:MODE?
```

Related Commands

DIAGnostic:ABORt, DIAGnostic:STOP, CALibration:ABORt

Arguments

`NORMal` disables any active state for either calibration or diagnostics. When entering the active state of normal, the hardware is set to a default state and the previous system state is restored and waveform playout is set to off.
`CALibration` enables the active state for the calibration. Entering the active state of calibration turns waveform playout off.
`DIAGnostic` enables the active state for the diagnostics. Entering the active state of diagnostics turns waveform playout off.

`*RST` sets this to NORM.

Returns

NORM
CAL
DIAG

Examples

`ACTIVE:MODE DIAGNOSTIC` enables the diagnostics mode.

`ACTIVE:MODE?` might return DIAG if in the diagnostics mode.

AWGControl[:CLOCk]:DRATe

***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:ECLock:DIVider.*

This command sets or returns the divider rate for the external clock.

Conditions

Setting the clock divider rate forces the clock multiplier rate to a value of 1.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Control

Syntax

```
AWGControl[:CLOCk]:DRATe <NR1>
AWGControl[:CLOCk]:DRATe?
```

Related Commands

CLOCk:ECLock:MULTiplier, CLOCk:SRATe, AWGControl[:CLOCk]:SOURce

Arguments

A single <NR1> value that is a power of 2.

Range: 1 to 16777216

`*RST` sets this to 1.

Returns

A single <NR1> value.

Examples

`AWGCONTROL:CLOCK:DRATE 4`
`*OPC?`
sets the external clock divider rate to 4. The overlapping command is followed with an Operation Complete query.

`AWGCONTROL:CLOCK:DRATE?` might return 4.

AWGControl:CLOCk:PHASe[:ADJust]

NOTE. *This command exists for backwards compatibility. Use the command CLOCk:PHASe[:ADJust].*

This command sets or returns the internal clock phase adjustment of the AWG.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Control

Syntax

```
AWGControl:CLOCk:PHASe[:ADJust] <NR1>
AWGControl:CLOCk:PHASe[:ADJust]?
```

Arguments

A single <NR1> value.

Range: –10800 degrees to 10800 degrees.

Returns

A single <NR1> value.

Examples

`AWGCONTROL:CLOCK:PHASE:ADJUST 100`
sets the clock phase to 100 degrees.

`AWGCONTROL:CLOCK:PHASE:ADJUST?` might return 100, indicating the clock phase is set to 100 degrees.

AWGControl[:CLOCk]:SOURce

***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:SOURce.*

This command sets or returns the source of the clock.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Control

Syntax

```
AWGControl[:CLOCk]:SOURce
{INTernal|EXTernal|EFIXed|EVARiable}
AWGControl[:CLOCk]:SOURce?
```

Related Commands

CLOCk:SOURce

Arguments

`INTernal` – clock signal is generated internally and the reference frequency is derived by the internal oscillator.
`EFIXed` – clock is generated internally and the reference frequency is derived from a fixed 10 MHz reference supplied at the Reference In connector.
`EVARiable` – clock is generated internally and the reference frequency is derived from a variable reference supplied at the Reference In connector.
`EXTernal` – clock signal supplied by the Clock In connector and the reference frequency is derived from the internal precision oscillator.

`*RST` sets this to INT.

Returns

```
INT
EXT
EFIXed
EVAR
```

Examples

`AWGCONTROL:CLOCK:SOURCE INTERNAL`
`*OPC?` sets the clock source to internal. The overlapping command is followed with an Operation Complete query.

`AWGCONTROL:CLOCK:SOURCE?` might return EXT, indicating that the clock source is set to use the Clock In connector.

AWGControl:CONFigure:CNUMber? (Query Only)

This command returns the number of channels available on the AWG.

Group Control

Syntax `AWGControl:CONFigure:CNUMber?`

Returns A single <NR1> value.

Examples `AWGCONTROL:CONFIGURE:CNUMBER?` might return 2.

AWGControl:INTerleave:ADJustment:AMPLitude

This command sets or returns the interleave amplitude adjustment as a percentage of the analog output voltage. The percentage is applied to both of the channel's interleave DACs such that the analog output voltage is minimally affected. When the analog output is changed, this amplitude percentage is applied at the same time.

Conditions This command is only valid on a single channel model.

Group Control

Syntax

```
AWGControl:INTerleave:ADJustment:AMPLitude <NRf>
AWGControl:INTerleave:ADJustment:AMPLitude?
```

Arguments A single <NRf> value.

Maximum percentage changed is ± 10% up to and including Min and Max of the analog output amplitude.
Minimum percentage that can change is 1%.

`*RST` sets this to 0%.

Returns <NRf>

Examples

`AWGCONTROL:INTERLEAVE:ADJUSTMENT:AMPLITUDE 10` adjusts the relationship between the two DACs by 10% of the analog output. The actual analog output is minimally affected.

`AWGCONTROL:INTERLEAVE:ADJUSTMENT:AMPLITUDE?` might return 10.0000000000, indicating that the interleave adjustment amplitude percentage is 10% of the analog output.

AWGControl:INTerleave:ADJustment:PHASe

This command sets or returns the interleave adjustment phase. The phase adjustment is applied to both of the channel's interleave DACs.

Conditions

This command is valid only for single channel models.

Group

Control

Syntax

```
AWGControl:INTerleave:ADJustment:PHASe <NRf>
AWGControl:INTerleave:ADJustment:PHASe?
```

Arguments

A single <NRf> value.

Range: –180 to 180 degrees, Resolution: 0.1 degrees.

`*RST` sets this to 0 degrees.

Returns

A single <NRf> value

Examples

`AWGCONTROL:INTERLEAVE:ADJUSTMENT:PHASE 120` sets the interleave adjustment phase to 120 degrees.

`AWGCONTROL:INTERLEAVE:ADJUSTMENT:PHASE?` might return 120.0000000000, indicating that the interleave adjustment phase is 120 degrees.

AWGControl:RMODe

NOTE. *This command exists for backwards compatibility. Use the command [SOURce[n]]:RMODe.*

This command sets or returns the run mode of the AWG.

Group

Control

Syntax

```
AWGControl:RMODe {CONTinuous|TRIGgered}
AWGControl:RMODe?
```

Related Commands

[SOURce[n]]:RMODe

Arguments

`CONTinuous` sets the Run Mode to Continuous (not waiting for a trigger event).

`TRIGgered` sets the Run Mode to Triggered, waiting for a trigger event. One waveform play out cycle completes, then play out stops, waiting for the next trigger event.

`*RST` sets this to CONT.

Returns

CONT
TRIG

Examples

`AWGCONTROL:RMODE TRIGGERED` sets the AWG Run mode to Triggered.

`AWGCONTROL:RMODE?` might return CONT if the AWG is in continuous mode.

AWGControl:RSTate? (Query Only)

This command returns the run state of the AWG.

Group Control

Syntax `AWGControl:RSTate?`

Related Commands [SOURce[n]]:RMODe

Returns A single <NR1> value.

0 indicates that the AWG has stopped.
1 indicates that the AWG is waiting for trigger.
2 indicates that the AWG is running.

Examples `AWGCONTROL:RSTATE?` returns 0 if waveform generation is stopped.

AWGControl:RUN[:IMMediate] (No Query Form)

This command initiates the output of a waveform or sequence. This is equivalent to pushing the play button ▶ on the front-panel or display. The AWG can be put in the run state only when waveforms or sequences are assigned to channels.

Conditions

This is a blockingcommand. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Control

Syntax

```
AWGControl:RUN[:IMMediate]
```

Related Commands

AWGControl:STOP[:IMMediate]

Examples

`AWGCONTROL:RUN:IMMEDIATE`
puts the AWG in the run state.

AWGControl:SNAMe? (Query Only)

This command returns the AWG's most recently saved setup location.

The response contains the full path for the file, including the disk drive letter (msus or, mass storage unit specifier).

Group

Control

Syntax

```
AWGControl:SNAMe?
```

Returns

Returns <file_name>,<msus>

<file_name> ::= <string>
a<msus> (mass storage unit specifier) ::= <string>
By default (when there has been no save setup command), this value is "","C:"

Examples

`AWGCONTROL:SNAME?` might return the following response: "\my\project\setups\mySetup.awgx","D:"

AWGControl:SREStore (No Query Form)

***NOTE.** This command exists for backwards compatibility. Use the command MMEMory:OPEN:SETup.*

This command opens a setup file into the AWG's setup memory.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Control

Syntax
```
AWGControl:SREStore <filepath>[,<msus>]
```

Related Commands MMEMory:OPEN:SETup

Arguments <filepath>::=<string>
<msus> (mass storage unit specifier) ::=<string>

Examples With mass storage unit specifier specified as a parameter:
```
AWGCONTROL:SRESTORE "\TestFiles\mySetup.awgx","C:"
```
With mass storage unit specifier specified within the file path:
```
AWGCONTROL:SRESTORE "C:\TestFiles\mySetup.awgx"
```

AWGControl:SSAVe (No Query Form)

***NOTE.** This command exists for backwards compatibility. Use the command MMEMory:SAVE:SETup.*

This command saves the AWG's setup with waveforms.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Control

Syntax
```
AWGControl:SSAVe <filepath>[,<msus>]
```

Related Commands MMEMory:SAVE:SETup

Arguments <filepath>::=<string>
<msus> (mass storage unit specifier)::=<string>

Examples
```
AWGCONTROL:SSAVE "C:\TestFiles\mySetup.awgx"

AWGCONTROL:SSAVE "\TestFiles\mySetup.awgx","C:"
```

AWGControl:STOP[:IMMediate] (No Query Form)

This command stops the output of a waveform or a sequence.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Control

Syntax
```
AWGControl:STOP[:IMMediate]
```

Related Commands AWGControl:RUN[:IMMediate]

Examples
```
AWGCONTROL:STOP:IMMEDIATE
*OPC?
```
stops the output of a waveform or sequence.

*CAL? (Query Only)

This command runs all selected calibrations. The command returns a status code indicating the success or failure of all of the calibrations. Any single calibration failure returns a failure code. *CAL? is equivalent to the CALibration[:ALL] command.

Use CALibration:RESult? to retrieve more detailed error information.

Conditions All calibrations are selected by default and cannot be modified by the user.

Group IEEE mandated and optional

Syntax `*CAL?`

Related Commands CALibration[:ALL], CALibration:RESult?

Returns A single <NR1> value, {0|-340}

Examples `*CAL?` might return -340 on any failure, 0 on all pass.

CALibration:ABORt (No Query Form)

This command stops the self calibration process and restores the previous calibration constants.

Conditions Setting only works in the active mode for calibration. See the ACTive:MODE command.

This command does not abort the CALibration[:ALL] command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Calibration

Syntax `CALibration:ABORt`

Related Commands ACTive:MODE, CALibration:STARt

Examples

```
CALIBRATION:ABORT
*OPC?
```

stops the calibration process. The overlapping command is followed with an Operation Complete query.

CALibration[:ALL]

This command does a full calibration of the AWG. In its query form, the command does a full calibration and returns a status indicating the success or failure of the operation. This command is equivalent to the *CAL? command.

Conditions

This command cannot be aborted.

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Calibration

Syntax

```
CALibration[:ALL]
CALibration[:ALL]?
```

Related Commands

*CAL?

Returns

<calibration error code> ::= <NR1>

0 indicates no error
–340 indicates an error

Examples

`CALIBRATION:ALL` performs a calibration.

`CALIBRATION:ALL?` performs a calibration and returns results. For example, it might return 0, indicating that the calibration completed without any errors.

CALibration:CATalog? (Query Only)

This command returns the list of calibration procedures.

All tests are grouped by areas. All areas are grouped by subsystems. The available subsystems, areas, and tests depend on the type of testing (such as POST or ALL).

Group

Calibration

Syntax `CALibration:CATalog?[{ALL|<subsystem>}[,{ALL|<area>}]]`

Arguments ALL – Keyword or as a string.
<subsystem> – a subsystem as a string.
<area> – an area as a string.

If there are no parameters, then the list of subsystems is returned.
If there is a valid subsystem parameter, then the list of areas for that subsystem is returned.
If the subsystem parameter is "ALL", then all the procedures of all the areas of all the subsystems is returned. Each procedure is prefixed with "<subsystem>:<area>:" and separated by a comma. Lists are always in priority of desired execution.
If the area parameter is "ALL", then all the procedures of all the areas for a specified subsystem is returned. Each procedure is prefixed with "<area>:" and separated by a comma. Lists are always in priority of desired execution.
If the subsystem and area parameters are valid, then the list of procedures for that subsystem and area is returned.

Returns String of all calibration "subsystems", "areas" and/or "procedures" separated by commas.

Examples `CALIBRATION:CATALOG?` might return "Initialization,Channel1,Channel2,System".

`CALIBRATION:CATALOG? "Channel1"` might return "Dc,Adc,Clock,Align,Dac,Marker1,Marker2"

`CALIBRATION:CATALOG? "ALL"` might return "Initialization:Init:Calibration Initialization,Channel1:Dc:Differential Offset,Channel1:Dc:Common Mode,Channel1:Dc:Amplitude,Channel1:Adc:Adc Internal,Channel1:Clock:Clock Amplitude,Channel1:Clock:Clock Offset, Channel1:Align:Sample Point, Channel1:Dac:Speed"

CALibration:LOG? (Query Only)

This command returns a string of continuous concatenated calibration results. The start time is recorded plus one or more <cal path>:<cal name> <result>. This command can be issued while calibration is still in progress. Use the CALibration:LOG:CLEar command to start a fresh log and provide additional information.

Log results are still valid if the calibration is aborted and the calibration constants are restored.

***NOTE.** The returned string is limited, which can cause lost results. Only the first 64K of text is recorded.*

Group Calibration

Syntax `CALibration:LOG?`

Related Commands CALibration:LOG:CLEar

Returns <string>::="<Started timestamp><LF delimiter><calibration name and result>[<LF delimiter><calibration name and result>]"

Examples `CALIBRATION:LOG?` might return "Channel1:Dc:Amplitude Started 6/14/2011 10:19 AM<LFCR>Channel1:Dc:Amplitude FAIL<LFCR>Channel1:Dc:Common Mode Offset Started 6/14/2011 10:23 AM<LFCR>Channel1:Dc:Common Mode Offset PASS<LFCR>"

CALibration:LOG:CLEar (No Query Form)

This command clears the results log.

The command works when in the active mode for calibration. See the ACTive:MODE command.

Group Calibration

Syntax `CALibration:LOG:CLEar`

Related Commands ACTive:MODE

Examples `CALIBRATION:LOG:CLEAR` clears the results log.

CALibration:LOG:DETails

This command sets or returns the flag that controls the amount of result information saved into the log.

Typically, the additional information is related to errors. It is important to note, that details are generated during the calibration, and need to be saved during execution. Enabling details potentially reduces the number of pass/fail results in the log due to log size limitations.

0 and 1 are the same as OFF and ON respectively.

Conditions

The set form of this command only works in the active mode for calibration. See the ACTive:MODE command.

Group

Calibration

Syntax

```
CALibration:LOG:DETails {OFF|ON|0|1}
CALibration:LOG:DETails?
```

Related Commands

ACTive:MODE

Arguments

OFF disables the detail mode.
ON enables the detail mode.
<Boolean> of 0 or 1 only. 0 and 1 are equivalent to OFF and ON respectively.

`*RST` sets this to 0.

Returns

A single <Boolean> value representing current cal log details setting {0|1}

Examples

`CALIBRATION:LOG:DETAILS`

`CALIBRATION:LOG:DETAILS OFF` disables the detail mode.

`CALIBRATION:LOG:DETAILS?` might return 0, showing the detail mode is disabled.

`CALIBRATION:LOG:DETAILS?` might return "Channel1:Dc:Amplitude Started 6/14/2011 10:19 AM<LFCR>Channel1:Dc:Amplitude FAIL<LFCR>"

`CALIBRATION:LOG:DETAILS 1` enables the detail mode.

`CALIBRATION:LOG:DETAILS?` might return 1, showing enabled detailed message in the log.

`CALIBRATION:LOG:DETAILS?` might return "Channel1:Dc:Amplitude, Started 6/14/2011 10:19 AM Temperature 40C<LFCR>Error 0x01dc2345 Negative Low Value Out of Range Expected 0.08V, Actual 0.78V<LFCR>Channel1:Dc:Amplitude FAIL <LFCR>"

CALibration:LOG:FAILuresonly

This command sets or returns the flag that controls the amount of result information saved into the log. This controls all tests that pass or fail or only tests that fail. It is important to note, that details are generated during the test, and need to be saved during the test execution.

Conditions

The set form of this command only works in the active mode for calibration. See the ACTive:MODE command.

Group

Calibration

Syntax

```
CALibration:LOG:FAILuresonly {OFF|ON|0|1}
CALibration:LOG:FAILuresonly?
```

Related Commands

ACTive:MODE

Arguments

OFF disables the failures only mode.
ON enables the failures only mode.
<Boolean> {0|1}. 0 and 1 are the equivalent of OFF and ON respectively.

`*RST` sets this to 0.

Returns

A single <Boolean> value representing current calibration log failures only state {0|1}.

Examples

`CALIBRATION:LOG:FAILURESONLY OFF` disables the failure only log mode.

`CALIBRATION:LOG:FAILURESONLY 1` enables the failure only log mode.

`CALIBRATION:LOG:FAILURESONLY?` might return 1, indicating the failure only log mode is enabled.

CALibration:RESTore (No Query Form)

This command restores the calibration constants from the factory non-volatile memory and copied to user storage.

Conditions

Setting only works in the active mode for calibration. See the ACTive:MODE command.

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Calibration

Syntax

`CALibration:RESTore`

Related Commands

ACTive:MODE

Examples

`CALIBRATION:RESTORE` sets all calibration constants to their factory settings.

CALibration:RESult? (Query Only)

This command returns the status of the last calibration procedure. This query-only command can be issued while calibration is in progress.

Group

Calibration

Syntax

`CALibration:RESult?`

Returns

"<result record>"
<result record>::= <subsystem>:[<area>:[<procedure>:]]<details>

<details>::= <Status>,<Loop Count>,<Pass>,<Fail>
<Status>::= S(C|R|U) Reflexs the "current" or "last" state. Currently by request, when the status reflects only the subsystem or area, then a U for Unknown/Uncalibrated will be set for any of the procedures that are unknown even if it is only 1 out of 10 selected procedures.
<Loop Count> ::= LC(#)
<Pass> ::= P(#)
<Fail> ::= F(#)
C ::= Calibrated
I ::= Initialized (selected) but has not run

R ::= Running
U ::= Unknown or Uncalibrated
# ::= <NR1>

Examples

Query a specific calibration result: `CAL:RESult? "Channel1""Clock","Amplitude"` might return might return "Channel1:Clock:Clock Amplitude::=S(C),LC(0),P(0),F(0);"

Query all calibration results: `CAL:RESult? "INT::=(C);"` signifying internal calibration completed and passed.

Query a specific area result: `CAL:RESult? "Channel1" "Clock"` might return "Channel1:Clock::=(C);"

Query a specific subsystem result: `CAL:RESult? "Channel1"` might return "Channel1::=(R);"

Query all calibration results of a specific area:
`CAL:RESult? "Channel1","Clock",ALL`
might return "Channel1:Clock:Clock Amplitude::=S(C),LC(0),P(0),F(0);Channel1:Clock:Clock Offset::=S(U),LC(0),P(0),F(0);"

Asking for all calibration results of a specific subsystem: `CAL:RESult? "Channel1",ALL` might return
"Channel1:Dc::=(U);Channel1:Adc::=(U);Channel1:Clock::=(U);
Channel1:Align::=(U);Channel1:Dac::=(U);Channel1:Marker1::=(U);
Channel1:Marker2::=(U);"

CALibration:RESult:TEMPerature? (Query Only)

This command returns the temperature of the last calibration. All temperatures are in °C.

Group

Calibration

Syntax

`CALibration:RESult:TEMPerature?`

Returns

<T> ::= {<NR1>} Returns the temp in °C. Uncalibrated returns an empty string.

Examples

Query a temperature result:

`CAL:RES:TEMP?` might return "INT::=Temp(33),".

CALibration:RESult:TIME? (Query Only)

This command returns the time of the last calibration.

Group Calibration

Syntax `CALibration:RESult:TIME?`

Returns <T> ::= "mm/dd/yyyy hh:mm {A|P}M"

Examples Query a specific time result: `CAL:RES:TIM?` might return "INT::=Time(2/6/2013 8:38:34 AM),".

CALibration:RUNNing? (Query Only)

This command returns the name of the subsystem, area, and procedure in progress. This command can be issued while procedure is in progress.

Group Calibration

Syntax `CALibration:RUNNing?`

Returns A string of colon separated "subsystem", "area:" and "procedure".

Examples `CALIBRATION:RUNNING?` might return "Channel1:Dc:Amplitude" indicating the subsystem, area, and procedure in progress. A return of "" indicates there isn't a currently running procedure.

CALibration:STARt (No Query Form)

This command starts the calibration.

Conditions Setting only works in the active mode for calibration. See the ACTive:MODE command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Calibration

Syntax
```
CALibration:STARt
```

Related Commands ACTive:MODE, CALibration:ABORt

Examples
```
CALIBRATION:START
*OPC?
```
starts the execution of calibration routines. The overlapping command is followed with an Operation Complete query.

CALibration:STATe:FACTory? (Query Only)

This command returns the current factory state of the calibration for the AWG.

- A calibration state will be Calibrated or Uncalibrated.
- Areas will be calibrated when all procedures for that area have been executed and passed.
- Subsystems will be calibrated when all areas for that subsystem are calibrated.
- Each calibrated (as opposed to uncalibrated) state will have a temperature and date time.
- An uncalibrated state will not have a valid temperature or date time and should be ignored.

Conditions Results will be undetermined if there is a calibration procedure in progress.

Group Calibration

Syntax
```
CALibration:STATe:FACTory?[<subsystem>][,<area>]]
```

Arguments
<subsystem> :::= <string>
<area> ::= <string>
<test> ::= <string>

Returns "<State>"

<State> ::= S(C|U) Reflects the "current" state.
C ::= Calibrated
U ::= Uncalibrated

D ::= Date and time
T ::= Temperature in °C

Examples Query the factory calibration state of the system: `CALIBRATION:STATE:FACTORY?` might return "INT::=S(C),D(2/1/2013 12:00:00 AM),T(44)"

Query a specific area state: `CALIBRATION:STATE:FACTORY? "Channel1","Dc"` might return "Channel1:Dc::=S(U),D(1/1/1970 12:00:00 AM),T(0)".

CALibration:STATe:USER? (Query Only)

This command returns the current user state of the calibration for the AWG.

- A calibration state will be Calibrated or Uncalibrated.
- Areas will be calibrated when all procedures for that area have been executed and passed.
- Subsystems will be calibrated when all areas for that subsystem are calibrated.
- Each calibrated (as opposed to uncalibrated) state will have a temperature and date time.
- An uncalibrated state will not have a valid temperature or date time and should be ignored.

Group Calibration

Syntax `CALibration:STATe:USER? [<subsystem>[,<area>]]`

Arguments <subsystem> ::= <string>
<area> ::= <string>

Returns "<State>"

<State> ::= S(C|U) Reflects the "current" state.
C ::= Calibrated
U ::= Uncalibrated
D ::= Date and time
T ::= Temperature in °C

Examples Asking for a specific subsystem state: `CALIBRATION:STATE:USER? "Channel1"` might return "Channel1::=S(C),D(1/1/2013 12:01:52 AM),T(112)"

Query a specific area state: `CALIBRATION:STATE:USER? "Channel1","Dc"` might return "Channel1:Dc::=S(C),D(1/1/2013 12:00:02 AM),T(32)"

CALibration:STOP:STATe? (Query Only)

This command returns the state of the calibration procedure.

Group

Calibration

Syntax

```
CALibration:STOP:STATe?
```

Returns

A single <Boolean> value, {0|1} 1 is stopped and 0 is not stopped.

Examples

`CALIBRATION:STOP:STATE?` might return 1.

CLOCk:ECLock:DIVider

This command sets or returns the divider rate for the external clock.

Conditions

Setting the external clock divider rate forces the external clock multiplier rate to a value of 1.

This command is only valid if the clock source is set to External. See the CLOCk:SOURce command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:ECLock:DIVider <NR1>
CLOCk:ECLock:DIVider?
```

Related Commands

CLOCk:ECLock:MULTiplier, CLOCk:SRATe, CLOCk:SOURce

Arguments

A single <NR1> value that is a power of 2.

Range: 1 to 16777216

`*RST` sets this to 1.

Returns

A single <NR1> value.

Examples

```
CLOCK:ECLOCK:DIVIDER 4
*OPC?
```
sets the external clock divider rate to 4. The overlapping command is followed with an Operation Complete query.

`CLOCK:ECLOCK:DIVIDER?` might returns 4, indicating the external clock divider rate is set to 4.

CLOCk:ECLock:FREQuency

This command sets or returns the expected frequency being provided by the external clock.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:ECLock:FREQuency <NR3>
CLOCk:ECLock:FREQuency?
```

Related Commands

CLOCk:SOURce

Arguments

A single <NR3> value.

Range: 6.25E9 to 12.5E9

`*RST` sets this to 6.25E9

Returns

A single <NR3> value.

Examples

```
CLOCK:ECLOCK:FREQUENCY 10E9
*OPC?
```

sets the expected frequency of the external clock to 10 GHz. The overlapping command is followed with an Operation Complete query.

`LOCK:ECLOCK:FREQUENCY?` might return 10.0000000000E+9, indicating that the expected frequency of the external clock is 10 GHz.

CLOCk:ECLock:FREQuency:ADJust (No Query Form)

This command initiates an adjustment (calibration) to the system clock circuitry.

The adjustment can be run at any time, but if the system detects setting changes that impact clock accuracy, the adjustment is required before any signals can be played.

A message is displayed in the status area when an adjustment is required.

A error message is generated if the adjustment fails.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:ECLock:FREQuency:ADJust
```

Examples

```
CLOCK:ECLOCK:FREQUENCY:ADJUST
*OPC?
```

performs a calibration of the system clock circuitry when using an external clock signal. The overlapping command is followed with an Operation Complete query.

CLOCk:ECLock:FREQuency:DETect (No Query Form)

This command detects the frequency of the signal applied to the Clock In connector and adjusts the system to use the signal. The frequency is detected once each time the command executes.

An error message is generated if no frequency is detected or is out of range.

Conditions

This command is only valid if the clock source is set to External. See the CLOCk:SOURce command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:ECLock:FREQuency:DETect
```

Related Commands

CLOCk:SOURce

Examples

```
CLOCK:ECLOCK:FREQUENCY:DETECT
*OPC?
```
detects the clock frequency applied to the Clock In connector. The overlapping command is followed with an Operation Complete query.

CLOCk:ECLock:MULTiplier

This command sets or returns the multiplier rate of the external clock.

Conditions

Setting the clock multiplier rate forces the clock divider rate to a value of 1.

This command is only valid if the clock source is set to External. See the CLOCk:SOURce command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Control

Syntax

```
CLOCk:ECLock:MULTiplier <NR1>
CLOCk:ECLock:MULTiplier?
```

Related Commands

CLOCk:ECLock:DIVider, CLOCk:SRATe, CLOCk:SOURce

Arguments

A single <NR1> value.

Range: 1, 2, or 4 (AWG70001)
Range: 1 or 2 (AWG70002)

`*RST` sets this to 1.

Returns

A single <NR1> value

Examples

```
CLOCK:ECLOCK:MULTIPLIER 4
*OPC?
```
sets the external clock multiplier to 4. The overlapping command is followed with an Operation Complete query.

`CLOCK:ECLOCK:MULTIPLIER?` might return 1.0000000000, indicating the clock multiplier is set to 1.

CLOCk:EREFerence:DIVider

This command sets or returns the divider rate of the external reference signal when the external reference is variable.

Conditions

Setting the external reference divider rate forces the external reference multiplier rate to a value of 1.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:EREFerence:DIVider <NR1>
CLOCk:EREFerence:DIVider?
```

Arguments

A single <NR1> value that is a power of 2.

Range: 1 to 16777216.

`*RST` sets this to 1.

Returns

A single <NR1> value.

Examples

```
CLOCK:EREFERENCE:DIVIDER 1
*OPC?
```
sets the external reference divider to 1. The overlapping command is followed with an Operation Complete query.

`CLOCK:EREFERENCE:DIVIDER?` might return 1, indicating the divider rate is set to 1.

CLOCk:EREFerence:FREQuency

This command sets or returns the expected frequency of the signal applied to the Reference In connector.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:EREFerence:FREQuency <NRf>
CLOCk:EREFerence:FREQuency?
```

Arguments

A single <NRf> value.

Range: 35 MHz to 250 MHz.

`*RST` sets this to 35 MHz.

Returns

A single <NRf> value.

Examples

```
CLOCK:EREFERENCE:FREQUENCY 35E6
*OPC?
```
sets the expected reference frequency applied to the Reference In connector to be 35 MHz. The overlapping command is followed with an Operation Complete query.

`CLOCK:EREFERENCE:FREQUENCY?` might return 200.0000000000E+6, indicating that the expected frequency of the signal applied to the Reference In connector is set to 200 MHz.

CLOCk:EREFerence:FREQuency:DETect

This command detects the frequency of the signal applied to the Reference In connector and adjusts the system to use the signal. The frequency is detected once each time the command executes.

An error message is generated if no frequency is detected, is out of range, or if the adjustment fails.

This command is only valid when the clock source is external.

Errors are not returned. They are added to the system error queue which can be accessed with SYSTem:ERRor[:NEXT]?.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:EREFerence:FREQuency:DETect
```

Examples

```
CLOCK:EREFERENCE:FREQUENCY:DETECT
*OPC?
```
detects the clock frequency applied to the Reference In connector. The overlapping command is followed with an Operation Complete query.

CLOCk:EREFerence:MULTiplier

This command sets or returns the multiplier rate of the variable external reference signal.

Conditions

Setting the external reference multiplier rate forces the external reference divider rate to a value of 1.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:EREFerence:MULTiplier <NR1>
CLOCk:EREFerence:MULTiplier?
```

Arguments

A single <NR1> value.

Range: 1 to 1000 (limited by the maximum sample rate).

`*RST` sets this to 1.

Returns

A single <NR1> value.

Examples

```
CLOCK:EREFERENCE:MULTIPLIER 50
*OPC?
```
sets the multiplier to 50. The overlapping command is followed with an Operation Complete query.

`CLOCK:EREFERENCE:MULTIPLIER?` might return 100, indicating that the external clock multiplier rate is set to 100.

CLOCk:JITTer

This command sets or returns whether or not low jitter (Jitter Reduction) is enabled on the internal system clock or the clock signal applied to the Reference In connector.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When enabled, the sample rate is limited by clock frequency multiples of 50 MHz.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:JITTer {0|1|OFF|ON}
```

Arguments

0 or OFF disables jitter reduction.

1 or ON enables jitter reduction.
`*RST` sets this to 0.

Returns

A single <Boolean> value, 0 or 1.

Examples

`CLOCK:JITTER ON`
enables the jitter reduction mode for system clock.

`CLOCK:JITTER?` might return 0, indicating that the jitter reduction mode is not enabled for the system clock.

CLOCk:OUTPut:FREQuency? (Query Only)

This command returns the frequency of the output clock on the Clock Out connector.

Conditions

If clock output state is not enabled, 0.0000 is returned.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Control

Syntax

```
CLOCk:OUTPut:FREQuency?
```

Returns

A single <NRf> value.

If the clock output state is not enabled, 0.0000 is returned.
If the clock output state is enabled, a value between 6.25 GHz and 12.5 GHz is returned, depending on the sample rate.

Examples

`CLOCK:OUTPUT:FREQUENCY?` might return 12.4999955600E+9, indicating that the clock output is enabled and the frequency is essentially 12.5 GHz.

CLOCk:OUTPut[:STATe]

This command sets or returns the state of the output clock. Enabling Clock Out provides a high speed clock (that is related to sample rate) to drive other devices or to measure.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is the master, the command choice is limited to ON or 1.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:OUTPut[:STATe] {0|1|OFF|ON}
CLOCk:OUTPut[:STATe]?
```

Related Commands

CLOCk:SOURce

Arguments

A single <Boolean> value.

0 or OFF disables the clock out.
1 or ON enables the clock out.

`*RST` sets this to 0.

Returns

A single <Boolean> value, 0 or 1.

Examples

`CLOCK:OUTPUT:STATE ON`
sets the Clock Output to ON.

`CLOCK:OUTPUT:STATE?` might return 1, indicating that the output clock is enabled.

CLOCk:PHASe[:ADJust]

This command sets or returns the phase adjustment to synchronize multiple AWGs. Setting the phase adjusts the phase of all signal outputs relative to the system clock.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Clock

Syntax

```
CLOCk:PHASe[:ADJust] <NR1>
CLOCk:PHASe[:ADJust]?
```

Related Commands

CLOCk:SOUT[:STATe]

Arguments

A single <NR1> value.

Range: –10800 to 10800.

`*RST` sets this to 0.

Returns

A single <NR1> value.

Examples

```
CLOCK:PHASE:ADJUST 100
*OPC?
```
sets the clock phase to 100 degrees. The overlapping command is followed with an Operation Complete query.

`CLOCK:PHASE:ADJUST?` might return 100, indicating the clock phase is set to 100 degrees.

CLOCk:SOURce

This command sets or returns the source of the clock.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Clock

Syntax

```
CLOCk:SOURce {INTernal|EFIXed|EVARiable|EXTernal}
CLOCk:SOURce?
```

Arguments INTernal - clock signal is generated internally and the reference frequency is derived by the internal oscillator.

EFIXed – clock is generated internally and the reference frequency is derived from a fixed 10 MHz reference supplied at the Reference In connector.

EVARiable – clock is generated internally and the reference frequency is derived from a variable reference supplied at the Reference In connector.

EXTernal – clock signal supplied by the Clock In connector and the reference frequency is derived from the internal precision oscillator.

`*RST` sets this to INT.

Returns {INT, EFIX, EVAR, EXT}

Examples

```
CLOCK:SOURCE INTERNAL
*OPC?
```

sets the clock source to internal. The overlapping command is followed with an Operation Complete query.

`CLOCK:SOURCE?` might return EFIX, indicating that the clock source is set to use the Reference In connector.

CLOCk:SOUT[:STATe]

This command sets or returns the state of the Sync Clock Out output.

Group

Clock

Syntax

```
CLOCk:SOUT[:STATe] {0|1|OFF|ON}
CLOCk:SOUT[:STATe]?
```

Arguments

0 or OFF disables the Sync Clock Out.
1 or ON enables the Sync Clock Out.

`*RST` sets this to 0.

Returns

A single <Boolean> value, 0 or 1.

Examples

`CLOCK:SOUT:STATE 1` sets the Sync Clock Out output to ON.

`CLOCK:SOUT:STATE?` might return 0, indicating that the Sync Clock Out output is off.

CLOCk:SRATe

This command sets or returns the sample rate for the clock.

Conditions

This command is not valid when CLOCk:SOURce is set to EXTernal.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Clock

Syntax

```
CLOCk:SRATe <NRf>
CLOCk:SRATe?
```

Related Commands

CLOCk:SOURce

Arguments A single <NRf> value.

Range:	AWG70001A	1.49 kS/s to 50 GS. When Clock Source is set to External: 4 times the External Clock In frequency
	AWG70002A	1.49 kS/s to 25 GS. When Clock Source is set to External: 2 times the External Clock In frequency

`*RST` sets this to the maximum value.

Returns A single <NR3f> value.

Examples

```
CLOCK:SRATE 5E8
*OPC?
```

sets the clock sample rate to 500 MS/s. The overlapping command is followed with an Operation Complete query.

`CLOCK:SRATE?` might return 25.0000000000E+9, indicating the clock sample rate is set to 25 GS/s.

*CLS (No Query Form)

This command clears all event registers and queues. (See page 3-1, *Status and events*.)

Group IEEE mandated and optional

Syntax `*CLS`

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

DIAGnostic:ABORt (No Query Form)

This command attempts to stop the current diagnostic test and stops the execution of any additional selected tests.

This may result in loss of logging information collected for the current test that responds to the abort event.

Conditions This command requires that ACTive:MODE is set to DIAGnostic.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Diagnostic

Syntax

```
DIAGnostic:ABORt
```

Related Commands

ACTive:MODE, DIAGnostic:STARt

Examples

```
DIAGNOSTIC:ABORT
*OPC?
```

stops the current diagnostic test. The overlapping command is followed with an Operation Complete query.

DIAGnostic:CATalog? (Query Only)

This command returns the list of all diagnostic tests per selected type per subsystems, areas, or ALL. All tests are grouped by areas. All areas are grouped by subsystems. The available subsystems, areas, and tests depend on the type of testing (such as POST only or Full diagnostics).

The selected type is set with the command DIAGnostic:TYPE.

Conditions

NOTE. *This can be queried anytime and does not depend on ACTive:MODE being set to DIAGnostic.*

It does however depend on the DIAG:TYPE which can only be changed if the ACTive:MODE is set to DIAGnostic.

Group

Diagnostic

Syntax

```
DIAGnostic:CATalog?  [{ALL|<subsystem>}[,{ALL|<area>}]]
```

Related Commands

DIAGnostic:TYPE

Arguments

This works in the current context as set by the DIAG:TYPE command.

- ALL – Keyword or as a string.
- <subsystem> – A subsystem as a string.
- <area> – An area as a string.

If there are no parameters, then the list of subsystems is returned.

If there is a valid subsystem parameter, then the list of areas for that subsystem is returned.

If the subsystem parameter is "ALL", then all the tests of all the areas of all the subsystems are returned. Each test is prefixed with "<subsystem>:<area>:" and separated by a comma. Lists are always in priority of the desired execution.

If the area parameter is "ALL", then all the tests of all the areas for a specified subsystem are returned. Each test is prefixed with "<area>:" and separated by a comma. Lists are always in priority of the desired execution.

If the subsystem and area parameters are valid, then the list of tests for that subsystem and area are returned.

Returns

String of diagnostic "subsystems", "areas" and/or "procedures" separated by commas.

Examples

`DIAGNOSTIC:CATALOG?` might return "System,Clock1,Channel1,Channel2"

`DIAGNOSTIC:CATALOG? "Channel1"` might return "Host Communications,Waveform Memory,Real Time,Marker1,Marker2"

`DIAGNOSTIC:CATALOG? "Channel1","Waveform Memory"` might return "Calibration,Data Lines,Address Lines,Cells"

`DIAGNOSTIC:CATALOG? "ALL"` might return "…,Channel1:Waveform Memory:Calibration,Channel1:Waveform Memory:Data Lines,Channel1:Waveform Memory:Address Lines,Channel1:Waveform Memory:Cells,…"

DIAGnostic:CONTrol:COUNt

This command sets or returns the number of loop counts used when the loop mode is set to COUNt. See DIAGnostic:CONTrol:LOOP.

Conditions

DIAGnostic:CONTrol:LOOP must be set to COUNt.

The set form of this command requires that ACTive:MODE is set to DIAGnostic.

Group

Diagnostic

Syntax

```
DIAGnostic:CONTrol:COUNt <NR1>
DIAGnostic:CONTrol:COUNt?
```

Related Commands

ACTive:MODE, DIAGnostic:CONTrol:LOOP

Arguments

A single <NR1> value.

Range: ≥0 to 1073741823 or 0x3FFFFFFF(2^30 – 1). A count of 0 is the same as a count of 1.

`*RST` sets this to 0.

Returns

A single <NR1> value.

Examples

`DIAGNOSTIC:CONTROL:COUNT 1000` sets the diagnostic looping to occur for 1000 times before exiting.

`DIAGNOSTIC:CONTROL:COUNT?` might return 1000 indicating that the diagnostic tests will loop 1000 times before halting.

DIAGnostic:CONTrol:HALT

This command sets or returns whether the next execution of diagnostics looping stops on the first diagnostic failure that occurs or continues to loop on the selected set of diagnostic functions.

Group

Diagnostic

Syntax

```
DIAGnostic:CONTrol:HALT {0|1|OFF|ON}
```

Arguments

0 or OFF disables the halt function, allowing the AWG to continue to loop on the entire set of diagnostics, even if a diagnostic failure occurs.

1 or ON enables the halt function, causing the execution of diagnostics looping to halt at the first diagnostic failure that occurs.

`*RST` sets this to 0.

Returns

A single <Boolean> value, 0 or 1.

Examples

`DIAGNOSTIC:CONTROL:HALT ON` enables the halt function, causing the execution of diagnostics looping to halt at the first diagnostic failure.

`DIAGNOSTIC:CONTROL:HALT?` might return 0, indicating that the halt function is disabled.

DIAGnostic:CONTrol:LOOP

This command sets or returns whether the next start of diagnostics runs once, runs continuous loops, or loops for a number times for the selected set of tests. All loops may be affected by the DIAGnostic:CONTrol:HALT command which determines what happens if an error occurs.

Conditions

This command requires that ACTive:MODE is set to DIAGnostic.

Group

Diagnostic

Syntax

```
DIAGnostic:CONTrol:LOOP {ONCE|CONTinuous|COUNt}
DIAGnostic:CONTrol:LOOP?
```

Related Commands

ACTive:MODE, DIAGnostic:CONTrol:COUNt, DIAGnostic:CONTrol:HALT

Arguments

`ONCE` disables the loop function, causes the execution of selected test(s), which may be one or more, of diagnostics once and then halt.

`CONTinuous` enables the loop function, causing the execution of diagnostics to continuously loop.

`COUNt` enables the loop function, causing the execution of diagnostics to loop for a predefined count. Exit of the loop happens when the predefined loop count occurs.

`*RST` sets this to ONCE.

Returns

ONCE
CONT
COUN

Examples

`DIAGNOSTIC:CONTROL:LOOP CONTinuous` enables the diagnostics loop continuously.

`DIAGNOSTIC:CONTROL:LOOP?` might return ONCE indicating that the test or tests will execute a single time before halting.

DIAGnostic:DATA? (Query Only)

This command returns the results of last executed tests for the NORMal diagnostic type in the form of a numeric value of 0 for no errors or -330 for one or more tests failed.

Additional error details can be found by using the subsystem, area, and test queries such as `DIAGnostic:RESult?  <subsystem>[,<area>[,<test>]]`.

Group

Diagnostic

Syntax

```
DIAGnostic:DATA?
```

Related Commands

DIAGnostic:TYPE, DIAGnostic:RESult?

Returns

A single <NR1> value.

0 indicates no error.
–330 indicates that the self test failed.

Examples

`DIAGNOSTIC:DATA?` might return 0, which indicates that the diagnostics completed without any errors.

DIAGnostic[:IMMediate]

This command executes all of the NORMal diagnostic tests. The query form of this command executes all of the NORMal diagnostics and returns the results in the form of numeric of values of 0 for no errors or -330 for one or more tests failed.

This changes the active mode to DIAGnostic, if necessary, and returns back to the original active mode when done.

This makes a single pass of all of the NORMal diagnostics.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Diagnostic

Syntax

```
DIAGnostic[:IMMediate]
DIAGnostic[:IMMediate]?
```

Related Commands

DIAGnostic:DATA?, DIAGnostic:RESult?

Returns

A single <NR1> value.

0 indicates no error.

–330 indicates that the test failed.

Examples

`DIAGNOSTIC` executes the NORMal test routines.

`DIAGNOSTIC?` executes the NORMal test routines and might return 0, indicating there are no errors.

DIAGnostic:LOG? (Query Only)

This command returns a string of continuous concatenated test results. The start time is recorded for each of the selected tests.

This command can be issued at any time including while diagnostics are in progress.

Conditions

The return string is limited to only the first 64K of text, which can cause a loss of results. Use the DIAGnostic:LOG:CLEar command to start a fresh log.

Group Diagnostic

Syntax `DIAGnostic:LOG?`

Related Commands DIAGnostic:LOG:CLEar

Returns <string> ::= "<Started timestamp><LF delimiter><test name and result>[<LF delimiter><test name and result>]

Examples `DIAGNOSTIC:LOG?` might return "Channel1:Memory:Data Lines Started 6/14/2011 10:19 AM Channel1:Memory:DataLines PASS Channel1:Memory:Address Lines Started 6/14/2011 10:20 AM Channel1:Memory:Address Lines PASS"

DIAGnostic:LOG:CLEar (No Query Form)

This command clears the diagnostics results log.

Conditions This command requires that ACTive:MODE is set to DIAGnostic.

Group Diagnostic

Syntax `DIAGnostic:LOG:CLEar`

Related Commands ACTive:MODE

Examples `DIAGNOSTIC:LOG:CLEAR` deletes the contents of the diagnostics log.

DIAGnostic:LOG:FAILuresonly

This command sets or returns the flag that controls the amount of result information saved into the diagnostic log. This controls all tests that pass or fail or only tests that fail.

The flag must be set before starting the diagnostic tests to obtain the expected data.

Conditions The set form of this command only works when ACTive:MODE is set to DIAGnostic.

Group Diagnostic

Syntax

```
DIAGnostic:LOG:FAILuresonly {0|1|OFF|ON}
DIAGnostic:LOG:FAILuresonly?
```

Related Commands ACTive:MODE, DIAGnostic:LOG?, DIAGnostic:LOG:CLEar

Arguments 0 or OFF disables the failure only mode.
1 or ON enables the failure only mode.

`*RST` sets this to 0.

Returns A single <Boolean> value, 0 or 1.

Examples `DIAGNOSTIC:LOG:FAILURESONLY OFF` disables the failure only mode.

`DIAGNOSTIC:LOG:FAILURESONLY 1` enables the failure only mode.

`DIAGNOSTIC:LOG:FAILURESONLY?` might return 1, showing the failure only mode is enabled.

DIAGnostic:LOOPs? (Query Only)

This command returns the number of times that the selected diagnostics set was completed during the current running or the last diagnostic running of the set. The current loop is reset after every start.

This command can be issued while diagnostics are still in progress.

Group Diagnostic

Syntax

```
DIAGnostic:LOOPs?
```

Returns A single <NR1> value, representing the number of loops completed.

Examples `DIAGNOSTIC:LOOPS?` might return 5, indicating that the selected set of diagnostics has completed five times.

DIAGnostic:RESult? (Query Only)

This command returns the status about the results of the last start of a set of selected tests.

An individual test result can have a status of Pass, Fail or Running.

Status for an area or a subsystem have the following requirements:

- The results only reflect the "selected" tests.
- The selected tests have to have results of pass or fail or be in the running state.
- Only selected tests in an area or subsystem contribute to the result. As an example, if 3 of the 4 tests in an area has been selected, then only those 3 contribute to the "area" result. If only 2 of the selected 3 have run and completed (a stop event occurred) then only those 2 contribute to the result.
- If all contributors have passed, then the result is passed. If any contributor has failed, then the result is failed. If any contributor is running, then the result is running.

Group

Diagnostic

Syntax

```
DIAGnostic:RESult?  [{ALL|<path>}]
```

Arguments

ALL: Keyword as a string.

<path> = <subsystem>[,<area>[,<test>]]
<subsystem>: One of the strings listed by DIAGnostic:CATalog?
<area>: One of the strings listed by DIAGnostic:CATalog? <subsystem>
<test>: One of the strings listed by DIAGnostic:CATalog? <subsystem>,<area>

Returns

"<result record>"
<result record>: = <subsystem>:[<area>:[<test>:]] <details>

<details>: <Status>,<Loop Count>,<Pass>,<Fail>
<Status>: S(P|F|R) Reflects the "current" or "last" state. When the status reflects only the subsystem or area, then an F for Fail will be set for any of the tests that have failed.
<Loop Count> ::= LC(#)
<Pass> ::= P(#)
<Fail> ::= F(#)
P ::= Pass
F ::= Fail
R ::= Running
#: <NR1>

Examples

Asking for a specific test result:
`DIAGNOSTIC:RESULT? "Channel1","Waveform Memory","Calibration"` might return "Channel1:Waveform Memory:Calibration::=S(F),LC(1),P(0),F(1).

Asking for a specific area result:
`DIAGNOSTIC:RESULT? "Channel1","Waveform Memory"` might return "Channel1:Waveform Memory::=S(F).

Asking for a specific subsystem result:
`DIAGNOSTIC:RESULT? "Channel1"` might return "Channel1::=S(F).

Asking for all test results of a specific area:

`DIAGNOSTIC:RESULT? "Channel1","Waveform Memory",ALL` might return "Channel1:Waveform Memory:Calibration::=S(F),LC(1),P(0),F(1);Channel1:Waveform Memory:Data Lines::=S(P),LC(1),P(1),F(0);Channel1:Waveform Memory:Address Lines::=S(P),LC(1),P(1),F(0);".

DIAGnostic:RESult:TEMPerature? (Query Only)

This command returns the temperature from the results of the last start of a set of selected tests. All temperatures will be in °C.

Temperature for an area or subsystem have the following requirements.

- The temperature only reflects the "selected" tests.
- The "selected" tests must have results of pass or fail. As an example, if 3 of the 4 tests in an area has been selected, then only those 3 contribute to the "area" result. If only 2 of the selected 3 have run and completed (a stop event occurred) then only those 2 contribute to the result.
- The highest temperature is returned when the results for more than one test is requested (as in an area). The time will also be recorded for the highest temperature and may be found with the Diag:Result:Time? query.

Group

Diagnostic

Syntax

```
DIAGnostic:RESult:TEMPerature?
"<subsystem>"[,"<area>"[,"<test>"]]
```

Related Commands

DIAGnostic:RESult:TIME?

Arguments

<subsystem> ::= <string>
<area> ::= <string>
<test> ::= <string>

Returns

"<temperature>"

<temperature> ::= <string>
<string> ::= Ascii text where a number will be in °C or "NA".

Examples

Asking for a specific temperature result:
`DIAGNOSTIC:RESULT:TEMPERATURE? "Channel1","Waveform Memory","Calibration"` might return "32".

DIAGnostic:RESult:TIME? (Query Only)

This command returns the time from the results of the last start of a set of selected tests. Time is returned as a date time string as in the following example of "3/14/2013 10:19 AM".

Time for an area or subsystem have the following requirements:

- The time only reflects the "selected" tests.
- The "selected" tests must have results of pass or fail. As an example, if 3 of the 4 tests in an area has been selected, then only those 3 contribute to the "area" result. If only 2 of the selected 3 have run and completed (a stop event occurred) then only those 2 contribute to the result.
- The time returned, which is associated with the highest temperature of any selected test, is returned when the results for more than one test is requested as in an area.

Group

Diagnostic

Syntax

```
DIAGnostic:RESult:TIME? "<subsystem>"[,"<area>"[,"<test>"]]
```

Arguments

<subsystem> ::= <string>
<area> ::= <string>
<test> ::= <string>

Returns

"<time>"

<time> ::= <string>
<string> ::= Ascii text in the form of mm/dd/yy followed by the time in hr:min as in the example of "3/14/2013 10:19 AM".

Examples

`DIAGNOSTIC:RESULT:TIME? "Channel1","Waveform Memory","Calibration"` might return "Channel1:Waveform Memory:Calibration::=Time(2/5/2013 4:51:53 PM)".

DIAGnostic:RUNNing? (Query Only)

This command returns the name of the subsystem, area, and test of the current diagnostic test. This command can be issued at any time.

Group

Diagnostic

Syntax

`DIAGnostic:RUNNing?`

Returns

String of the path of the test which includes subsystem, area and test names of currently running test. If there is no currently running test, then the string is empty.

Examples

`DIAGNOSTIC:RUNNING?` might return "Channel1:Waveform Memory:Calibration" indicating the currently running diagnostic test by the subsystem name, area name, and test name.

DIAGnostic:SELect (No Query Form)

This command (no query form) selects one or more tests of the current test list. Tests can be selected by the keyword ALL, by "subsystem", by "area", or by "test". The selection by "area" requires "subsystem" and a "test" requires both the "subsystem" and "area".

NOTE. *The keywords may be in quotes but is not necessary.*

This command requires that ACTive:MODE is set to DIAGnostic. If not, the following error is generated:

- -300,"Device-specific error; Not in Diagnostics mode - diag:sel ""Channel1"""

If in the proper active of DIAGnostic, then an invalid string generates the following error:

- -220,"Parameter error; Invalid subsystem - diag:sel ""Channel2"""

Group

Diagnostic

Syntax

`DIAGnostic:SELect {ALL|<path>}`

Related Commands

ACTive:MODE, DIAGnostic:UNSelect

Arguments

ALL selects all available tests

<path> ::= <subsystem>[,<area>[,<test>]]
<subsystem> One of the strings listed by the DIAGnostic:CATalog? command.
<area> One of the strings listed by the DIAGnostic:CATalog?<subsystem> command.
<test> One of the strings listed by the DIAGnostic:CATalog?<subsystem>,<area> command.

Examples

`DIAGNOSTIC:SELECT All` selects all available tests.

`DIAGNOSTIC:SELECT "System"` selects all tests in System subsystem.

`DIAGNOSTIC:SELECT "Clock1","Clock Internal"` selects all tests in the Clock Internal area of the Clock1 subsystem.

`DIAGNOSTIC:SELECT "Clock1","Clock Internal","ALL"` selects all tests in the Clock Internal area of the Clock1 subsystem.

`DIAGNOSTIC:SELECT "Channel1","Waveform Memory","Data Lines"` selects one test.

DIAGnostic:SELect:VERify? (Query Only)

This command returns selection status of one specific test. A specific test requires the "subsystem", "area", and "test".

This is context sensitive and is dependent on the type as set with the command DIAGnostic:TYPE.

Group

Diagnostic

Syntax

```
DIAGnostic:SELect:VERify?  <subsystem>,<area>,<test>
```

Related Commands

DIAGnostic:TYPE, DIAGnostic:UNSelect

Arguments

<subsystem> One of subsystems listed in by the system:catalog
<area> One of the areas listed by the area:catalog
<test> One of the tests listed by the test:catalog

Returns

A single <Boolean> value, 0 or 1. 0 is not selected, 1 is selected.

Examples

`DIAGNOSTIC:SELECT "Channel1","Waveform Memory","Data Lines"` selects one test.

`DIAGNOSTIC:SELECT:VER? "Channel1","Waveform Memory","Data Lines"` returns 1.

DIAG:UNS "Channel1", "Waveform Memory", "Data Lines" unselects one test.

DIAG:SEL:VER? "Channel1", "Waveform Memory", "Data Lines" returns 0.

DIAGnostic:STARt (No Query Form)

This command starts the execution of the selected set of diagnostic tests.

Conditions

This command requires that ACTive:MODE is set to DIAGnostic.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Diagnostic

Syntax

```
DIAGnostic:STARt
```

Related Commands

ACTive:MODE, DIAGnostic:ABORt, DIAGnostic:STOP

Examples

```
DIAGNOSTIC:START
*OPC?
```

starts the execution of the selected set of tests. The overlapping command is followed with an Operation Complete query.

DIAGnostic:STOP (No Query Form)

This command stops the diagnostic tests from running, after the diagnostic test currently in progress completes.

This also terminates diagnostic test looping.

Conditions

This command requires that ACTive:MODE is set to DIAGnostic.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Diagnostic

Syntax

```
DIAGnostic:STOP
```

Related Commands ACTive:MODE, DIAGnostic:ABORt, DIAGnostic:STOP, DIAGnostic:STOP:STATe?

Examples
```
DIAGNOSTIC:STOP
*OPC?
```
stops the execution of the selected set of tests. The overlapping command is followed with an Operation Complete query.

DIAGnostic:STOP:STATe? (Query Only)

This command returns the current state of diagnostic testing.

Group Diagnostic

Syntax `DIAGnostic:STOP:STATe?`

Returns A single <Boolean> value, 0 or 1. 1 represents a stopped state and 0 represents running state.

Examples `DIAGNOSTIC:STOP:STATE?` might return 1, indicating that testing has stopped.

DIAGnostic:TYPE

This command sets or returns the diagnostic type. The diagnostics work on a list of tests that support different types of testing.

This sets the context for other commands such as selecting a test to run.

Conditions This command requires that ACTive:MODE is set to DIAGnostic. If not, the following error is generated:

- -300,"Device-specific error;Not in Diagnostics mode - diag:type post"'

The diagnostic type can only be changed if no testing is currently in progress. If there is, the following error is generated:

- -300,"Device-specific error;Diagnostics procedures still in progress - diag:type post"'

Group Diagnostic

Syntax

```
DIAGnostic:TYPE {NORMal|POST}
DIAGnostic:TYPE?
```

Related Commands DIAGnostic:SELect, DIAGnostic:UNSelect, DIAGnostic:STARt

Arguments `NORMal` – Normal operating mode
`POST` – Power On Self Test

`*RST` sets this to NORM.

Returns `NORM`
`POST`

Examples `DIAGNOSTIC:TYPE NORMAL` sets the AWG to normal operating mode.

`DIAGNOSTIC:TYPE?` might return NORM.

DIAGnostic:TYPE:CATalog? (Query Only)

This command returns a list of diagnostic types available.

NOTE. *This can be queried anytime and does not depend on ACTive:MODE being set to DIAGnostic.*

Group Diagnostic

Syntax `DIAGnostic:TYPE:CATalog?`

Returns `NORM` – Normal operating mode
`POST` – Power On Self Test

Examples `DIAGNOSTIC:TYPE:CATALOG?` might return NORM.

DIAGnostic:UNSelect (No Query Form)

This command unselects one or more tests of the current test list.

Tests can be unselected by the keyword ALL, or by "subsystem", or by "area", or by "test". To unselect an "area", "subsystem" is required. To unselect a "test" requires both the "subsystem" and "area".

Conditions This command requires that ACTive:MODE is set to DIAGnostic.

Group Diagnostic

Syntax

```
DIAGnostic:UNSelect {ALL|<"subsystem">,<"area">,<"test">}
```

Related Commands ACTive:MODE, DIAGnostic:SELect

Arguments

- <subsystem> One of subsystems listed by the system:catalog
- <area> One of the areas listed by the area:catalog
- <test> One of the tests listed by the test:catalog
- ALL selects all available tests

Table 2-26: DIAGnostic:UNSelect arguments

subsystem	area	test
System	System Interface	
Clock1	Clock Internal	Communications Internal Reference Attenuator Check
Channel1	Host Communications	Local Bus Serial PCIe Communications Host Bus
	Waveform Memory	Calibration Data Lines Address Lines Cells
	Real Time	Real Time Clock Real Time Trigger Icc Dac Connect Real Time Alignment

Table 2-26: DIAGnostic:UNSelect arguments (cont.)

subsystem	area	test
	Marker1	Communications Adc Offset Negative Offset Positive Offset Amplitude Crossing Point
	Marker2	Communications Adc Offset Negative Offset Positive Offset Amplitude Crossing Point

Examples

`DIAGNOSTIC:UNSELECT "ALL"` unselects all available tests.

`DIAGNOSTIC:UNSELECT "System"` unselects all the tests in System subsystem.

`DIAGNOSTIC:UNSELECT "Channel1","Host Communications"` unselects all the tests in the Host Communications area of in the Channel1 subsystem.

`DIAGNOSTIC:UNSELECT "Channel1","Host Communications","ALL"` unselects all the tests in Host Communications area of the Channel1 subsystem.

`DIAGNOSTIC:UNSELECT "Channel1","Host Communications", "Local Bus"` unselects the single test named Local Bus in the Host Communications area of the Channel1 subsystem.

DISPlay[:PLOT][:STATe]

This command minimizes or restores the plot's display area on the Home screen's channel window of the AWG. This command only minimizes or restores the display area; it does not close the window.

Plots in the Function generator window are not affected.

Group

Display

Syntax

```
DISPlay[:PLOT][:STATe] {0|1|OFF|ON}
DISPlay[:PLOT][:STATe]?
```

Arguments

0 or OFF minimizes the plot display.
1 or ON restores the plot display.

`*RST` sets this to 1.

Returns

A single <NR1> value 0 or 1.

Examples

`DISPLAY:STATE 0` minimizes the plots on the Home screen window.

`DISPLAY:STATE?` might return 1, indicating that the plot display area on the Home screen is not minimized.

*ESE

This command sets or returns the status of Event Status Enable Register (ESER). (See page 3-1, *Status and events*.)

Group

IEEE mandated and optional

Syntax

```
*ESE <NR1>
*ESE?
```

Related Commands

*CLS, *ESR?, *SRE, *STB?

Arguments

A single <NR1> value.

Returns

A single <NR1> value.

Examples

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

`*ESE?` might return 177.

*ESR? (Query Only)

This command returns the status of Standard Event Status Register (SESR). (See page 3-1, *Status and events*.)

Group

IEEE mandated and optional

Syntax

```
*ESR?
```

Related Commands

*CLS, *ESE, *SRE, *STB?

Returns

A single <NR1> value.

Examples

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

FGEN[:CHANnel[n]]:AMPLitude

This command sets or returns the function generator's waveform amplitude value of the selected channel.

Group

Function generator

Syntax

```
FGEN[:CHANnel[n]]:AMPLitude <NRf>
FGEN[:CHANnel[n]]:AMPLitude?
```

Related Commands

INSTrument:MODE, FGEN[:CHANnel[n]]:HIGH, FGEN[:CHANnel[n]]:LOW, FGEN[:CHANnel[n]]:OFFSet

Arguments

A single <NRf> value.

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)
Range: 0 to 500 mV.

`*RST` sets this to 500 mV.

Returns

A single <NRf> value.

Examples

`FGEN:CHANNEL1:AMPLITUDE 0.35` sets the function generator output for channel 1 to 350 mV_{pp}.

`FGEN:CHANNEL1:AMPLITUDE?` might return 250.0000000000E-3, indicating that the function generator output for channel 1 is set to 250 mV.

FGEN[:CHANnel[n]]:DCLevel

This command sets or returns the DC level of the generated waveform of the selected channel.

Conditions

If the value exceeds the designated maximum or minimum offset, then the respective max/min values are used.

Group

Function generator

Syntax

```
FGEN[:CHANnel[n]]:DCLevel <NRf>
FGEN[:CHANnel[n]]:DCLevel?
```

Arguments

A single <NR3> value.

Range: –250 mV to 250 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1)

`*RST` sets this to 0.

Returns

A single <NRf> value.

Examples

`FGEN:CHANNEL1:DCLEVEL 0.12` sets the function generator DC level for channel 1 to 120 mV.

`FGEN:CHANNEL1:DCLEVEL?` might return 250.0000000000E-3, indicating that the function generator DC level for channel 1 is set to 250 mV.

FGEN[:CHANnel[n]]:FREQuency

This command sets or returns the function generator's waveform frequency.

Conditions If the value entered is higher than the designated maximum frequency or lower than the designated minimum, then the respective max/min values are used.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:FREQuency <NRf>
FGEN[:CHANnel[n]]:FREQuency?
```

Related Commands INSTrument:MODE

Arguments A single <NRf> value.

Range: 1 Hz to 50 MHz.

`*RST` sets this to 1.2 MHz.

Returns A single <NRf> value.

Examples `FGEN:CHANNEL:FREQUENCY 1.25E6` sets the function generator frequency to 1.25 MHz.

`FGEN:CHANNEL:AMPLITUDE?` might return 1.2000000000E+6, indicating that the function generator frequency is set to 1.2 MHz.

FGEN[:CHANnel[n]]:HIGH

This command sets or returns the function generator's waveform high voltage value of the selected channel.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:HIGH <NRf>
FGEN[:CHANnel[n]]:HIGH?
```

Related Commands INSTrument:MODE, FGEN[:CHANnel[n]]:AMPLitude, FGEN[:CHANnel[n]]:LOW, FGEN[:CHANnel[n]]:OFFSet

Arguments

A single <NRf> value.

Range: –250 mV to 250 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 250 mV.

Returns

A single <NRf> value.

Examples

`FGEN:CHANNEL1:HIGH 0.25` sets the function generator waveform high voltage value for channel 1 to 250 mV.

`FGEN:CHANNEL1:HIGH?` might return 200.0000000000E-3, indicating that the function generator waveform high voltage value for channel 1 is 200 mV.

FGEN[:CHANnel[n]]:LOW

This command sets or returns the function generator’s waveform low voltage value of the selected channel.

Group

Function generator

Syntax

```
FGEN[:CHANnel[n]]:LOW <NRf>
FGEN[:CHANnel[n]]:LOW?
```

Related Commands

INSTrument:MODE, FGEN[:CHANnel[n]]:AMPLitude, FGEN[:CHANnel[n]]:HIGH, FGEN[:CHANnel[n]]:OFFSet

Arguments

A single <NRf> value.

Range: -250 mV to 250 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to -250 mV.

Returns

A single <NRf> value

Examples

`FGEN:CHANNEL1:LOW -0.25` sets the function generator waveform low voltage value for channel 1 to -250 mV.

`FGEN:CHANNEL1:LOW?` might return -200.0000000000E-3, indicating that the function generator waveform low voltage value for channel 1 is -200 mV.

FGEN[:CHANnel[n]]:OFFSet

This command sets or returns the function generator's waveform offset value of the selected channel.

If the offset value is higher than the designated maximum offset or lower than the designated minimum offset, then the respective max/min values are used.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:OFFSet <NR3>
FGEN[:CHANnel[n]]:OFFSet?
```

Related Commands INSTrument:MODE, FGEN[:CHANnel[n]]:AMPLitude, FGEN[:CHANnel[n]]:HIGH, FGEN[:CHANnel[n]]:LOW

Arguments A single <NR3> value.

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 0.

Returns A single <NR3> value.

Examples `FGEN:CHANNEL1:OFFSET 0.1` sets the function generator offset of channel 1 to 100 mV.

`FGEN:CHANNEL1:OFFSET?` might return 100.0000000000E-3, indicating that the function generator offset of channel 1 is 100 mV.

FGEN[:CHANnel[n]]:PHASe

This command sets or returns the function generator's waveform phase value of the selected channel.

Conditions If the value is higher than the designated maximum phase or lower than the designated minimum, then the respective max/min values are used.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:PHASe <NRf>
FGEN[:CHANnel[n]]:PHASe?
```

Related Commands INSTrument:MODE

Arguments A single <NRf> value.

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)
Range: –180.0 degrees to +180.0 degrees.

`*RST` sets this to 0.

Returns A single <NRf> value.

Examples `FGEN:CHANNEL1:PHASE 10` sets the phase of the function generator for channel 1 to 10°.

`FGEN:CHANNEL1:PHASE?` might return 0.0000, indicating the function generator phase is set to 0° for channel 1.

FGEN[:CHANnel[n]]:SYMMetry

This command sets or returns the function generator's triangle waveform symmetry value of the selected channel.

Conditions If the value is higher than the designated maximum symmetry value or lower than the designated minimum, then the respective max/min values are used.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:SYMMetry <NR1>
FGEN[:CHANnel[n]]:SYMMetry?
```

Related Commands INSTrument:MODE

Arguments A single <NR1> value.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)
Range: 0 to 100%.

`*RST` sets this to 100.

Returns A single <NR1> value

Examples `FGEN:CHANNEL1:SYMMETRY 10` sets the symmetry to 10%.

`FGEN:CHANNEL1:SYMMETRY?` might return 100, indicating the symmetry is set to 100%.

FGEN[:CHANnel[n]]:TYPE

This command sets or returns the function generator's waveform type (shape) of the selected channel.

Group Function generator

Syntax

```
FGEN[:CHANnel[n]]:TYPE
{SINE|SQUare|TRIangle|NOISe|DC|GAUSsian|EXPRise|EXPDecay|NONE}
FGEN[:CHANnel[n]]:TYPE?
```

Arguments {SINE|SQUare|TRIangle|NOISe|DC|GAUSsian|EXPRise|EXPDecay|NONE}
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to SINE.

Returns {SINE|SQU|TRI|NOIS|DC|GAUS|EXPR|EXPD|NONE}

Examples `FGEN:CHANNEL1:TYPE SINE` sets the function generator waveform type for channel 1 to a Sinewave.

`FGEN:CHANNEL1:TYPE?` might return SINE, indicating that the function generator waveform type for channel 1 is set to Sinewave.

FGEN:COUPle:AMPLitude

This command sets or returns the coupling mode of the function generator's waveform amplitude controls between channel 1 and channel 2 of a two channel AWG.

The set form of this command forces the channel 2 amplitude settings to match channel 1. After the initial coupling of the settings, changes made to either channel 1 or channel 2 amplitude settings affect both channels.

Group Function generator

Syntax

```
FGEN:COUPle:AMPLitude {0|1|OFF|ON}
FGEN:COUPle:AMPLitude?
```

Arguments

0 or OFF disables the function generator's amplitude coupling.
1 or ON enables the function generator's amplitude coupling.

`*RST` sets this to 0.

Returns

A single <Boolean> value, 0 or 1.

Examples

`FGEN:COUPLE:AMPLITUDE ON` couples the amplitude controls of channel 1 and channel 2 together.

`FGEN:COUPLE:AMPLITUDE?` might return 0, indicating that the amplitude controls of channel 1 and channel 2 are not coupled together.

FGEN:PERiod? (Query Only)

This command returns the function generator's waveform period.

Group

Function generator

Syntax

```
FGEN:PERiod?
```

Related Commands

INSTrument:MODE

Returns

A single <NR3> value.

Examples

`FGEN:PERIOD?` might return 1.0000000000E-6, indicating that the waveform period is 1.0 µs.

*IDN? (Query Only)

This command returns identification information for the AWG. Refer to Std IEEE 488.2 for additional information.

Group IEEE mandated and optional

Syntax

```
*IDN?
```

Returns <Manufacturer>, <model>, <serial number>, <Firmware version>

<Manufacturer>:: = TEKTRONIX
<Model>:: = AWG70001A, AWG70002A
<Serial number>:: = XXXXXXX (indicates an actual serial number)
<Firmware version>:: = SCPI:99.0 FW:x.x.x.x (x.x.x.x is software version)

Examples `*IDN?` might return TEKTRONIX,AWG70001A,B010123,SCPI:99.0 FW:1.0.136.602

INSTrument:COUPle:SOURce

This command sets or returns the coupled state of the channel's Analog and Marker output controls of two channel instrument.

The set form of this command forces channel 2 to match channel 1.

After the initial coupling of the settings, changes made to either channel 1 or channel 2 amplitude settings affect both channels.

NOTE. *Output Skew and Delay settings are not coupled between channels.*

Group Instrument

Syntax

```
INSTrument:COUPle:SOURce {0|1|OFF|ON}
INSTrument:COUPle:SOURce?
```

Arguments 0 or OFF disables channel coupling.
1 or ON enables channel coupling.

`*RST` sets this to 0.

Returns A single <Boolean> value, 0 or 1.

Examples `INSTRUMENT:COUPLE:SOURCE 1` couples the CH1 parameters and CH2 parameters.

`INSTRUMENT:COUPLE:SOURCE?` might return 0.

INSTrument:MODE

This command sets or returns the AWG mode, either the AWG mode or the Function generator mode.

Group Instrument

Syntax

```
INSTrument:MODE {AWG|FGEN}
INSTrument:MODE?
```

Arguments `AWG` sets the instrument to the Arbitrary Waveform Generator mode.
`FGEN` sets the instrument to the Function generator mode.

`*RST` sets this to AWG.

Returns {AWG|FGEN}

Examples `INSTRUMENT:MODE FGEN` sets the AWG to the function generator mode.

`INSTRUMENT:MODE?` might return FGEN, indicating the AWG is in the function generator mode.

MMEMory:CATalog? (Query Only)

This command returns the current contents and state of the mass storage media.

Conditions Directories will not have their size determined. Directory's <file size> will always be 0.

Group Mass Memory

Syntax

```
MMEMory:CATalog?  [<msus>]
```

Related Commands MMEMory:CDIRectory, MMEMory:MSIS

Arguments <msus> (mass storage unit specifier) ::= <string>.

Returns <NR1>,<NR1> [,<file_entry>]

The first <NR1> indicates the total amount of storage currently used in bytes. The second <NR1> indicates the free space of the mass storage in bytes.

<file_entry> ::= "<file_name>,<file_type>,<file_size>"
<file_name> ::= the exact name of the file
<file_type> ::= is DIR for an entry that is a directory, empty/blank otherwise
<file_size> ::= <NR1> is the size of the file in bytes. For <file_type> marked DIR, the file size will always be 0.

Examples `MMEMORY:CATALOG?` might return
484672,3878652,"SAMPLE1.AWG,,2948","aaa.txt,,1024","ddd,DIR,0","zzz.awg,,2948"

MMEMory:CDIRectory

This command sets or returns the current directory of the file system on the AWG. The current directory for the programmatic interface is different from the currently selected directory in the Windows Explorer on the AWG.

Conditions The <msus> cannot be specified in the CDIR action.

Group Mass Memory

Syntax

```
MMEMory:CDIRectory [<directory_name>]
MMEMory:CDIRectory?
```

Arguments <directory_name> ::= <string>

Returns <directory_name>

Examples Assuming the current <msus> is "C:"

`MMEMORY:CDIRECTORY "\Users"` changes the current directory to C:\Users.

If the current directory is C:\Program Files

`MMEMORY:CDIRECTORY "..\Program Files"` changes the current directory to C:\Program Files

`MMEMORY:CDIRECTORY?` returns "\Program Files" if the current directory is C:\Program Files.

`MMEMORY:CDIRECTORY "\\Windows"` changes the current directory to C:\Windows.

MMEMory:DATA

This command sets or returns block data to/from a file in the current mass storage device.

NOTE. *The file path may contain a full file path. However, if the file path only contains a file name, the current directory is assumed.*

Conditions

As the IEEE 488.2 is a limitation that the largest read or write that may occur in a single command is 999,999,999 bytes as the structure is defined as a '#' followed by a byte to determine the number of bytes to read '9'. '9' indicates that we need to read 9 bytes to determine the length of the following data block: 999,999,999 (separated by commas to help separate - they will not be present normally).

Because of the size limitation, it is suggested that the user make use of the starting index (and size for querying) to append data in multiple commands/queries.

NOTE. *If querying a size that is larger than the remaining data on the file (according to the size of the file and/or the starting index) the returned size will be all of the remaining data (size will be truncated to the size of the remaining number of bytes left in the file).*

Group

Mass Memory

Syntax

```
MMEMory:DATA <file_path>[,<start_index>],<block_data>
MMEMory:DATA? <file_path>[,<start_index>[,<size>]]
```

Related Commands

MMEMory:CDIRectory, MMEMory:MSIS

Arguments

<file_path> ::= <string>
<start_index> ::= <NR1> is the byte index where writing/reading will commence in the desired <file_path>.
<size> ::= <NR1> is the size, in bytes, to read.
<block_data> ::= IEEE 488.2 data block.

Returns <block_data>

Examples `MMEMORY:DATA "123.TXT",#13ABC` loads "ABC" into 123.TXT in the current directory.

Assuming C:\123.txt already contains "ABC":
`MMEMORY:DATA "C:\123.txt",3,#223DEFGHIJKLMNOPQRSTUVWXYZ` starts loading (appends) the data at byte index 3 of C:\123.txt. The file will now contain: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Assuming C:\123.txt contains the final text in the example above:
`MMEMORY:DATA? "C:\123.txt"` Return is:
"#226ABCDEFGHIJKLMNOPQRSTUVWXYZ

Assuming C:\123.txt contains the final text in the example above:
`MMEMORY:DATA? "C:\123.txt",3,15` Return, starting at index 3 for 15 bytes is: "#215DEFGHIJKLMNOPQR"

Following these principles, you can edit or append large or small segments in existing files and alternatively read smaller or large sections in a currently existing file.

MMEMory:DATA:SIZE? (Query Only)

This command returns the size in bytes of a selected file.

Group Mass Memory

Syntax `MMEMory:DATA:SIZE? <file_path>`

Related Commands MMEMory:CDIRectory, MMEMory:MSIS

Arguments <file_path> ::= <string>

Returns <NR1> is the size, in bytes, of the selected file

Examples Assuming that the current file is in the current directory:

`MMEMORY:DATA:SIZE? "waveform1.wfm"` might return 1024.

`MMEMORY:DATA:SIZE? "C:\Tektronix\Waveforms\myFile.wfm"` might return 65535.

MMEMory:DELete (No Query Form)

This command deletes a file or directory from the AWG's hard disk. When used on a directory, this command succeeds only if the directory is empty.

Group Mass Memory

Syntax `MMEMory:DELete <file_name>[,msus]`

Related Commands MMEMory:CDIRectory, MMEMory:MSIS

Arguments <file_name> ::= <string>
<msus> (mass storage unit specifier) ::= <string>

Examples `MMEMORY:DELETE "SETUP1.AWG"` deletes SETUP1.AWG in the current directory.

`MMEMORY:DELETE "\my\proj\awg\test.awg","D:"` deletes D:\my\proj\awg\test.awg, regardless of the current directory and msus.

MMEMory:IMPort (No Query Form)

NOTE. *This command exists for backwards compatibility. Use the command MMEMory:OPEN.*

This command imports a file into the AWG's waveform list.

NOTE. *If the waveform name already exists, it is overwritten without warning. The file name must contain a path and drive letter.*

File formats supported:

ISF - TDS3000 and DPO4000 waveform format
TDS - TDS5000/TDS6000/TDS7000, DPO7000/DPO70000/DSA70000 Series waveform
TXT - Text file with analog data
TXT8 - Text file with 8 bit DAC resolution
TXT10 - Text file with 10 bit DAC resolution
TXT14 - Text file with 14 bit DAC resolution
WFM - AWG400/AWG500/AWG600/AWG700 Series waveform

PAT - AWG400/AWG500/AWG600/AWG700 Series pattern file
TFW - AFG3000 Series waveform file format
IQT - RSA3000 Series waveform file format
TIQ - RSA6000 Series waveform file format

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:IMPort <wfm_name>,<filepath>,<type>
```

Related Commands

MMEMory:OPEN

Arguments

<wfm_name> ::=<string>
<filepath> ::=<string>
<type> ::={ISF|TDS|TXT|TXT8|TXT10|TXT14|WFM|PAT|TFW|IQT|TIQ}

Examples

To import the waveform file named "MyWaveform":

```
MMEMORY:IMPORT "MyWaveform","C:\TestFiles\WFM#001.wfm",WFM
*OPC?
```

The overlapping command is followed with an Operation Complete query.

To import a TXT file:

```
MMEMORY:IMPORT "MyWaveform","C:\TestFiles\my8bit.txt",TXT8
*OPC?
```

The overlapping command is followed with an Operation Complete query.

MMEMory:IMPort:PARameter:NORMalize

***NOTE.** This command exists for backwards compatibility. Use the command MMEMory:OPEN:PARameter:NORMalize.*

This command sets or queries if the imported data is normalized during select file format import operations. The imported waveform data (for select file formats) is normalized based on the option set in this command.

File Formats supported:

.WFM - AWG400/AWG500/AWG600/AWG700 Series waveform
.AWG - AWG5000,AWG7000 Series waveforms
.TXT - Analog text files from AWG
.RFD - RFXpress AWG Series waveforms

Conditions

Normalization will not be carried out on file formats which are not supported.

Group

Mass Memory

Syntax

```
MMEMory:IMPort:PARameter:NORMalize <Type>
MMEMory:IMPort:PARameter:NORMalize?
```

Related Commands

MMEMory:OPEN:PARameter:NORMalize

Arguments

<type> ::= {NONE|FSCale|ZREFerence}

`NONE` will not normalize the imported data. The data may contain points outside of the AWG's amplitude range.
`FSCale` normalizes the imported data to the full amplitude range.
`ZREFerence` normalizes the imported data while preserving the offset.

Returns

{NONE | FSC | ZREF}

Examples

`MMEMORY:IMP:NORM NONE` imports the waveform with no normalization.

`MMEMORY:IMP:NORM?` might return ZREF, indicating that imported data is normalized while preserving the offset.

MMEMory:MDIRectory (No Query Form)

This command creates a new directory in the current path on the mass storage system.

Group

Mass Memory

Syntax

```
MMEMory:MDIRectory <directory_name>
```

Related Commands

MMEMory:CDIRectory, MMEMory:MSIS

Arguments

<directory_name> ::= <string>

Examples

`MMEMORY:MDIRECTORY "Waveform"` makes the directory "Waveform" in the current directory.

MMEMory:MSIS

This command selects or returns a mass storage device used by all MMEMory commands. <msus> specifies a drive using a drive letter. The drive letter can represent hard disk drives, network drives, external DVD/CD-RW drives, or USB memory.

Group

Mass Memory

Syntax

```
MMEMory:MSIS [<msus>]
MMEMory:MSIS?
```

Arguments

<msus> (mass storage unit specifier) ::= <string>

Returns

<msus>

NOTE. *If the mass storage device has not been defined, the returned <msus> value is the system's default drive which is typically the :C drive.*

Examples

`MMEMORY:MSIS?` might return "X:", assuming the current MSUS is the X: drive.

`MMEMORY:MSIS "D:"` changes the MSUS to the D: drive.

MMEMory:OPEN (No Query Form)

This command loads a file into the AWG waveform list.

File formats supported:

.WFMX - Native waveform format
.ISF - TDS3000 and DPO4000 waveform format
.TDS - TDS5000/TDS6000/TDS7000, DPO7000/DPO70000/DSA70000 Series waveform
.WFM - AWG400/AWG500/AWG600/AWG700/AWG5000/AWG7000 Series waveform
.PAT - AWG400/AWG500/AWG600/AWG700 Series pattern file
.TFW - AFG3000 Series waveform file format
.IQT - RSA3000 Series waveform file format
.TIQ - RFXpress series waveforms
.TIQ - RSA6000 Series waveform file format
.WFM - MDO files
.SEQX - AWG70000 sequence format
.SEQ - AWG400/AWG500/AWG600 sequence format

NOTE. *If the waveform name already exists, it will be overwritten without warning. The file name must contain a path and drive letter.*

Conditions

AWG5000/7000 setup (*.AWG), AWG 70000 setup (*.AWGX), TXT, and .MAT files will not work using this command.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:OPEN <filepath>
```

Related Commands

MMEMory:OPEN:SASSet[:WAVeform], MMEMory:OPEN:PARameter:NORMalize

Arguments

<filepath> ::= <string>

Examples

```
MMEMORY:OPEN "C:\TestFiles\WFM#001.wfm"
*OPC?
```
loads the WFM#001 waveform into the AWG waveform list. The overlapping command is followed with an Operation Complete query.

MMEMory:OPEN:PARameter:NORMalize

This command sets or queries if the imported data is normalized during select file format import operations. The imported waveform data (for select file formats) is normalized based on the option set in this command.

File formats supported:

.WFM - AWG400/AWG500/AWG600/AWG700 Series waveform
.AWG - AWG5000, AWG7000 Series waveform
.TXT - Analog text files from AWG
.RFD - RFXpress AWG Series waveforms
.MAT - Matlab files

Conditions

Normalization will not be carried out on file formats which are not supported.

Group

Mass Memory

Syntax

`MMEMory:OPEN:PARameter:NORMalize <Type>`

Arguments

<type> ::= {NONE|FSCale|ZREFerence}
`NONE` will not normalize the imported data. The data may contain points outside of the AWG's amplitude range.
`FSCale` normalizes the imported data to the full amplitude range.
`ZREFerence` normalizes the imported data while preserving the offset.

`*RST` sets the arguments to NONE.

Returns

{NONE|FSC|ZREF}

Examples

`MMEMORY:OPEN:NORM NONE` imports the waveform with no normalization.

`MMEMORY:OPEN:NORM?` might return ZREF, indicating that imported data is normalized while preserving the offset.

MMEMory:OPEN:SASSet:SEQuence (No Query Form)

This command loads all sequences, or a single sequence if <desired_sequence> is designated, into the Sequences list and all associated (used) sequences and waveforms within the designated file in <filepath>.

File formats supported:

.AWG - AWG7000 Series setup
.AWGX - AWG70000 Series setup

.SEQ - AWG400, AWG500, AWG600 Series sequence
.SEQX - AWG70000 Series sequence

NOTE. *If the sequence, any subsequent sequence, or any associated waveform name already exists, it will be overwritten without warning.*

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:OPEN:SASSet:SEQuence <filepath>[,<desired_sequence>]
```

Arguments

<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.
<desired_sequence> ::= <string>

Examples

Assuming the file AWG_w_2seqs.awgx has two sequences named Sequence1 and Sequence2 in it:

```
MMEMORY:OPEN:SASSET:SEQUENCE
"C:\TestFiles\AWG_w_2seqs.awgx","Sequence1"
*OPC?
```

imports Sequence1 alone and all waveforms used by Sequence1. The overlapping command is followed with an Operation Complete query.

Assuming the file AWG_w_2seqs.awgx has waveforms Sequence1 and Sequence2 in it:

```
MMEMORY:OPEN:SASSET:SEQUENCE "C:\TestFiles\AWG_w_2seqs.awgx"
*OPC?
```

imports both Sequence1 and Sequence2 and all waveforms used by both sequences. The overlapping command is followed with an Operation Complete query.

Assuming the file AWG_w_2seqs.awgx has two sequences named SequenceA and SequenceB in it and SequenceA uses SequenceB as a subsequence:

```
MMEM:OPEN:SASSET:SEQUENCE
"C:\TestFiles\AWG_w_2seqs.awgx","SequenceA"
*OPC?
```

imports SequenceA as a separate sequence, SequenceB as separate sequence, and all waveforms used by both sequences. The overlapping command is followed with an Operation Complete query.

MMEMory:OPEN:SASSet[:WAVeform] (No Query Form)

This command loads a single waveform if <desired_waveform> is designated. Otherwise the command imports all waveforms within the designated file in <filepath>.

File formats supported:

.AWG - AWG5000, AWG7000 Series waveforms
.AWGX - AWG70000 Series waveforms
.MAT - MATLAB files
.SEQX - AWG70000 Series sequence

NOTE. *If the waveform name already exists, it is overwritten without warning.*

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:OPEN:SASSet[:WAVeform]
<filepath>[,<desired_waveform>]
```

Related Commands

MMEMory:OPEN:PARameter:NORMalize

Arguments

<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.
<desired_waveform> ::= <string>

Examples

Assuming the test file AWG_x000_4CH.awg has waveforms Untitled36 and Untitled37 in it:

```
MMEMORY:OPEN:SASSET:WAVEFORM
"C:\TestFiles\AWG_x000_4CH.awg","Untitled36"
*OPC?
```
imports Untitled36 alone. The overlapping command is followed with an Operation Complete query.

```
MMEMORY:OPEN:SASSET:WAVEFORM "C:\TestFiles\AWG_x000_4CH.awg"
*OPC?
```
imports both Untitled36 and Untitled37. The overlapping command is followed with an Operation Complete query.

MMEMory:OPEN:SETup (No Query Form)

This command restores a setup file designated by the <filepath>.

The supported file format is the native setup format (.AWGX).

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Mass Memory

Syntax `MMEMory:OPEN:SETup <filepath>`

Arguments <filepath> ::= <string>, must contain the complete path (with drive letter) and file name.

Examples `MMEMORY:OPEN:SETUP "C:\TestFiles\mySetup.awgx"` opens the setup file named mySetup.awgx.

MMEMory:OPEN:TXT (No Query Form)

This command loads a waveform from a .TXT file into the AWG's waveform list.

NOTE. *If the waveform name already exists, it is overwritten without warning.*

Conditions Only AWG TXT compatible files can be opened using this method.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Mass Memory

Syntax `MMEMory:OPEN:TXT <filepath>,<bitdepth>`

Related Commands MMEMory:OPEN:PARameter:NORMalize

Arguments <filepath> ::= <string>, must contain the complete path (with drive letter) and file name.
<bitdepth> ::= {ANALog, DIG8, DIG9, DIG10}

Examples

```
MMEMORY:OPEN:TXT "C:\TestFiles\my8bitTXTfile.txt",DIG8
*OPC?
```

opens the digital eight bit file named my8bitTXTfile. The overlapping command is followed with an Operation Complete query.

```
MMEMORY:OPEN:TXT "C:\TestFiles\myAnalogTXTfile.txt",ANALOG
*OPC?
```

opens the analog file named myAnalogTXTfile.txt. The overlapping command is followed with an Operation Complete query.

MMEMory:SAVE:SEQuence (No Query Form)

This command exports a sequence given a unique name to an eligible storage location as the .SEQX file type.

NOTE. *If a file already exists in the selected file path, it is overwritten without warning. If the save fails, the file is deleted.*

NOTE. *The waveform name is renamed to the filename (without extension) if the waveform source is different from the selected file path.*

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:SAVE:SEQuence <sequence>,<filepath>
```

Arguments

<sequence> ::= <string>
<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.

Examples

```
MMEMORY:SAVE:SEQUENCE "mySequence","C:\mySequence.SEQX"
*OPC?
```

saves the sequence named mySequence to the filepath and names the sequence to mySequence. The overlapping command is followed with an Operation Complete query.

MMEMory:SAVE:SETup (No Query Form)

This command saves the AWG's setup and optionally includes the assets (waveforms and sequences).

NOTE. *If a file already exists in the selected file path, it is overwritten without warning. If the save fails, the file is deleted.*

This command supports the native setup file format (.AWGX).

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:SAVE:SETup <filepath>[,<with_assets>]
```

Arguments

<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.
<with_assets> ::= <Boolean>

0 indicates that the setup file be saved without waveforms and sequences.
1 indicates that the setup file will be saved with waveforms and sequences.

NOTE. *By default, if <with_assets> is not included, then the setup will be saved with assets.*

Examples

To save the setup with waveforms and sequences, use one of the two following commands:

```
MMEMORY:SAVE:SETUP "C:\mySetup.awgx"
*OPC?
```

```
MMEMORY:SAVE:SETUP "C:\mySetup.awgx",1
*OPC?
```

The overlapping commands are followed with an Operation Complete query.

To save the setup without waveforms and sequences, use the following command:

```
MMEMORY:SAVE:SETUP "C:\mySetup.awgx",0
*OPC?
```

The overlapping command is followed with an Operation Complete query.

MMEMory:SAVE[:WAVeform]:TXT (No Query Form)

This command exports a waveform given a unique waveform name to an eligible storage location from the AWG's waveforms as a text file as the .TXT file type.

NOTE. *If a file already exists in the selected file path, it is overwritten without warning. If the save fails, the file is deleted.*

NOTE. *The waveform name is renamed to the filename (without extension) if the waveform source is different from the selected file path.*

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:SAVE[:WAVeform]:TXT <wfm_name>,<filepath>,<bitdepth>
```

Arguments

<wfm_name> ::= <string>
<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.
<bitdepth> ::= {ANALog, DIG8, DIG9, DIG10}

Examples

```
MMEMORY:SAVE:WAVEFORM:TXT
"myWFM","C:\myNewTXTfile.TXT",DIGI8
*OPC?
```

saves the digital, eight bit waveform file named "myWFM" to the filepath and renames the waveform to "myNewTXTfile". The overlapping command is followed with an Operation Complete query.

MMEMory:SAVE[:WAVeform][:WFMX] (No Query Form)

This command exports a waveform given a unique waveform name to an eligible storage location from the AWG's waveforms as the .WFMX file type.

NOTE. *If a file already exists in the selected file path, it is overwritten without warning. If the save fails, the file is deleted.*

NOTE. *The waveform name is renamed to the filename (without extension) if the waveform source is different from the selected file path.*

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Mass Memory

Syntax

```
MMEMory:SAVE[:WAVeform][:WFMX] <wfm_name>,<filepath>
```

Arguments

<wfm_name> ::= <string>
<filepath> ::= <string>, must contain the complete path (with drive letter) and file name.

Examples

```
MMEMORY:SAVE:WAVEFORM:WFMX
"myWFM","C:\TestFiles\myNewWFMX.WFMX"
*OPC?
```
saves the waveform named "myWFM" to the filepath and renames the waveform to “myNewWFMX”. The overlapping command is followed with an Operation Complete query.

*OPC

This command causes the AWG to sense the internal flag referred to as the “No-Operation-Pending” flag. The command sets bit 0 in the Standard Event Status Register when pending operations are complete.

The query form returns a “1” when the last overlapping command operation is finished.

Conditions

*OPC is limited to one overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

IEEE mandated and optional

Syntax

```
*OPC
*OPC?
```

Related Commands

*WAI

Returns A single <NR1> value.

Examples `*OPC?` returns 1 to indicate that the last issued overlapping command is finished.

*OPT? (Query Only)

This command returns the implemented options for the AWG. (See page 3-1, *Status and events*.)

Group IEEE mandated and optional

Syntax

```
*OPT?
```

Returns <opt>[,<opt> [,<opt>]]]
<opt> ::= {0|xx|xx|xx} where xx is the option identifier

Examples `*OPT?` might return 0 to indicate that no option is installed in the AWG.

OUTPut:OFF

This command sets or returns the state of the All Outputs Off control.

Enabling All Output Off causes each channel's output and markers to go to an ungrounded (or open) state. Disabling the control causes each channel to go to its currently defined state. A channel's defined state can be changed while the All Outputs Off is in effect, but the actual output remains open until the All Outputs Off is disabled.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Output

Syntax

```
OUTPut:OFF {0|1|OFF|ON}
OUTPut:OFF?
```

Arguments `0` or `OFF` disables the All Output Off function, allowing the channel and marker outputs to go to their defined state.

`1` or `ON` enables the All Output Off function, disabling all channel outputs and marker outputs.

`*RST` sets all channels to 0.

Returns

A single <Boolean> value.

Examples

`OUTPUT:OFF ON`
enables All Outputs Off.

`OUTPUT:OFF?` might return 0, indicating the All Outputs Off control is not enabled and each individual channel output will function as set.

OUTPut[n][:STATe]

This command sets or returns the output state of the specified channel.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Output

Syntax

```
OUTPut[n][:STATe] {0|1|OFF|ON}
OUTPut[n][:STATe]?
```

Arguments

`0` or `OFF` disables the channel's output.
`1` or `ON` enables the channel's output.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets all channels to 0.

Returns

A single <Boolean> value.

Examples

`OUTPUT1:STATE ON`
sets the analog output state of channel 1 to on.

`OUTPUT2:STATE?` might return 0, indicating channel 2 output is off.

OUTPut[n]:SVALue[:ANALog][:STATe]

This command sets or returns the output condition of a waveform of the specified channel while the instrument is in the stopped state.

Group

Output

Syntax

```
OUTPut[n]:SVALue[:ANALog][:STATe] {OFF|ZERO}
OUTPut[n]:SVALue[:ANALog][:STATe]?
```

Related Commands

[SOURce[n]]:RMODe

Arguments

`OFF` sets the stop state output for channel "n" to open (electrically disconnected).
`ZERO` sets the stop state output for channel "n" value to 0 volts.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets all channels to ZERO.

Returns

OFF
ZERO

Examples

`OUTPUT1:SVALUE:ANALOG:STATE OFF` sets channel 1's output to be disconnected when in the stopped state.

`OUTPUT1:SVALUE:ANALOG:STATE?` might return ZERO, indicating that when channel 1 is in the stopped state, the output will be 0 volts.

OUTPut[n]:SVALue:MARKer[1|2]

This command sets or returns the condition of the specified marker of the specified channel when in the stopped state.

Group

Output

Syntax

```
OUTPut[n]:SVALue:MARKer[1|2] {OFF|LOW}
OUTPut[n]:SVALue:MARKer[1|2]?
```

Arguments

`OFF` sets the stop state marker output for channel "n" to open (electrically disconnected).
`LOW` sets the stop state marker output for channel "n" value to 0 volts.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets all channel markers to LOW.

Returns

```
OFF
LOW
```

Examples

`OUTPUT1:SVALUE:MARKER1 OFF` sets the channel 1's marker 1, to be disconnected when in the stopped state.

`OUTPUT2:SVALUE:MARKER1?` might return LOW, indicating that the channel 2's, marker 1 will be a logic level low when in the stopped state.

OUTPut[n]:WVALue[:ANALog][:STATe]

This command sets or returns the output condition of a waveform of the specified channel while the instrument is in the waiting-for-trigger state or for a brief period after the waveform loads to the DAC and before the first point plays.

Conditions

This is valid only when the Run Mode is Triggered.

When synchronization is enabled and playing, this command is not available.

Group

Output

Syntax

```
OUTPut[n]:WVALue[:ANALog][:STATe] {FIRSt|ZERO}
OUTPut[n]:WVALue[:ANALog][:STATe]?
```

Related Commands

OUTPut[n]:SVALue[:ANALog][:STATe]OUTPut[n]:SVALue[:ANALog][:STATe]

Arguments

`FIRSt` sets the output level for channel "n" to match the first point in the waveform when channel "n" is in the Waiting-for-trigger state.
`ZERO` sets the output level for channel "n" to 0 volts when channel "n" is in the Waiting-for-trigger state.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets all channels to ZERO.

Returns

```
FIRS
ZERO
```

Examples

`OUTPUT1:WVALUE:ANALOG:STATE FIRST` sets the output level for channel 1 to match the first point in the waveform when channel 1 is in the Waiting-for-trigger state.

`OUTPUT2:WVALUE:ANALOG:STATE?` might return ZERO, indicating that when channel 2 is in the Waiting-for-trigger state, the output will be 0 volts.

OUTPut[n]:WVALue:MARKer[1|2]

This command sets or returns the output condition of the specified marker of the specified channel while the instrument is in the waiting-for-trigger state or for a brief period after the waveform loads to the DAC and before the first point plays.

Conditions

This is valid only when the Run Mode is in a triggered mode.

When synchronization is enabled and playing, this command is not available.

Group

Output

Syntax

```
OUTPut[n]:WVALue:MARKer[1|2] {FIRSt|LOW|HIGH}
OUTPut[n]:WVALue:MARKer[1|2]?
```

Related Commands

OUTPut[n]:WVALue[:ANALog][:STATe]

Arguments

`FIRSt` sets the marker output level to match the first point in the waveform of channel "n" when channel "n" is in the waiting-for-trigger state.
`LOW` sets the marker output to a logic level low for channel "n" when channel "n" is in the waiting-for-trigger state.
`HIGH` sets the marker output to a logic level high for channel "n" when channel "n" is in the waiting-for-trigger state.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets all channels to LOW.

Returns

```
FIRS
LOW
HIGH
```

Examples

`OUTPUT1:WVALUE:MARKER1 FIRST` sets the channel 1 marker 1 output state to the first point of the waveform to play while in the waiting-for-trigger state.

`OUTPUT1:WVALUE:MARKER2?` might return LOW, indicating that marker 2 of channel 1 will be a logic level low while channel 1 is in the waiting-for-trigger state.

*RST (No Query Form)

This command resets the AWG to its default state. (See page C-1, *Factory initialization settings*.)

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and playing, this command is not available.

Group IEEE mandated and optional

Syntax `*RST`

Examples `*RST` resets the AWG.

SLISt:NAME? (Query Only)

This command returns the name of the sequence at the specified sequence list index.

Group Sequence

Syntax `SLISt:NAME? <sequence_list_index>`

Related Commands SLISt:SIZE?

Arguments <sequence_list_index> := <NR1>

Returns <sequence_name> := <string>

NOTE. *If there is not a sequence at the chosen index, an empty string is returned.*

Examples `SLIST:NAME? 45` might return "AnotherSequence" which is the name of the 45th sequence in the current sequence list, where SLISt:SIZE? returned a value greater than 45.

SLISt:SEQuence:DELete (No Query Form)

This command deletes a specific sequence or all sequences from the sequence list.

Group Sequence

Syntax `SLISt:SEQuence:DELete {<sequence_name>|ALL}`

Arguments <sequence_name> := {<string>|ALL}

Examples `SLIST:SEQUENCE:DELETE ALL` deletes all sequences from the current sequence list.

`SLIST:SEQUENCE:DELETE "MySequence"` deletes the sequence named MySequence.

SLISt:SEQuence:EVENt:JTIMing

This command sets or returns the condition of when the sequencer jumps upon a logic event, pattern jump, or software forced jump. The jump can occur immediately or at the end of the current sequence step.

Group

Sequence

Syntax

```
SLISt:SEQuence:EVENt:JTIMing <sequence_name>,
{END|IMMediate}
SLISt:SEQuence:EVENt:JTIMing?  <sequence_name>
```

Arguments

`END` – on receiving an event, wait until the end of current step before jumping to specified event jump step
`IMMediate` – on receiving an event, immediately jump to specified event jump step

Returns

`END`
`IMM`

Examples

`SLIST:SEQUENCE:EVENT:JTIMING "MySequence", END` requires all event jumps to wait for the end of current sequence step before jumping to the event jump step.

`SLIST:SEQUENCE:EVENT:JTIMING? "MySequence"` might return IMM, indicating that all event jumps are to be processed immediately in sequence.

SLISt:SEQuence:EVENt:PJUMp:DEFine

This command associates an event pattern with the jump-to-step for Pattern Jump. The query returns the jump step associated to the specified pattern.

The event pattern is read from the Pattern Jump In connector on the rear panel. Eight bits of data and a strobe are required. When the strobed event pattern is received, an event pattern jump is created, moving the sequence to the step defined in this command.

Conditions

The pattern jump feature for the sequence must be set to enabled. See SLISt:SEQuence:EVENt:PJUMp:ENABle.

Group

Sequence

Syntax

```
SLISt:SEQuence:EVENt:PJUMp:DEFine <sequence_name>,
<pattern>, <jump_step>
SLISt:SEQuence:EVENt:PJUMp:DEFine?  <sequence_name>,
<pattern>
```

Related Commands

SLISt:SEQuence:EVENt:PJUMp:ENABle

Arguments

<sequence_name> := <string>

<pattern>:=<NR1>. The value range is between 0 and 255. This parameter specifies the event pattern to make an event jump. The pattern bits are mapped to the integer value as follows:

	MSB							LSB
Event bits	7	6	5	4	3	2	1	0

<jump_step>::=<NR1> between 1 and 16383.

Returns

<NR1> ::= <jump_step>

Examples

`SLIST:SEQUENCE:EVENT:PJUMP:DEFINE "MySequence", 15, 3` sets the jump target index to the third sequence step of "MySequence" for the event pattern 00001111.

`SLIST:SEQUENCE:EVENT:PJUMP:DEFINE? "MySequence", 84` might return 1200, indicating that at pattern event 84, the sequence will jump to step 1200 of "MySequence".

SLISt:SEQuence:EVENt:PJUMp:ENABle

This command sets or returns the Event Pattern Jump enable for the specified sequence.

When enabled, the data at the Pattern Jump In connector can be strobed in, causing a sequence to jump to a defined step. The sequence and step are defined with the command SLISt:SEQuence:EVENt:PJUMp:DEFine.

Group

Sequence

Syntax

```
SLISt:SEQuence:EVENt:PJUMp:ENABle <sequence_name>,
{0|1|OFF|ON}
SLISt:SEQuence:EVENt:PJUMp:ENABle?  <sequence_name>
```

Related Commands SLISt:SEQuence:EVENt:PJUMp:DEFine

Arguments <sequence_name> ::= <string>

OFF or 0 disables pattern jump as an event source independent of any values present at the Pattern Jump In connector.
ON or 1 enables pattern jump as an event source.

`*RST` sets this to 0.

Returns A single <Boolean> value.

Examples `SLIST:SEQUENCE:EVENT:PJUMP:ENABLE "MySequence", ON` enables the pattern jump.

`SLIST:SEQUENCE:EVENT:PJUMP:ENABLE? "MySequence"` might return 1, indicating the pattern jump is enabled.

SLISt:SEQuence:EVENt:PJUMp:SIZE? (Query Only)

This command returns the maximum number of entries in the pattern jump table.

Group Sequence

Syntax `SLISt:SEQuence:EVENt:PJUMp:SIZE?`

Returns A single <NR1> value of 256.

Examples `SLIST:SEQUENCE:EVENT:PJUMP:SIZE?` will return 256, indicating the maximum number of entries in the pattern jump table.

SLISt:SEQuence:LENGth? (Query Only)

This command returns the total number of steps in the named sequence.

Group Sequence

Syntax `SLISt:SEQuence:LENGth?  <sequence_name>`

Arguments <sequence_name> := <string>

Returns <number_of_steps> := <NR1>

Examples `SLIST:SEQUENCE:LENGTH? "LongSequence"` might return 10000, indicating there are 10,000 steps in the sequence named “LongSequence”.

SLISt:SEQuence:NEW (No Query Form)

This command creates a new sequence with the selected name, number of steps, and number of tracks.

Group Sequence

Syntax

```
SLISt:SEQuence:NEW <sequence_name>, <number_of_steps>
[, <number_of_tracks>]
```

Arguments

<sequence_name> := <string>

<number_of_steps> := <NR1> maximum of 16383 steps and a minimum of 1
<number_of_tracks> := <NR1> maximum of 8 and minimum of 1 (defaults to number of available channels (1 or 2))

Examples

`SLIST:SEQUENCE:NEW "LongSequence", 16000, 4` creates a second sequence named LongSequence with 16000 steps and four tracks.

SLISt:SEQuence:RFLag

This command sets or returns the Enable Flag Repeat value for the sequence. If the value is ON, then the flags will change each time that the step plays out. For example if Wfm1 is at a step in Sequence with repeat 2 and one of the flags is set to Toggle, then the flag state will toggle twice at this step if the Enable Flag Repeat value is ON.

Group

Sequence

Syntax

```
SLISt:SEQuence:RFLag <sequence_name>, {0|1|OFF|ON}
SLISt:SEQuence:RFLag?  <sequence_name>
```

Arguments

<sequence_name> ::= <string>

0 or OFF disables the Flag Repeat. This is the default value.
1 or ON enables the Flag Repeat.

Returns

A single <Boolean> value.

Examples

`SLISt:SEQuence:RFLag “MyTest”, ON` enables the Repeat Flag.

`SLISt:SEQuence:RFLag?  “MyTest”` returns 0 if the Repeat Flag is not set.

SLISt:SEQuence:STEP:MAX? (Query Only)

This command returns the maximum number of steps allowed in a sequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP:MAX?
```

Returns A single <NR1> value of 16383.

Examples `SLIST:SEQUENCE:STEP:MAX?` will return 16383, indicating the maximum number of steps allowed in a sequence.

SLISt:SEQuence:STEP[n]:EJINput

This command sets or returns wether the sequence will jump when it receives Trigger A, Trigger B, Internal Trigger, or no jump at all. This is settable for every step in a sequence.

Group Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:EJINput <sequence_name>,
{ATRigger|BTRigger|OFF|ITRigger}
SLISt:SEQuence:STEP[n]:EJINput?  <sequence_name>
```

Arguments [n] is a step in the sequence with a value between 1 and 16383.

<sequence_name> := <string>

`ATRigger` – This enables the sequencer to jump to the event of a ATRIG.
`BTRigger` – This enables the sequencer to jump to the event of a BTRIG.
`ITRigger` – This enables the sequencer to jump to the event of an Internal Trigger.
`OFF` – Ignores all events, even if an event occurs during that step.

`*RST` sets this to OFF.

Returns

```
ATR
BTR
ITR
OFF
```

Examples `SLISt:SEQuence:STEP1:EJINput "MySequence", ATR` allows the sequencer to jump to step 1 after receiving a Trigger A event from Force Trig A or a signal on the Trigger A input connector.

`SLISt:SEQuence:STEP1:EJINput?  "MySequence"` might return BTR, indicating this step will only jump after receiving a Trigger B event from a Force Trig B or a signal on the Trig B input connector.

SLISt:SEQuence:STEP[n]:EJUMp

This command sets or returns the step that the specified sequence will jump to on a trigger event. This setting is only available if the event jump input has been selected as Trigger A or Trigger B for the specified step.

Conditions

The Event Input must be set at the same step with the command SLISt:SEQuence:STEP[n]:EJINput.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:EJUMp <sequence_name>,
{<NR1>|NEXT|FIRSt|LAST|END}
SLISt:SEQuence:STEP[n]:EJUMp?  <sequence_name>
```

Related Commands

SLISt:SEQuence:STEP[n]:EJINput

Arguments

[n] is a step in the sequence <NR1>
<sequence_name> := <string>

<NR1> - This enables the sequencer to jump to the specified step, where the value is between 1 and 16383.
`NEXT` – This enables the sequencer to jump to the next sequence step.
`FIRSt` – This enables the sequencer to jump to first step in the sequence.
`LAST` – This enables the sequencer to jump to the last step in the sequence.
`END` – This enables the sequencer to jump to the end and play 0 V until play is stopped.

Returns

<NR1> a single value.

```
LAST
FIRS
NEXT
END
```

Examples

`SLISt:SEQuence:STEP1:EJUMp "MySequence", 6` causes the sequencer to jump to the sixth step after executing the first step after the trigger event.

`SLISt:SEQuence:STEP1:EJUMp?  "MySequence"` might return 6, that when step 1 completes, the sequence will jump to step 6 after the trigger event.

`SLISt:SEQuence:STEP1:EJUMp "MySequence", LAST allows the sequencer to jump to last step in the sequence after executing step 1.`

`SLISt:SEQuence:STEP1:EJUMp? "MySequence" might return NEXT, indicating the sequencer will proceed to the next step after the trigger event.`

SLISt:SEQuence:STEP[n]:GOTO

This command sets or returns the target step for the GOTO command of the sequencer at the specified step.

After generating the waveform(s) specified in a sequence step, the sequencer jumps to the step specified as the GOTO step. This is an unconditional jump. If the GOTO step is not specified, the sequencer moves to the next step. If the Repeat Count is Infinite, the specified GOTO step is not used.

Group Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:GOTO <sequence_name>,
{<NR1>|LAST|FIRSt|NEXT|END}
SLISt:SEQuence:STEP[n]:GOTO? <sequence_name>
```

Related Commands SLISt:SEQuence:STEP[n]:RCOunt

Arguments

[n] is a step in the sequence.

<sequence_name> := <string>

<NR1> –This enables the sequencer to go to the specified step, where the value is between 1 and 16383.
`LAST` –This enables the sequencer to go to the last step in the sequence.
`FIRSt` –This enables the sequencer to go to first step in the sequence.
`NEXT` –This enables the sequencer to go to the next sequence step. `SLISt:SEQuence:STEP[n]:EJUMp:STEP` setting is ignored.
`END` –This enables the sequencer to go to the end and play 0 V until play is stopped.

Returns

<NR1> a single value.
`LAST`
`FIRS`
`NEXT`
`END`

Examples

`SLISt:SEQuence:STEP1:GOTO "MySequence", 6` causes the sequencer to jump to the sixth step after executing the first step.

`SLISt:SEQuence:STEP1:GOTO? "MySequence"` might return LAST, indicating that after playing this step, it will proceed to the last step of the sequence.

SLISt:SEQuence:STEP[n]:RCOunt

This command sets or returns the repeat count, which is the number of times the assigned waveform(s) play before proceeding to the next step in the sequence.

Group Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:RCOunt <sequence_name>,
{ONCE|INFinite|<NR1>}
SLISt:SEQuence:STEP[n]:RCOunt?  <sequence_name>
```

Arguments [n] is a step in the sequence

<sequence_name> := <string>

`ONCE` – Plays the waveform one time during this sequence step.
`INFinte` – Plays the waveform continuously during this sequence step.
<NR1> - Plays this waveform the selected number of times during this sequence step. The allowed value is between 1 and 2^20.

Returns ONCE
INF
<NR1> a single value.

Examples `SLIST:SEQUENCE:STEP1:RCOUNT "MySequence", 55` sets the repeat count to 55 for step 1.

`SLIST:SEQUENCE:STEP1:RCOUNT? "MySequence"` might return ONCE, indicating that a waveform(s) in the track(s) will only play once before continuing to the next specified step.

`SLIST:SEQUENCE:STEP12:RCOUNT "MySequence", INFINTE` sets the repeat count to Infinite on step 12, indicating that a waveform(s) in track(s) will play until stopped externally by the AWGControl:STOP command or the SLISt:SEQuence:JUMP:IMMediate command.

SLISt:SEQuence:STEP:RCOunt:MAX? (Query Only)

This command returns the maximum number of repeats allowed for a step in a sequence.

Group

Sequence

Syntax

`SLISt:SEQuence:STEP:RCOunt:MAX?`

Related Commands

SLISt:SEQuence:STEP[n]:RCOunt

Returns

A single <NR1> value of 1048576.

Examples

`SLIST:SEQUENCE:STEP:RCOUNT:MAX?` will return 10478576, indicating the maximum number of repeats of a step in a sequence.

SLISt:SEQuence:STEP[n]:TASSet[m]? (Query Only)

This command returns the name of the waveform or subsequence at the specified sequence's step number and track asset value.

Waveform or subsequence can be distinguished by the SLISt:SEQuence:STEP[n]:TASSet[m]:TYPE? query.

Group

Sequence

Syntax

`SLISt:SEQuence:STEP[n]:TASSet[m]?  <sequence_name>`

Related Commands

SLISt:SEQuence:STEP[n]:TASSet[m]:TYPE?

Arguments

<asset_name> ::= <string>

[n] is a step in the sequence <NR1>. [n] is a value between 1 and 16383, not to exceed the number of steps for sequence [m].
[m] is a specific track in a sequence <NR1>. [m] is a value between 1 and 8, not to exceed the number of tracks in the sequence.

Returns

<asset_name> ::= <string>
An empty string is returned if no waveform has been assigned to this track and step.

Examples

`SLISt:SEQuence:STEP5:TASSet2? “MyTest”` might return “Sin360” which is the waveform assigned to the fifth step of track 2 of the sequence named “MyTest”.

`SLISt:SEQuence:STEP5:TASSet? “MyTest”` might return “Seq1” which is a subsequence set at the fifth step of all tracks of the sequence named “MyTest”.

SLISt:SEQuence:STEP[n]:TASSet:SEQuence (No Query Form)

This command assigns a subsequence for a specific sequence's step and track.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TASSet:SEQuence <sequence_name>,
<subsequence_name>
```

Arguments

<sequence_name> ::= <string>
<subsequence_name> ::= <string>

[n] is a step in the sequence <NR1> [n] is a value between 1 and 16383, not to exceed the number of steps for this sequence.

Examples

`SLISt:SEQuence:STEP5:TASSet:SEQuence “MyTest”,“Seq360”` sets the subsequence “Seq360” to the fifth step of all tracks in the sequence named “MyTest”.

SLISt:SEQuence:STEP[n]:TASSet[m]:TYPE? (Query Only)

This command returns the type of asset assigned at the step and track for a specified sequence. The types of assets are waveform and subsequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TASSet[m]:TYPE? <sequence_name>
```

Arguments

<sequence_name> ::= <string>

[n] is a step in the sequence <NR1> [n] is a value between 1 and 16383, not to exceed the number of steps for sequence [m].
[m] is a specific track in a sequence <NR1> [m] is a value between 1 and 8, not to exceed the number of tracks in the sequence.

Returns { WAVeform | SEQuence}

WAVEform – signifies a waveform loaded at the step and track for this sequence SEQuence – signifies a subsequence loaded at the step and track for this sequence.

Examples `SLISt:SEQuence:STEP5:TASSet2:TYPE? “MyTest”` might return WAV because “Sin360” was the waveform set at the fifth step of Track 2 to the sequence named “MyTest”.

`SLISt:SEQuence:STEP10:TASSet1:TYPE? “MyTest”` might return SEQ because “Seq6” was the waveform set at the tenth step of Track 1 to the sequence named “MyTest”.

SLISt:SEQuence:STEP[n]:TASSet[m]:WAVeform (No Query Form)

This command assigns a waveform for a specific sequence's step and track. This waveform is played whenever the playing sequence reaches this step. A track in a sequence is assigned to a channel with the command [SOURce[n]]:CASSET:SEQ.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TASSet[m]:WAVeform <sequence_name>,
<waveform_name>
```

Related Commands

[SOURce[n]]:CASSet:SEQuence

Arguments

<sequence_name> ::= <string>
<waveform_name> ::= <string>

[n] is a step in the sequence <NR1> [n] is a value between 1 and 16383, not to exceed the number of steps for sequence [m].
[m] is a specific track in a sequence <NR1> [m] is a value between 1 and 8, not to exceed the number of tracks in the sequence.

Examples

`SLIST:SEQUENCE:STEP5:TASSET2:WAVEFORM “MyTest”,“Sine360”`
assigns the waveform “Sine360” to the step 5 of track 2 of the sequence named “MyTest”.

SLISt:SEQuence:STEP[n]:TFLag[m]:AFLag

This command sets or returns the Flag A value of the track in a sequence step.

Conditions

Flags are not allowed in sequence steps containing a subsequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TFLag[m]:AFLag <sequence_name>,
{NCHange|HIGH|LOW|TOGGle|PULSe}
SLISt:SEQuence:STEP[n]:TFLag[m]:AFLag?  <sequence_name>
```

Arguments

[n] is a step in the sequence <NR1>.
[m] is a specific track in a sequence <NR1>.

<sequence_name> ::= <string>

`NCHange` – The flag state continues to be in the state is defined in the previous step Default value.
`HIGH` – The flag signal transitions to the high state.
`LOW` – The flag signal transitions to the low state.
`TOGGle` – The flag signal transitions to the high state if the previous step defined the flag to be in the low state and vice versa.
`PULSe` – The flag signal outputs a pulse signal of a fixed width.

Returns

NCH
HIGH
LOW
TOGG
PULS

Examples

`SLISt:SEQuence:STEP5:TFLAG1:AFLag “MyTest”,HIGH` sets the Flag output of Flag A to high when the instrument is playing out the fifth step of the first track of sequence "MyTest".

`SLISt:SEQuence:STEP2:TFLAG3:AFLag?  “MyTest”` might return "LOW" when Flag A of sequence "MyTest" is set to "LOW" in the second step of track 3.

SLISt:SEQuence:STEP[n]:TFLag[m]:BFLag

This command sets or returns the Flag B value of the track in a sequence step.

Conditions

Flags are not allowed in sequence steps containing a subsequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TFLag[m]:BFLag <sequence_name>,
{NCHange|HIGH|LOW|TOGGle|PULSe}
SLISt:SEQuence:STEP[n]:TFLag[m]:BFLag?  <sequence_name>
```

Arguments

[n] is a step in the sequence <NR1>.
[m] is a specific track in a sequence <NR1>.

<sequence_name> ::= <string>

`NCHange` – The flag state continues to be in the state is defined in the previous step Default value.
`HIGH` – The flag signal transitions to the high state.
`LOW` – The flag signal transitions to the low state.
`TOGGle` – The flag signal transitions to the high state if the previous step defined the flag to be in the low state and vice versa.
`PULSe` – The flag signal outputs a pulse signal of a fixed width.

Returns

NCH
HIGH
LOW
TOGG
PULS

Examples

`SLISt:SEQuence:STEP5:TFLAG1:BFLag “MyTest”,HIGH` sets the Flag output of Flag B to high when the instrument is playing out the fifth step of the first track of sequence "MyTest".

`SLISt:SEQuence:STEP2:TFLAG3:BFLag?  “MyTest”` might return "LOW" when Flag B of sequence "MyTest" is set to "LOW" in the second step of track 3.

SLISt:SEQuence:STEP[n]:TFLag[m]:CFLag

This command sets or returns the Flag C value of the track in a sequence step.

Conditions

Flags are not allowed in sequence steps containing a subsequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TFLag[m]:CFLag <sequence_name>,
{NCHange|HIGH|LOW|TOGGle|PULSe}
SLISt:SEQuence:STEP[n]:TFLag[m]:CFLag?  <sequence_name>
```

Arguments

[n] is a step in the sequence <NR1>.
[m] is a specific track in a sequence <NR1>.

<sequence_name> ::= <string>

`NCHange` – The flag state continues to be in the state is defined in the previous step Default value.
`HIGH` – The flag signal transitions to the high state.
`LOW` – The flag signal transitions to the low state.
`TOGGle` – The flag signal transitions to the high state if the previous step defined the flag to be in the low state and vice versa.
`PULSe` – The flag signal outputs a pulse signal of a fixed width.

Returns

NCH
HIGH
LOW
TOGG
PULS

Examples

`SLISt:SEQuence:STEP5:TFLAG1:CFLag “MyTest”,HIGH` sets the Flag output of Flag C to high when the instrument is playing out the fifth step of the first track of sequence "MyTest".

`SLISt:SEQuence:STEP2:TFLAG3:CFLag?  “MyTest”` might return "LOW" when Flag C of sequence "MyTest" is set to "LOW" in the second step of track 3.

SLISt:SEQuence:STEP[n]:TFLag[m]:DFLag

This command sets or returns the Flag D value of the track in a sequence step.

Conditions

Flags are not allowed in sequence steps containing a subsequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:TFLag[m]:DFLag <sequence_name>,
{NCHange|HIGH|LOW|TOGGle|PULSe}
SLISt:SEQuence:STEP[n]:TFLag[m]:DFLag?  <sequence_name>
```

Arguments

[n] is a step in the sequence <NR1>.
[m] is a specific track in a sequence <NR1>.

<sequence_name> ::= <string>
`NCHange` – The flag state continues to be in the state is defined in the previous step Default value.
`HIGH` – The flag signal transitions to the high state.
`LOW` – The flag signal transitions to the low state.
`TOGGle` – The flag signal transitions to the high state if the previous step defined the flag to be in the low state and vice versa.
`PULSe` – The flag signal outputs a pulse signal of a fixed width.

Returns

NCH
HIGH
LOW
TOGG
PULS

Examples

`SLISt:SEQuence:STEP5:TFLAG1:DFLag “MyTest”,HIGH` sets the Flag output of Flag D to high when the instrument is playing out the fifth step of the first track of sequence "MyTest".

`SLISt:SEQuence:STEP2:TFLAG3:DFLag?  “MyTest”` might return "LOW" when Flag D of sequence "MyTest" is set to "LOW" in the second step of track 3.

SLISt:SEQuence:STEP[n]:WINPut

This command sets or returns the trigger source for the wait input state for a step.

Send a trigger signal in one of the following ways:

- Use an external trigger signal.
- Push the Force Trigger button on the front panel.
- Send the *TRG or TRIGger[:SEQuence][:IMMediate] remote commands.
- Use the Internal Trigger.

Group

Sequence

Syntax

```
SLISt:SEQuence:STEP[n]:WINPut <sequence_name>,
{ATRigger|BTRigger|ITRigger|OFF}
SLISt:SEQuence:STEP[n]:WINPut?  <sequence_name>
```

Related Commands

TRIGger[:SEQuence][:IMMediate]

*TRG

Arguments

[n] is a step in the sequence <NR1>; [n] is a value between 1 and 16383.

<sequence_name> ::= <string>

`ATRigger` – This enables the sequencer to move due to a trigger event from the A Trigger Input connector or the A Force Trigger front panel button.
`BTRigger` – This enables the sequencer to move due to a trigger event from the B Trigger Input connector or the B Force Trigger front panel button.
`ITRigger` – This enables the sequencer to move due to an Internal Trigger event.
`OFF` – Disables the wait for trigger event, allowing the waveforms(s) of this step to be played immediately.

Returns

ATR
BTR
ITR
OFF

Examples

`SLIST:SEQUENCE:STEP1:WINPUT "MYSEQUENCE", ATR` allows the sequencer play the waveform(s) specified in this step after receiving a trigger event from the Trigger A Input connector or a Force A Trigger.

`SLIST:SEQUENCE:STEP1:WINPUT? "MySequence"` might return BTR, indicating this step will only play the waveform(s) specified after receiving a trigger event from the Trigger B Input connector or a Force B Trigger.

SLISt:SEQuence:TRACk? (Query Only)

This command returns the number of tracks defined in the specified sequence.

Group

Sequence

Syntax

```
SLISt:SEQuence:TRACk?  <sequence_name>
```

Related Commands

SLISt:SEQuence:NEW

Arguments

<sequence_name> ::= <string>

Returns

A single <NR1> value.

Examples `SLIST:SEQUENCE:TRACK? "MySequence"` might return 4, indicating the number of tracks defined in this sequence.

SLISt:SEQuence:TRACk:MAX? (Query Only)

This command returns the maximum number of tracks allowed in a sequence

Group

Sequence

Syntax

`SLISt:SEQuence:TRACk:MAX?`

Returns

A single <NR1> value of 8.

Examples

`SLIST:SEQUENCE:TRACK:MAX?` will return 8, indicating the maximum number of tracks allowed in a sequence.

SLISt:SEQuence:TSTamp? (Query Only)

This command returns the timestamp of the named sequence. Every sequence has a timestamp that indicates when the sequence was created or last modified.

Group

Sequence

Syntax

`SLISt:SEQuence:TSTamp?  <sequence_name>`

Arguments

<sequence_name> ::= <string>

Returns

String with "yyyy/mm/dd hh:mm:ss" as the sequence timestamp.

Where:

yyyy refers to a four-digit year number.
mm refers to two-digit month number from 01 to 12.
dd refers to two-digit day number in the month.
hh refers to two-digit hour number.
mm refers to two-digit minute number.
ss refers to two-digit second number.

Examples

`SLIST:SEQUENCE:TSTAMP? "MySequence"` might return "2012/07/25 9:05:21" which is the date and time the sequence named "MySequence" was created or last modified.

SLISt:SIZE? (Query Only)

This command returns the number sequences in sequence list.

Group Sequence

Syntax `SLISt:SIZE?`

Returns A single <NR1> value.

Examples `SLIST:SIZE?` might return 4500, which is the number of existing sequences in the sequence list.

[SOURce]:FREQuency[:CW]|[:FIXed]

NOTE. *This command exists for backwards compatibility. Use the command CLOCk:SRATe.*

This command sets or returns the clock sample rate of the AWG.

[:CW] and [:FIXed] are optional to provide legacy support but provide no added functionality.

Conditions This command is not valid when CLOCk:SOURce is set to EXTernal.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Source

Syntax

```
[SOURce]:FREQuency[:CW]|[:FIXed] <NR3>
[SOURce]:FREQuency[:CW]|[:FIXed]?
```

Related Commands CLOCk:SOURce

Arguments

A single <NR3> value.

Range:	AWG70001A	1.49 kS/s to 50 GS.
	AWG70002A	1.49 kS/s to 25 GS.

`*RST` sets the frequency to 25 GHz.

Returns

A single <NR3> value.

Examples

```
SOURCE:FREQUENCY 10E6
*OPC?
```
sets the clock sample rate to 10 MS/s. The overlapping command is followed with an Operation Complete query.

`SOURCE:FREQUENCY?` might return 8.0000000000+E9, indicating that the clock sample rate is set to 8 GS/s.

[SOURce]:RCCouple

This command sets or returns the coupled state of the channel's run controls of a two channel instrument. The Run controls consist of the Run Mode and Trigger Input.

The set form of the command forces channel 2 to match channel 1.

After the initial coupling of the settings, changes made to either channel 1 or channel 2 run controls affect both channels.

Group

Source

Syntax

```
[SOURce]:RCCouple {0|1|ON|OFF}
```

Arguments

0 or OFF
1 or ON

`*RST` sets this to 0.

Returns

A single <Boolean> value.

Examples

`SOURCE:RCCOUPLE 1` sets the Run Control Coupled state to On.

`SOURCE:RCCOUPLE?` might return 0, indicating that the Run Control Coupled state is Off.

[SOURce]:ROSCillator:MULTiplier

***NOTE.** This command exists for backwards compatibility. Use the command CLOCk:EREFerence:MULTiplier.*

This command sets or returns the multiplier of the external reference signal when the external reference is variable.

Conditions

Setting the external reference multiplier rate forces the external reference divider rate to a value of 1.

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Source

Syntax

```
[SOURce]:ROSCillator:MULTiplier <NR1>
[SOURce]:ROSCillator:MULTiplier?
```

Related Commands

CLOCk:EREFerence:MULTiplier

Arguments

A single <NR1> value.
Range: 1 to 1000 (limited by the maximum sample rate).

Returns

A single <NR1> value.

Examples

```
SOURCE:ROSCILLATOR:MULTIPLIER 50
*OPC?
```
sets the multiplier to 50. The overlapping command is followed with an Operation Complete query.

`SOURCE:ROSCILLATOR:MULTIPLIER?` might return 100.

[SOURce[n]]:CASSet? (Query Only)

This command returns the asset (waveform or sequence) assigned to the specified channel.

Group Source

Syntax `[SOURce[n]]:CASSet?`

Arguments [n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Returns If a waveform is assigned to the channel, a single <string> value representing a waveform name.
If a sequence is assigned to the channel, a single <string> value representing a sequence name and the track number separated by a comma ("sequence_name,track_number").

Examples `SOURCE1:CASSET?` might return "SEQ100,1" if track 1 of SEQ100 is assigned to channel 1.

`SOURCE1:CASSET?` might return "SINE100" if waveform "SINE100" is assigned to channel 1.

[SOURce[n]]:CASSet:SEQuence (No Query Form)

This command assigns a track of a sequence (from the sequence list) to the specified channel.

Conditions When synchronization is enabled and playing, this command is not available.

Group Source

Syntax `[SOURce[n]]:CASSet:SEQuence <sequence_name>, <track_number>`

Arguments <sequence_name> ::= <string>
<track_number> ::= <NR1>
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Examples `SOURCE1:CASSET:SEQUENCE "Sequence10", 1` assigns track 1 of "Sequence10" to Channel 1.

[SOURce[n]]:CASSet:TYPE? (Query Only)

This command returns the type of the asset (waveform or sequence) assigned to a channel.

Group Source

Syntax `[SOURce[n]]:CASSet:TYPE?`

Arguments [n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Returns WAV – a waveform is assigned to the specified channel.
SEQ – a sequence is assigned to the specified channel.
NONE – nothing is assigned to the specified channel.

Examples `SOURCE1:CASSET:TYPE?` might return WAV, indicating that a waveform is assigned to Channel 1.

[SOURce[n]]:CASSet:WAVeform (No Query Form)

This command assigns a waveform (from the waveform list) to the specified channel.

Conditions When synchronization is enabled and playing, this command is not available.

Group Source

Syntax `[SOURce[n]]:CASSet:WAVeform <wfm_name>`

Arguments <wfm_name>::=<string>
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Examples `SOURCE1:CASSET:WAVEFORM "SINE100"` assigns waveform "SINE100" to Channel 1.

[SOURce[n]]:DAC:RESolution

This command sets or returns the DAC resolution.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is enabled and playing, this command is not available.

Group

Source

Syntax

```
[SOURce[n]]:DAC:RESolution {8|9|10}
[SOURce[n]]:DAC:RESolution?
```

Arguments

8 indicates 8 bit DAC Resolution + 2 marker bits.
9 indicates 9 bit DAC Resolution + 1 marker bit.
10 indicates 10 bit DAC Resolution + 0 marker bits.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 10.

Returns

A single <NR1> value: 8, 9, or 10.

Examples

```
SOURCE1:DAC:RESOLUTION 10
*OPC?
```
sets the channel 1 DAC resolution to 10 bits + 0 marker bits. The overlapping command is followed with an Operation Complete query.

`SOURCE1:DAC:RESOLUTION?` might return 8 indicating 8 bit DAC resolution + 2 marker bits.

[SOURce[n]]:JUMP:FORCe (No Query Form)

This command forces the sequencer to jump to the specified step per channel. A force jump does not require a trigger event to execute the jump.

For two channel instruments, if both channels are playing the same sequence, then both channels jump simultaneously to the same sequence step.

Group

Source

Syntax

```
[SOURce[n]]:JUMP:FORCe {FIRSt|CURRent|LAST|END|<NR1>}
```

Arguments

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)
FIRSt - This enables the sequencer to jump to first step in the sequence.
CURRent - This enables the sequencer to jump to the current sequence step, essentially starting the current step over.
LAST - This enables the sequencer to jump to the last step in the sequence.
END - This enables the sequencer to go to the end and play 0 V until play is stopped.
<NR1> - This enables the sequencer to jump to the specified step, where the value is between 1 and 16383.

Examples

`SOURCE1:JUMP:FORCE 240` specifies that Channel 1 will jump to step 240 at end of sequence step or immediately, depending on the state of SLISt:SEQuence:EVENt:JTIMing.
If Channel 1 and Channel 2 are playing the same sequence, Channel 2 also jumps to location 240 simultaneously.

`SOURCE2:JUMP:FORCE CURRENT` starts playing the current step on Channel 2 based on the SLISt:SEQuence:EVENt:JTIMing value.

[SOURce[n]]:JUMP:PATTern:FORCe (No Query Form)

This command generates an event, forcing the sequencer to the step specified by the pattern in the pattern jump table. If the sequence is playing on both channels, the force jump is applied to both channels simultaneously.

Group

Source

Syntax

```
[SOURce[n]]:JUMP:PATTern:FORCe <pattern>
```

Arguments

<pattern>:=<NR1>. The values ranges between 0 and 255. This parameter specifies the event pattern to make an event jump. The pattern bits are mapped to the integer value as follows:

	MSB	LSB
Event bits	7 6 5 4 3 2 1 0	

Examples

`SOURCE2:JUMP:PATTERN:FORCE 15` jumps to the location chosen in the definition of event pattern 00001111 for channel 2.

If `SLIST:SEQUENCE:EVENT:PJUMP:DEFINE "MySequence", 255, 4` and "MySequence" is playing, then `SOURCE1:JUMP:PATTern:FORCe 255` the sequence will jump to step 4 for channel 1.

[SOURce[n]]:MARKer[1|2]:DELay

This command sets or returns the marker delay. Marker delay is independent for each channel.

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Source

Syntax

```
[SOURce[n]]:MARKer[1|2]:DELay <NR3>
[SOURce[n]]:MARKer[1|2]:DELay?
```

Related Commands

[SOURce[n]]:DAC:RESolution

Arguments

A single <NRf> value.

Range: 0 to 100E-12 seconds.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1).

`*RST` sets this to 0.

Returns

A single <NRf> value.

Examples

`SOURCE1:MARKER1:DELAY 20PS`
sets the marker1 delay of channel 1 to 20 picoseconds.

`SOURCE1:MARKER1:DELAY?` might return 10.0000000000E-12 indicating a delay of 10 ps.

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]

This command sets or returns the marker voltage amplitude of the selected marker of the selected channel.

NOTE. *The following commands may overwrite the values set with this command:*

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Source

Syntax

```
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:
AMPLitude] <NRf>
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:
AMPLitude]?
```

Related Commands

[SOURce[n]]:DAC:RESolution,
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH,
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW,
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet

Arguments

A single <NRf> value.

Range: 0.5 V to 1.4 V.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 1 V.

Returns

A single <NRf> value.

Examples

```
SOURCE1:MARKER1:VOLTAGE:AMPLITUDE 0.5V
*OPC?
```

sets the channel 1, marker 1, amplitude to 0.5 volts. The overlapping command is followed with an Operation Complete query.

`SOURCE1:MARKER1:VOLTAGE:AMPLITUDE?` might return 500.0000000000E-3 volts.

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH

This command sets or returns the marker high voltage level of the selected marker of the selected channel.

NOTE. *The following command may overwrite the values set with this command:*

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Source

Syntax

```
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH
<NRf>
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH?
```

Related Commands

[SOURce[n]]:DAC:RESolution, [SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW

Arguments

A single <NRf> value.

Range: –1.4 V to 1.4 V.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 1 V.

Returns

A single <NRf> value.

Examples

`SOURCE1:MARKER1:VOLTAGE:HIGH 0.75`
sets the channel 1, marker 1, high level to 750 mV.

`SOURCE1:MARKER1:VOLTAGE:HIGH?` might return 500.0000000000E-3, indicating the channel 1, marker 1, high level is set to 500 mV.

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW

This command sets or returns the marker low voltage level of the selected marker of the selected channel.

NOTE. *The following command may overwrite the values set with this command:*

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]

Conditions

This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group

Source

Syntax

```
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW
<NRf>
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW?
```

Related Commands

[SOURce[n]]:DAC:RESolution, [SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH

Arguments

A single <NRf> value.

Range: –1.4 V to 1.4 V.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 0 V.

Returns

A single <NRf> value.

Examples

`SOURCE1:MARKER1:VOLTAGE:LOW 0.5`
sets the channel 1, marker 1, low level to 500 mV.

`SOURCE1:MARKER1:VOLTAGE:LOW?` might return 500.0000000000E-3, indicating that the channel 1, marker 1, low level is set to 500 mV.

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet

This command sets or returns the offset voltage of the selected marker of the selected channel.

NOTE. *The following command may overwrite the values set with this command:*

[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]

Group

Source

Syntax

```
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet
<NR3>
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet?
```

Related Commands

[SOURce[n]]:DAC:RESolution,
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH,
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW

Arguments

A single <NR3> value.

Range: –1.15 V to 1.15 V.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 500 mV.

Returns

A single <NR3> value.

Examples

`SOURCE1:MARKER1:VOLTAGE:OFFSET 0.5` sets the channel 1, marker 1, offset to 500 mV.

`SOURCE1:MARKER1:VOLTAGE:OFFSET?` might return 500.0000000000E-3, indicating that the channel 1, marker 1, offset is set to 500 mV.

[SOURce[n]]:RMODe

This command sets or returns the run mode of the specified channel.

Group

Source

Syntax

```
[SOURce[n]]:RMODe {CONTinuous|TRIGgered|TCONtinuous}
[SOURce[n]]:RMODe?
```

Related Commands

[SOURce[n]]:TINPut,
*TRG

Arguments

`CONTinuous` sets the Run Mode to Continuous (not waiting for trigger).
`TRIGgered` sets the Run Mode to Triggered, waiting for a trigger event. One waveform play out cycle completes, then play out stops, waiting for the next trigger event.
`TCONtinuous` sets the Run Mode to Triggered Continuous, waiting for a trigger. Once a trigger is received, the waveform plays out continuously.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to CONT.

Returns

CONT
TRIG
TCON

Examples

`SOURCE1:RMODE TRIG` sets the AWG Run mode for channel 1 to wait for a trigger.

`SOURCE1:RMODE?` might return CONT, indicating that the Run mode for channel 1 is set to continuous.

[SOURce[n]]:SCSTep? (Query Only)

This command allows you to read the current step of the sequence while the system is running.

Conditions

The return value is between 0 and 16383 or END. A 0 indicates that the sequence is not playing or is waiting for a trigger.

Group

Source

Syntax

```
[SOURce[n]]:SCSTep?
```

Arguments

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Returns

<string>
END indicates the sequence has reached the end of the sequence and the outputs are defined by the Output Options for Sequence End.

Examples

`:SCST?` might return 4, indicating that channel 1 is currently at step 4 of the sequencer.

`SOURCE2:SCSTEP?` might return 12, indicating that channel 2 is currently at step 12 of the sequencer.

`SOURCE2:SCSTEP?` might return Sequence_1,2, indicating that channel 2 is currently at step 2 of the subsequence named Sequence_1.

`SOURCE1:SCSTEP?` might return END, indicating that channel 1 is playing 0 V until the play ends.

`SOURCE1:SCSTEP?` might return <Subsequence_Name>,<Step_Index> when playing out step <Step_Index> of subsequence <Sequence_Name>.

[SOURce[n]]:SKEW

This command sets or returns the skew for the waveform associated with a channel.

Group

Source

Syntax

```
[SOURce[n]]:SKEW <NR3>
[SOURce[n]]:SKEW?
```

Arguments

A single <NR3> value.

Range: -100 ps to +100 ps. Minimum increments is 0.5 ps.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 0.

Returns

A single <NR3> value.

Examples

`SOURCE2:SKEW 75PS` sets the skew for channel 2 to 75 ps.

`SOURCE2:SKEW?` might return 75.0000000000E-12, indicating that the skew for channel 2 is set to 75 ps.

[SOURce[n]]:TINPut

This command sets or returns the trigger input source of the specified channel.

Group

Source

Syntax

```
[SOURce[n]]:TINPut {ATRigger|BTRigger}
[SOURce[n]]:TINPut?
```

Arguments

`ATRigger` selects trigger input A.
`BTRigger` selects trigger input B.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to ATR.

Returns

ATR
BTR

Examples

`SOURce1:TINPut BTRIGGER` selects Trigger B as the external trigger input source for channel 1.

`SOURce1:TINPut?` might return BTR, indicating that Trigger B is the external trigger input source for channel 1.

[SOURce[n]]:VOLTage[:LEVel][:IMMediate][:AMPLitude]

This command sets or returns the amplitude for the waveform associated with a channel.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Source

Syntax

```
[SOURce[n]]:VOLTage[:LEVel][:IMMediate][:AMPLitude] <NRf>
[SOURce[n]]:VOLTage[:LEVel][:IMMediate][:AMPLitude]?
```

Arguments A single <NRf> value.

Range: 250 mV to 500 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 500 mV.

Returns A single <NRf> value.

Examples `SOURCE1:VOLTAGE:AMPLITUDE 0.25`
sets the output amplitude of channel 1 to 250 mV_{pp}.

`SOURCE1:VOLTAGE:AMPLITUDE?` might return 350.0000000000E-3, indicating that the amplitude output for channel 1 is set to 350 mV_{pp}.

[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH

This command sets or returns the high voltage level for the waveform associated with a channel.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Source

Syntax

```
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH <NRf>
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH?
```

Related Commands [SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW

Arguments A single <NRf> value.

Range: 125 mV to 250 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to 250 mV.

Returns A single <NRf> value.

Examples `SOURCE1:VOLTAGE:LEVEL:IMMEDIATE:HIGH 0.125V`
sets the amplitude high of channel 1 to 125 mV.

`SOURCE2:VOLTAGE:LEVEL:IMMEDIATE:HIGH?` might return 250.0000000000E-3 indicating that the high voltage output voltage level for channel 2 is 250 mV.

[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW

This command sets or returns the low voltage level for the waveform associated with a channel.

Conditions This is a blocking command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Source

Syntax

```
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW <NRf>
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW?
```

Related Commands [SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH

Arguments A single <NRf> value.

Range: -250 mV to -125 mV.
[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

`*RST` sets this to -250 mV.

Returns A single <NRf> value.

Examples

`SOURCE1:VOLTage:LEVEL:IMMEDIATE:LOW -0.125V`
sets the amplitude low of Channel 1 to -0.125 mV.

`SOURCE1:VOLTage:LEVEL:IMMEDIATE:LOW?` might return -250.0000000000E-3, indicating that the low voltage output voltage level for channel 1 is -250 mV.

[SOURce[n]]:WAVeform

This command sets or returns the name of the waveform assigned to the channel.

Conditions

When synchronization is enabled and playing, this command is not available.

Group

Source

Syntax

```
[SOURce[n]]:WAVeform <wfm_name>
[SOURce[n]]:WAVeform?
```

Arguments

<wfm_name> ::= <string>

[n] ::= 1|2 ("n" determines the channel number. If omitted, interpreted as 1.)

Returns

A single <string> value representing a waveform name.

Examples

`SOURCE1:WAVEFORM "SINE100"` assigns waveform "Sine100" to channel 1.

`SOURCE1:WAVEFORM?` might return "Sine100".

*SRE

This command sets or queries the bits in the Service Request Enable register. (See page 3-1, *Status and events*.)

Group IEEE mandated and optional

Syntax

```
*SRE <NR1>
*SRE?
```

Related Commands *CLS, *ESE, *ESR?, *STB?

Arguments A single <NR1> value.

Returns A single <NR1> value.

Examples `*SRE 48` sets the bits in the SRER to the binary value 00110000.

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

STATus:OPERation:CONDition? (Query Only)

This command returns the contents of the Operation Condition Register (OCR).

Group Status

Syntax

```
STATus:OPERation:CONDition?
```

Returns A single <NR1> value showing the contents of the OCR.

Examples `STATUS:OPERATION:CONDITION?` might return 0, showing that the bits in the OCR have the binary value 0000000000000000.

STATus:OPERation:ENABle

This command sets or returns the mask for the Operation Enable Register.

Conditions

The most-significant bit cannot be set true.

Group

Status

Syntax

```
STATus:OPERation:ENABle <NR1>
STATus:OPERation:ENABle?
```

Arguments

A single <NR1> value.

Returns

A single <NR1> value.

Examples

`STATUS:OPERATION:ENABLE 1` enables the Calibrating bit.

`STATUS:OPERATION:ENABLE?` might return 1, showing that the bits in the OENR have the binary value 00000000 00000001, which means that the Calibrating bit is valid.

STATus:OPERation[:EVENt]? (Query Only)

This command returns the contents of the Operation Event Register (OEVR). Reading the OEVR clears it.

Group

Status

Syntax

```
STATus:OPERation[:EVENt]?
```

Returns

A single <NR1> value showing the contents of the OEVR.

Examples

`STATUS:OPERATION:EVENT?` might return 1, showing that the bits in the OEVR have the binary value 00000000 00000001, which means that the CALibrating bit is set.

STATus:OPERation:NTRansition

This command sets or returns the negative transition filter value of the Operation Transition Register (OTR).

Conditions The most-significant bit cannot be set true.

Group Status

Syntax
```
STATus:OPERation:NTRansition <bit_value>
STATus:OPERation:NTRansition?
```

Arguments <bit_value> ::= <NR1> is the negative transition filter value.

Returns A single <NR1> value showing the contents of the OTR.

Examples `STATUS:OPERATION:NTRANSITION 17` sets the negative transition filter value to 17.

`STATUS:OPERATION:NTRANSITION?` might return 17.

STATus:OPERation:PTRansition

This command sets or returns the positive transition filter value of the Operation Transition Register (OTR).

Conditions The most-significant bit cannot be set true.

Group Status

Syntax
```
STATus:OPERation:PTRansition <bit_value>
STATus:OPERation:PTRansition?
```

Arguments <bit_value> ::= <NR1> is the positive transition filter value.

Returns A single <NR1> value showing the contents of the OTR.

Examples `STATUS:OPERATION:PTRANSITION 0` sets the positive transition filter value to 17.

`STATUS:OPERATION:PTRANSITION?` might return 0.

STATus:PRESet (No Query Form)

This command sets the Operation Enable Register (OENR) and Questionable Enable Register (QENR).

Group Status

Syntax `STATus:PRESet`

Examples `STATUS:PRESET` resets the SCPI enable registers.

STATus:QUEStionable:CONDition? (Query Only)

This command returns the status of the Questionable Condition Register.

Group Status

Syntax `STATus:QUEStionable:CONDition?`

Related Commands STATus:QUEStionable:ENABle, STATus:QUEStionable[:EVENt]?

Returns A single <NR1> value.

Examples `STATUS:QUESTIONABLE:CONDITION?` might return 0.

STATus:QUEStionable:ENABle

This command sets or returns the enable mask of the Questionable Enable Register (QENR) which allows true conditions in the Questionable Event Register to be reported in the summary bit.

Group Status

Syntax

```
STATus:QUEStionable:ENABle <bit_value>
STATus:QUEStionable:ENABle?
```

Arguments

<bit_value> ::= <NR1> is the enable mask of the QENR.

Returns

A single <NR1> value showing the contents of the QENR.

Examples

`STATUS:QUESTIONABLE:ENABLE 64` enables the FREQuency bit.

`STATUS:QUESTIONABLE:ENABLE?` might return 64, showing that the bits in the QENR have the binary value 00000000 00100000, which means that the FREQuency bit is valid.

STATus:QUEStionable[:EVENt]? (Query Only)

This command returns the contents of the Questionable Event Register (QEVR). Reading the QEVR clears it.

Group

Status

Syntax

```
STATus:QUEStionable[:EVENt]?
```

Returns

A single <NR1> value showing the contents of the QEVR.

Examples

`STATUS:QUESTIONABLE:EVENT?` might return 64, showing that the bits in the QEVR have the binary value 00000000 00100000, which means that the FREQuency bit is set.

STATus:QUEStionable:NTRansition

This command sets or returns the negative transition filter value of the Questionable Transition Register (QTR).

Group

Status

Syntax

```
STATus:QUEStionable:NTRansition <bit_value>
STATus:QUEStionable:NTRansition?
```

Arguments <bit_value> ::= <NR1> is the negative transition filter value.

Returns A single <NR1> value showing the contents of the QTR.

Examples `STATUS:QUESTIONABLE:NTRANSITION 32` sets the negative transition filter value to 32.

`STATUS:QUESTIONABLE:NTRANSITION?` might return 32, indicating the negative transition filter value is 32.

STATus:QUEStionable:PTRansition

This command sets or queries the positive transition filter value of the Questionable Transition Register (QTR).

Group Status

Syntax

```
STATus:QUEStionable:PTRansition <bit_value>
STATus:QUEStionable:PTRansition?
```

Arguments <bit_value> ::= <NR1> is the positive transition filter value.

Returns A single <NR1> value showing the contents of the QTR.

Examples `STATUS:QUESTIONABLE:PTRANSITION 0` sets the positive transition filter value to 0.

`STATUS:QUESTIONABLE:PTRANSITION?` might return 0, indicating that the positive transition filter value is 0.

*STB? (Query Only)

This command returns the contents of Status Byte Register. (See page 3-1, *Status and events*.)

Group IEEE mandated and optional

Syntax

```
*STB?
```

Related Commands *CLS, *ESE, *ESR?, *SRE

Returns A single <NR1> value.

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

SYNChronize:ADJust:[STARt] (No Query Form)

This command only performs a system sample rate calibration on the synchronized system. This command may take up to 3 minutes to complete.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is not enabled, this command is not available.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Synchronization

Syntax `SYNChronize:ADJust:[STARt]`

Examples `SYNCHRONIZE:ADJUST:[START]` starts the calibration on the synchronized system.

SYNChronize:CONFigure

This command configures the ports in a synchronized system and forces an initialization within the selected configuration.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

When synchronization is not enabled, this command is not available.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Synchronization

Syntax

```
SYNChronize:CONFigure <port_configuration>
SYNChronize:CONFigure?
```

Arguments

<port_configuation> ::= <NR1> (NR1 = Sum of Port 1, Port 2, Port 3 and Port 4)

Where:

Port 1 = 1 (shall always be on as Master of synchronized system)
Port 2 = 2 if enabled, otherwise 0
Port 3 = 4 if enabled, otherwise 0
Port 4 = 8 if enabled, otherwise 0

Returns

<NR1> ::= <port_configuration>

Examples

```
SYNCHRONIZE:CONFIGURE 3
*OPC?
```
enables Ports 1 and 2 in this synchronized system. The overlapping command is followed with an Operation Complete query.

`SYNCHRONIZE:CONFIGURE?` might return 15 indicating that Ports 1, 2, 3, and 4 are enabled in this synchronized system.

SYNChronize:DESKew:ABORt (No Query Form)

This command cancels a system deskew calibration.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

The command might take up to 10 minutes to cancel.

When synchronization is not enabled, this command is not available.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Synchronization

Syntax

```
SYNChronize:DESKew:ABORt
```

Examples

```
SYNCHRONIZE:DESKEW:ABORT
*OPC?
```
returns when deskew calibration is cancelled. The overlapping command is followed with an Operation Complete query.

SYNChronize:DESKew:[STARt] (No Query Form)

This command only performs a system deskew calibration.

Conditions

This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

The command might take up to 30 minutes to complete.

When synchronization is not enabled, this command is not available.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Synchronization

Syntax

```
SYNChronize:DESKew:[STARt]
```

Examples

```
SYNCHRONIZE:DESKEW:[START]
*OPC?
```

returns when the deskew calibration is complete. The overlapping command is followed with an Operation Complete query.

SYNChronize:DESKew:STATe? (Query Only)

This command returns the state of the system deskew condition.

Conditions The command is only valid when synchronization is enabled and the instrument is the master.

When synchronization is enabled and the instrument is not the master, the command returns 0.

Group Synchronization

Syntax

```
SYNChronize:DESKew:STATe?
```

Returns 1, the deskew calibration is running.
0, the deskew calibration is stopped, cancelled, complete, or when synchronization is disabled.

Examples `SYNCHRONIZE:DESKEW:STATE?` returns 0 when the deskew calibration is cancelled or complete.

SYNChronize:ENABle

This command enables or disables synchronization in the instrument.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Synchronization

Syntax

```
SYNChronize:ENABle {OFF|ON|0|1}
SYNChronize:ENABle?
```

Arguments OFF or 0 disables synchronization. OFF or 0 is the default value.
ON or 1 enables synchronization.

Returns

A single <Boolean> value.

Examples

`SYNCHRONIZE:ENABLE ON`
`*OPC?`
enables synchronization in the instrument to be part of a synchronized system. The overlapping command is followed with an Operation Complete query.

`SYNCHRONIZE:ENABLE?` might return 0 indicating the synchronization is not enabled.

SYNChronize:TYPE? (Query Only)

This command returns the instrument type (master or slave) in the synchronized system.

Conditions

This command is not active until synchronization enable has completed.

Group

Synchronization

Syntax

`SYNChronize:TYPE?`

Returns

`MAST` when synchronization is enabled and the instrument is the master of the synchronized system.
`SLAV` when synchronization is enabled and the instrument is a slave in the synchronized system.
`UNKN` when the instrument is unknown in the synchronized system. This indicates synchronization has not been enabled or a possible cable connection problem.

Examples

`SYNCHRONIZE:TYPE?` returns MAST indicating synchronization is enabled and the instrument is the master.

SYSTem:DATE

This command sets or returns the system date. When the values are nonintegers, they are rounded off to nearest integral values.

Group System

Syntax `SYSTem:DATE <year>,<month>,<day>`

Arguments <year>::=<NR1> (Four digit number)
<month>::=<NR1> from 1 to 12
<day>::=<NR1> from 1 to 31

Returns <year>,<month>,<day>

Examples `SYSTEM:DATE 2012,11,20` sets the date to November 20, 2012.

SYSTem:ERRor:ALL? (Query Only)

This command returns the error and event queue for all the unread items and removes them from the queue.

Group System

Syntax `SYSTem:ERRor:ALL?`

Returns <ecode>,"<edesc>[;<einfo>]"{,<ecode>,"<edesc>[;<einfo>]"}

Where:

<ecode> ::= <NR1> is the error/event code.
<edesc> ::= <string> is the description on the error/event.
<einfo> ::= <string> is the detail of the error/event.

If the queue is empty, the response is 0, "No error".

Examples `SYSTEM:ERROR:ALL?` might return -113, "Undefined header", indicating the command was not a recognized command.

SYSTem:ERRor:CODE:ALL? (Query Only)

This command returns the error and event queue for the codes of all the unread items and removes them from the queue.

Group System

Syntax `SYSTem:ERRor:CODE:ALL?`

Returns <ecode>{,<ecode>}

Where:

<ecode> ::= <NR1> is the error/event code.
If the queue is empty, the response is 0.

Examples `SYSTEM:ERROR:CODE:ALL?` might return -101,-108.

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

This command returns the error and event queue for the next item and removes it from the queue.

Group System

Syntax `SYSTem:ERRor:CODE[:NEXT]?`

Returns <ecode> ::= <NR1> is the error and event code.

Examples `SYSTEM:ERROR:CODE[:NEXT]?` might return -101.

SYSTem:ERRor:COUNt? (Query Only)

This command returns the error and event queue for the number of unread items.

Group System

Syntax `SYSTem:ERRor:COUNt?`

Returns <enum> ::= <NR1> is the number of errors/events.
If the queue is empty, the response is 0.

Examples `SYSTEM:ERROR:COUNT?` might return 3.

SYSTem:ERRor:DIALog

This command enables or disables error dialogs from displaying on the UI when an error condition occurs on the AWG.

Group System

Syntax

```
SYSTem:ERRor:DIALog <show_dialog>
SYSTem:ERRor:DIALog?
```

Arguments <show_dialog> ::= <Boolean>

0 hides the error dialogs.
1 displays the error dialogs.
`*RST` sets this value to 1.

Returns A single <NR1> value.

Examples `SYSTEM:ERROR:DIALOG 0` hides the error dialogs from display.

`SYSTEM:ERROR:DIALOG?` might return 1, indicating that error messages will be displayed on the AWG.

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

This command returns data from the error and event queues.

Group System

Syntax

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

Returns <Error number>, <error description>
Error number <NR1>.
error description <string>.

Examples `SYSTEM:ERROR:NEXT?` might return 0,"No error" indicating there are not errors.

SYSTem:TIME

This command sets or returns the system time (hours, minutes and seconds). This command is equivalent to the time setting through the Windows Control Panel.

Group System

Syntax

```
SYSTem:TIME <hour>,<minute>,<second>
SYSTem:TIME?
```

Arguments <hour>,<minute>,<second>

<hour> ::= <NR1> specifies the hours. Range: 0 to 23.
<minute> ::= <NR1> specifies the minutes. Range: 0 to 59.
<second> ::= <NR1> specifies the seconds. Range: 0 to 59.

Returns <hour>,<minute>,<second>

<hour> ::= <NR1> specifies the hours.
<minute> ::= <NR1> specifies the minutes.
<second> ::= <NR1> specifies the seconds.
These values are rounded to the nearest integer.

Examples `SYSTEM:TIME 10,15,30` sets the time to 10:15:30.

`SYSTEM:TIME?` might return 12,20,32 indicating the system time is 12:20:32.

SYSTem:VERSion? (Query Only)

This command returns the SCPI version number to which the command conforms.

Group System

Syntax

```
SYSTem:VERSion?
```

Returns A single <NR2> value.
<NR2> ::= YYYY.V where YYYY is the year version and V is revision number for that year.

Examples `SYSTEM:VERSION?` might return 1999.0.

*TRG (No Query Form)

This command generates a trigger event for Trigger A only. This is equivalent to pressing the Trig A button on front panel.

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group IEEE mandated and optional

Syntax `*TRG`

Related Commands TRIGger[:SEQuence][:IMMediate]

Examples `*TRG` generates a trigger event.

TRIGger[:SEQuence][:IMMediate] (No Query Form)

This command generates a trigger A or B event.

If a trigger is not specified, the command is then equivalent to the *TRG command.

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group Trigger

Syntax `TRIGger[:SEQuence][:IMMediate] [<input_trigger>]`

Related Commands *TRG

[SOURce[n]]:TINPut

Arguments <input_trigger> ::= {ATRigger|BTRigger}
Defaults to trigger A if not specified.

Examples `TRIGGER:SEQUENCE:IMMEDIATE ATRIGGER` generates a trigger A event.

TRIGger[:SEQuence]:IMPedance

This command sets or returns the external trigger impedance. It applies only to the external trigger.

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group Trigger

Syntax

```
TRIGger[:SEQuence]:IMPedance <impedance>[,<input_trigger>]
TRIGger[:SEQuence]:IMPedance?  [<input_trigger>]
```

Arguments <impedance> ::= <NR1> the value will be 50 or 1000.
<input_trigger> ::= {ATRigger|BTRigger}, Defaults to trigger A if not specified.

`*RST` sets this to 50.

Returns <NR1>

Examples `TRIGGER:SEQUENCE:IMPEDANCE 50` selects 50 Ω impedance for the external trigger A input.

`TRIGGER:SEQUENCE:IMPEDANCE 50,BTRIGGER` selects 50 Ω impedance for the external trigger B input.

`TRIGGER:SEQUENCE:IMPEDANCE? BTRIGGER` might return 1000, indicating impedance for external trigger B input is set to 1 KΩ.

TRIGger[:SEQuence]:INTerval

This command sets or returns the internal trigger interval.

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group Timing

Syntax
```
TRIGger[:SEQuence]:INTerval<NR3>
TRIGger[:SEQuence]:INTerval?
```

Arguments A single <NR3> value; range is from 1 to 10 µs.

Returns A single <NR3> value.

Examples `TRIGGER[:SEQUENCE]:INTERVAL 5E-6` sets the internal trigger interval to 5 µs.

`TRIGGER[:SEQUENCE]:INTERVAL?` might return 8.0000000000E-3 indicating 8 µs.

TRIGger[:SEQuence]:LEVel

This command sets or returns the external trigger input level (threshold).

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group Trigger

Syntax
```
TRIGger[:SEQuence]:LEVel <NRf>[,<input_trigger>]
TRIGger[:SEQuence]:LEVel?  [<input_trigger>]
```

Related Commands TRIGger[:SEQuence]:SOURce

Arguments A single <NRf> value.
Range: –5 V to 5 V.
<input_trigger> ::= {ATRigger|BTRigger}, Defaults to ATR if not specified.

`*RST` sets this to 1.4 V.

Returns <NRf>

Examples `TRIGGER:SEQUENCE:LEVEL 0.2` sets the trigger A level to 200 mV.

`TRIGGER:SEQUENCE:LEVEL? ATRIGGER` might return 200.0000000000E-3 indicating the Trigger A input level is 200 mV.

TRIGger[:SEQuence]:MODE

This command sets or returns the trigger timing used when an external trigger source is being used.

Conditions The trigger run mode must be set to Triggered or Trig'd Continuous.

When synchronization is enabled and the instrument is the master, the command choice is limited to SYNChronous.

When synchronization is enabled and the instrument is not the master, this command is not available.

Group Trigger

Syntax
```
TRIGger[:SEQuence]:MODE
{SYNChronous|ASYNchronous}[,<input_trigger>]
TRIGger[:SEQuence]:MODE? <input_trigger>
```

Arguments `SYNChronous`: Synchronous triggering. This is the recommended trigger type when using the Sync Clock Out to synchronize with external devices.
`ASYNchronous:` Asynchronous triggering. This is the fastest triggering type.
<input_trigger> ::= {ATRigger|BTRigger}
Defaults to trigger A if not specified.
`*RST` sets this to `ASYNchronous`.

Returns ASYN
SYNC

Examples `TRIGGER[:SEQUENCE]:MODE ASYNCHRONOUS` sets the trigger timing to asynchronous type.

`TRIGGER[:SEQUENCE]:MODE?` might return ASYN, indicating that the trigger mode is set to Asynchronous triggering.

TRIGger[:SEQuence]:SLOPe

This command sets or returns the polarity of the external trigger slope. Use this command to set the polarity in modes other than gated mode.

Conditions

When synchronization is enabled and the instrument is not the master, this command is not available.

Group

Trigger

Syntax

```
TRIGger[:SEQuence]:SLOPe
{POSitive|NEGative}[,<input_trigger>]
TRIGger[:SEQuence]:SLOPe?  [<input_trigger>]
```

Related Commands

TRIGger[:SEQuence]:SOURce

Arguments

`POSitive` specifies a trigger on the rising edge of the external trigger signal.
`NEGative` specifies a trigger on the falling edge of the external trigger signal.
<input_trigger> ::= {ATRigger|BTRigger}, defaults to ATR if not specified.

`*RST` sets all external trigger slopes to POSitive.

Returns

POS
NEG

Examples

`TRIGGER[:SEQUENCE]:SLOPE NEGATIVE` selects the Negative slope for Trigger A.

`TRIGGER[:SEQUENCE]:SLOPE NEGATIVE,BTRIGGER` selects the Negative slope for Trigger B.

`TRIGGER[:SEQUENCE]:SLOPE? ATRIGGER` might return POS for Trigger A.

TRIGger[:SEQuence]:SOURce

This command sets or returns the trigger source.

NOTE. *This command exists for backwards compatibility. Use the command [SOURce[n]]:TINPut*

Conditions When synchronization is enabled and the instrument is not the master, this command is not available.

Group Trigger

Syntax
```
TRIGger[:SEQuence]:SOURce {EXTernal|INTernal}
TRIGger[:SEQuence]:SOURce?
```

Arguments `EXTernal` selects external trigger as the trigger source.
INTernal select internal interval timing as the trigger source.

`*RST` sets this to EXT.

Returns EXT
INT

Examples `TRIGGER[:SEQUENCE]:SOURCE EXTERNAL` selects the internal interval timing as the trigger source.

`TRIGGER[:SEQUENCE]:SOURCE?` might return EXT, indicating the trigger source is set to external trigger.

TRIGger[:SEQuence]:WVALue

NOTE. *This command exists for backwards compatibility. Use the commands OUTPut[n]:WVALue[:ANALog][:STATe] and OUTPut[n]:WVALue:MARKer[1|2].*

This command sets or returns the channel's output state when in the Waiting-for-trigger mode.

This value is applied to all channels and markers.

Group Trigger

Syntax
```
TRIGger[:SEQuence]:WVALue {FIRSt}
TRIGger[:SEQuence]:WVALue?
```

Related Commands OUTPut[n]:WVALue[:ANALog][:STATe], OUTPut[n]:WVALue:MARKer[1|2]

Arguments `FIRSt` specifies the first value of the waveform as the output level.

`*RST` sets this to ZERO.

Returns FIRS: Output is set to the first value of the waveform
ZERO: Output is set to zero volts.

Examples `TRIGGER[:SEQUENCE]:WVALUE FIRST` selects the first value of the waveform as the output level.

`TRIGGER[:SEQUENCE]:WVALUE?` might return FIRS, indicating that the trigger value while in the wait state is set to the first value of the waveform.

*TST? (Query Only)

This command executes the Power On Self Test (POST) and returns the results. Use DIAGnostic:RESult? to retrieve more detailed error information.

Group IEEE mandated and optional

Syntax `*TST?`

Related Commands DIAGnostic[:IMMediate], DIAGnostic:DATA?, DIAGnostic:RESult?

Returns A single <NR1> value.

A returned value of 0 indicates no error.

Examples `*TST?` might return –330 indicating that the self test failed.

*WAI (No Query Form)

This command is used to ensure that the previous command is complete before the next command is issued.

(See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group IEEE mandated and optional

Syntax

```
*WAI
```

Related Commands *OPC

Examples Assuming that you want to use the DIAG:START command, followed by the DIAG:RES? command. To ensure the DIAG:START command finishes before starting the next command, insert the *WAI command between the two commands, such as:

```
DIAG:START
*WAI
DIAG:RES?
```

WLISt:LAST? (Query Only)

This command returns the name of the most recently added waveform in the waveform list.

Group Waveform

Syntax `WLISt:LAST?`

Returns <string> ::= <wfm_name>

Examples `WLIST:LAST?` might return "untitled2".

WLISt:NAME? (Query Only)

This command returns a waveform name from the waveform list in the position specified by the index value.

Group Waveform

Syntax `WLISt:NAME? <Index>`

Arguments <Index> ::= <NR1>

Returns <string> ::= <wfm_name> is the waveform name specified by <index>.

Examples `WLIST:NAME? 21` might return "untitled21".

WLISt:SIZE? (Query Only)

This command returns the number of waveforms in the waveform list.

Group Waveform

Syntax `WLISt:SIZE?`

Returns <NR1>

Examples `WLIST:SIZE?` might return 2 when there are two waveforms in the waveform list.

WLISt:WAVeform:DATA

This command transfers waveform data from the external controller into the specified waveform or from a waveform to the external control program.

NOTE. *Before transferring data to the instrument, a waveform must be created using the WLISt:WAVeform:NEW command.*

Use this command to set both analog and marker data. To set only the marker data, use the WLISt:WAVeform:MARKer:DATA command.

Using StartIndex and Size, part of a waveform can be transferred at a time. Very large waveforms can be transferred in chunks.

Waveform data is always transferred in the LSB first format.

The format of the transferred data depends on the waveform type.

If Size is omitted, the length of waveform is assumed to be the value of the Size parameter.

Transferring large waveforms in chunks allows external programs to cancel the operation before it is completed.

The instrument supports floating point format. Floating point waveform data points occupy four bytes. So the total bytes will be four times the size of the waveform.

The minimum size of the waveform must be 1 and the maximum size depends on the instrument model and configuration.

This command has a limit of 999,999,999 bytes of data.

As the IEEE 488.2 is a limitation that the largest read or write that may occur in a single command is 999,999,999 bytes as the structure is defined as a '#' followed by a byte to determine the number of bytes to read '9'. '9' indicates that we need to read 9 bytes to determine the length of the following data block: 999,999,999 (separated by commas to help separate - they will not be present normally).

Because of the size limitation, it is suggested that the user make use of the starting index (and size for querying) to append data in multiple commands/queries.

Group Waveform

Syntax

```
WLISt:WAVeform:DATA
<wfm_name>[,<StartIndex>[,<Size>]],<block_data>
WLISt:WAVeform:DATA? <wfm_name>[,<StartIndex>[,<Size>]]
```

Related Commands WLISt:WAVeform:NEW, WLISt:WAVeform:MARKer:DATA

Arguments StartIndex, Size,<block_data>
<wfm_name> ::= <string>
<StartIndex> ::= <NR1>
<Size> ::= <NR1>
<block_data> ::= <IEEE 488.2 block>

Returns <block_data>

Examples `WLIST:WAVEFORM:DATA "TestWfm",0,1024,#44096xxxx…` transfers the waveform data to a waveform called "TestWfm" created earlier using the WLISt:WAVeform:NEW command. The data size is 1024 points (4096 bytes) and the start index is 0 (the first data point).

WLISt:WAVeform:DELete (No Query Form)

This command deletes the waveform from the waveform list.

NOTE. *When ALL is specified, all user-defined waveforms in the list are deleted in a single action. Note that there is no "UNDO" action once the waveforms are deleted. Use caution before issuing this command.*

If the deleted waveform is currently loaded into waveform memory, it is unloaded. If the RUN state of the AWG is ON, the state is turned OFF. If the channel is on, it will be switched off.

Group Waveform

Syntax `WLISt:WAVeform:DELete {<wfm_name>|ALL}`

Related Commands WLISt:SIZE?

Arguments <wfm_name> ::= <string>

Examples `WLIST:WAVEFORM:DELETE ALL` deletes all user-defined waveforms from the currently loaded setup.

`WLIST:WAVEFORM:DELETE "Test1"` deletes a waveform called "Test1".

WLISt:WAVeform:GRANularity? (Query Only)

This command returns the granularity of sample points required for the waveform to be valid. The number of sample points of a single channel instrument must be divisible by 2.

Group Waveform

Syntax `WLISt:WAVeform:GRANularity?`

Related Commands WLISt:WAVeform:LMINimum?, WLISt:WAVeform:LMAXimum?

Returns A single <NR1> value.

Examples `WLIST:WAVEFORM:GRANULARITY?` might return 2, indicating that the number of sample points must be divisible by 2.

WLISt:WAVeform:LENGth? (Query Only)

This command returns the size of the waveform. The returned value represents data points (not bytes).

Group Waveform

Syntax `WLISt:WAVeform:LENGth? <wfm_name>`

Arguments <wfm_name> ::= <string>

Returns <NR1>

Examples `WLIST:WAVEFORM:LENGTH? "Sine 360"` might return 360.

WLISt:WAVeform:LMAXimum? (Query Only)

This command returns the maximum number of waveform sample points allowed.

Conditions The returned value is dependent on the instrument model, the installed options, and sampling rate setting.

Group Waveform

Syntax `WLISt:WAVeform:LMAXimum?`

Related Commands WLISt:WAVeform:LMINimum?

Returns A single <NR1> value.

Examples `WLIST:WAVEFORM:LMAXIMUM?` might return 2000000000, indicating that the maximum number of allowed waveform sample points is 2 Giga samples.

WLISt:WAVeform:LMINimum? (Query Only)

This command returns the minimum number of waveform sample points required for a valid waveform. The number of required sample points is dependent on the instrument model.

Group Waveform

Syntax `WLISt:WAVeform:LMINimum?`

Related Commands WLISt:WAVeform:LMAXimum?

Returns A single <NR1> value.

Examples `WLIST:WAVEFORM:LMINIMUM?` might return 2400, indicating that the minimum number of waveform sample points required is 2.4 k.

WLISt:WAVeform:MARKer:DATA

This command sets or returns the waveform marker data.

NOTE. *This command returns or sends only marker data for the waveform.*

Each marker data occupies one bit. Two most significant bits of each byte are used for marker1 and marker2 (bit 6 for marker1 and bit 7 for marker2).

You will have to use bit masks to obtain the actual value.

When used on a waveform with n data points, you get only n bytes, each byte having values for both markers.

Block data can be sent in batches using "Size" and "StartIndex" parameters.

This command has a limit of 999,999,999 bytes of data. If this limit is insufficient, consider the following alternatives:

- Send a more efficient file format using MMEM:DATA.
- Use Ethernet (ftp, http, or file sharing) to transfer the file.

Group Waveform

Syntax

```
WLISt:WAVeform:MARKer:DATA
<wfm_name>[,<StartIndex>[,<Size>]],<block_data>
WLISt:WAVeform:MARKer:DATA?
<wfm_name>[,<StartIndex>[,<Size>]]
```

Related Commands

WLISt:WAVeform:DATA

Arguments

<wfm_name> ::= <string>
<StartIndex> ::= <NR1>
<Size> ::= <NR1>
<block_data> ::= <IEEE 488.2 block>

Returns

<block_data>

Examples

`WLIST:WAVEFORM:MARKER:DATA "myWaveform",0,1000,#41000….` transfers the marker data to the waveform named myWaveform (previously created with the WLISt:WAVeform:NEW command.

`WLIST:WAVEFORM:MARKER:DATA? "myWaveform",0,1000` returns 1000 marker values from myWaveform starting at the first sample.

WLISt:WAVeform:NEW (No Query Form)

This command creates a new empty waveform in the waveform list of the current setup.

Group

Waveform

Syntax

```
WLISt:WAVeform:NEW <wfm_name>,<Size>
```

Related Commands

WLISt:WAVeform:DATA

Arguments

<wfm_name> ::= <string>
<Size> ::= <NR1>

Examples

`WLIST:WAVEFORM:NEW "Test1", 1024` creates a new waveform called Test1 with 1024 points.

WLISt:WAVeform:NORMalize (No Query Form)

This command normalizes a waveform in the waveform list.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Waveform

Syntax

```
WLISt:WAVeform:NORMalize <wfm_name>,{FSCale|ZREFerence}
```

Arguments <wfm_name> ::= <string>
FSCale normalizes the waveform to the full amplitude range.
ZREFerence normalizes the waveform while preserving the offset.

Examples

```
WLIST:WAVEFORM:NORMALIZE "Untitled25",FSCALE
*OPC?
```

normalizes the waveform titled "Untitled25", if it exists, using full scale. The overlapping command is followed with an Operation Complete query.

WLISt:WAVeform:RESample (No Query Form)

This command resamples the number of points in a waveform in the waveform list.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Waveform

Syntax

```
WLISt:WAVeform:RESample <wfm_name>,<size>
```

Arguments <wfm_name ::= <string>
<size>::=<NR1>

Examples

```
WLIST:WAVEFORM:RESAMPLE "Untitled25", 1024
*OPC?
```

resamples the waveform titled "Untitled25" to 1024 points. The overlapping command is followed with an Operation Complete query.

WLISt:WAVeform:SHIFt (No Query Form)

This command shifts the phase of a waveform in the waveform list.

Conditions This is an overlapping command. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Group Waveform

Syntax

```
WLISt:WAVeform:SHIFt <wfm_name>,<phase>
```

Arguments <wfm_name ::= <string>
<phase> ::= <NR1>

Returns <wfm_name::=<string>
<Size>::=<NR3>

Examples

```
WLIST:WAVEFORM:SHIFT "Untitled25",180
*OPC?
```

shifts the waveform titled "Untitled25" (if it exists) by 180 degrees. The overlapping command is followed with an Operation Complete query.

WLISt:WAVeform:TSTamp? (Query Only)

This command returns the timestamp of the waveform.

***NOTE.** The timestamp is updated whenever the waveform is created or changed.*

The command returns the date as a string in the form yyyy/mm/dd hh:mm:ss (a blank space between date and time).

Group

Waveform

Syntax

```
WLISt:WAVeform:TSTamp?  <wfm_name>
```

Arguments

<wfm_name> ::= <string>

Returns

"yyyy/mm/dd hh:mm:ss" is the waveform timestamp.

Where
yyyy refers to a four-digit year number mm refers to two-digit month number from 01 to 12.
dd refers to two-digit day number in the month.
hh refers to two-digit hour number mm refers to two-digit minute number.
ss refers to two-digit second number.

Examples

`WLIST:WAVEFORM:TSTAMP? "Sine"` might return "2012/07/25 9:05:21" which is the date and time the "Sine" waveform was created or last modified.

WLISt:WAVeform:TYPE? (Query Only)

This command returns the type of the waveform.

***NOTE.** This command exists for backwards compatibility.*

Group

Waveform

Syntax

```
WLISt:WAVeform:TYPE? <wfm_name>
```

Arguments

`<wfm_name>` ::= <string>

Returns

REAL

Examples

`WLIST:WAVEFORM:TYPE? "Ramp1000"` returns REAL.

Status and events

Status and events

The SCPI interface in the analyzer includes a status and event reporting system that enables the user to monitor crucial events that occur in the instrument. The instrument is equipped with four registers and one queue that conform to IEEE Std 488.2. This section discusses these registers and queues along with status and event processing.

Status and event reporting system

The following figure outlines the status and event reporting mechanism offered in the AWG70000A Series arbitrary waveform generators. It contains three major blocks

- Standard Event Status
- Operation Status
- Questionable Status (fan-out structure)

The processes performed in these blocks are summarized in the Status Byte. The three blocks contain four types of registers as shown in the following table.

Table 3-1: Register type

Register	Description
Condition register	Records event occurrence in the instrument. Read only.
Transition register (positive/negative)	A positive transition filter allows an event to be reported when a condition changes from false to true. A negative filter allows an event to be reported when a condition changes from true to false. Setting both positive and negative filters true allows an event to be reported anytime the condition changes. Clearing both filters disables event reporting.
Event register	Records events filtered by the transition register. Read only.
Enable register	Masks the event register to report in the summary bit. User-definable.

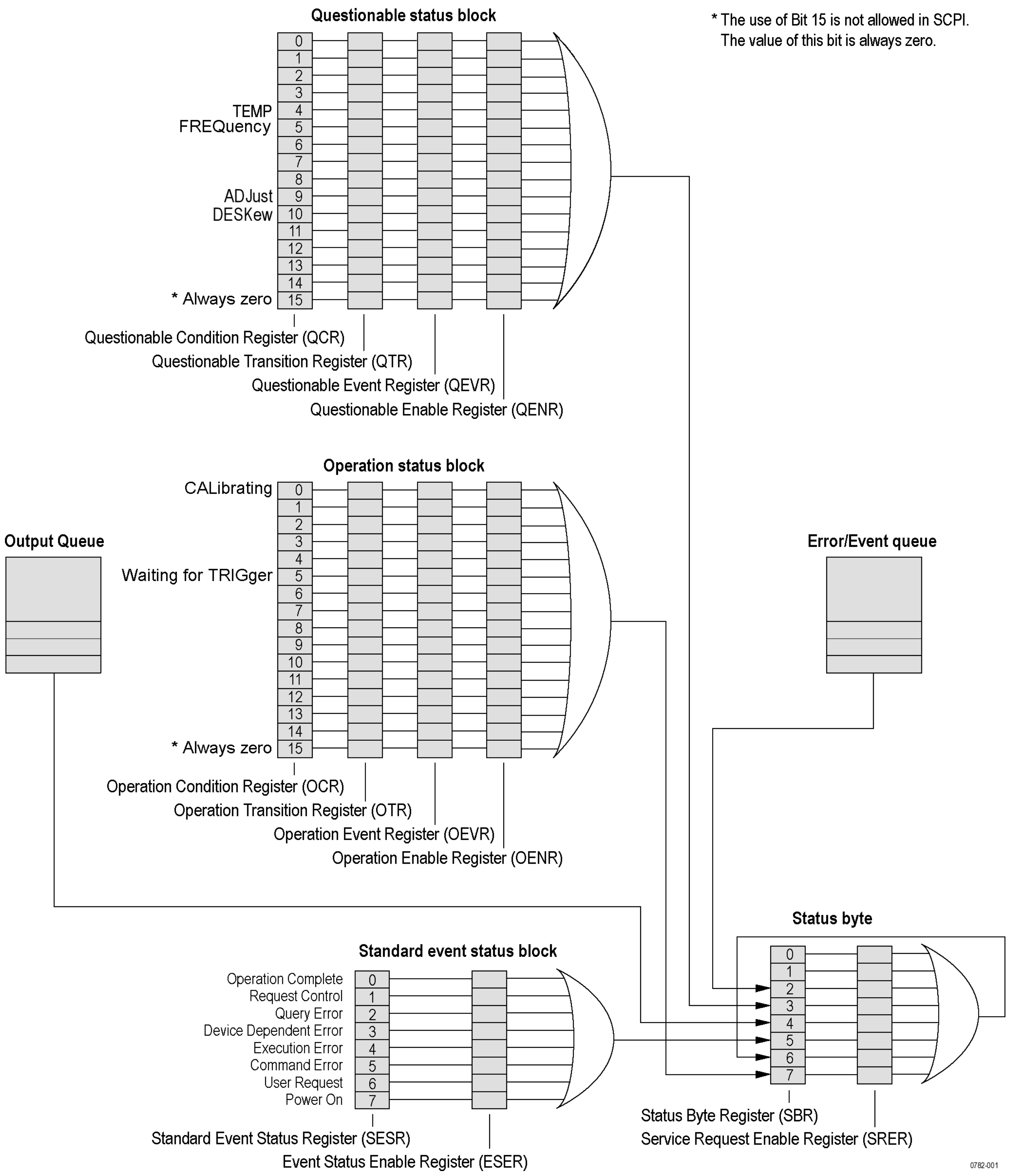

Questionable status block
* The use of Bit 15 is not allowed in SCPI.
The value of this bit is always zero.
TEMP
FREQuency
ADJust
DESKew
* Always zero
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Questionable Condition Register (QCR)
Questionable Transition Register (QTR)
Questionable Event Register (QEVR)
Questionable Enable Register (QENR)
Operation status block
CALibrating
Waiting for TRIGger
* Always zero
Output Queue
Error/Event queue
Operation Condition Register (OCR)
Operation Transition Register (OTR)
Operation Event Register (OEVR)
Operation Enable Register (OENR)
Status byte
Standard event status block
Operation Complete
Request Control
Query Error
Device Dependent Error
Execution Error
Command Error
User Request
Power On
Standard Event Status Register (SESR)
Event Status Enable Register (ESER)
Status Byte Register (SBR)
Service Request Enable Register (SRER)
0782-001

Figure 3-1: Status/Event reporting mechanism

Status byte

The Status Byte contains the following two registers

- Status Byte Register (SBR)
- Service Request Enable Register (SRER)

Status Byte Register (SBR)

The SBR is made up of 8 bits. Bits 4, 5 and 6 are defined in accordance with IEEE Std 488.2. These bits are used to monitor the output queue, SESR and service requests, respectively. The contents of this register are returned when the *STB? query is used.

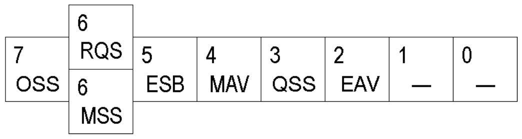


Figure 3-2: Status Byte Register (SBR)

Table 3-2: SBR bit functions

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

Service Request Enable Register (SRER)

The SRER is made up of bits defined exactly the same as bits 0 through 7 in the SBR as shown in the following figure. This register is used by the user to determine what events will generate service requests.

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

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

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

7	6	5	4	3	2	1	0
OSB	—	ESB	MAV	QSB	—	—	—

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

Standard Event Status Block (SESB)

Reports the power on/off state, command errors, and the running state. It consists of the following registers

- Standard Event Status Register (SESR)
- Event Status Enable Register (ESER)

These registers are made up of the same bits defined in the following figure and table. Use the *ESR? query to read the contents of the SESR. Use the *ESE() command to access the ESER.

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

Figure 3-4: Standard event status register

Table 3-3: Standard event status register bit definition

Bit	Description
7	Power On (PON). Indicates that the power to the instrument is on.
6	Not used.
5	Command Error (CME). Indicates that a command error has occurred while parsing by the command parser was in progress.
4	Execution Error (EXE). Indicates that an error occurred during the execution of a command. Execution errors occur for one of the following reasons ■ When a value designated in the argument is outside the allowable range of the instrument, or is in conflict with the capabilities of the instrument. ■ When the command could not be executed properly because the conditions for execution differed from those essentially required.
3	Device-Dependent Error (DDE). An instrument error has been detected.

Table 3-3: Standard event status register bit definition (cont.)

Bit	Description
2	Query Error (QYE). Indicates that a query error has been detected by the output queue controller. Query errors occur for one of the following reasons ■ An attempt was made to retrieve messages from the output queue, despite the fact that the output queue is empty or in pending status. ■ The output queue messages have been cleared despite the fact that they have not been retrieved.
1	Not used.
0	Operation Complete (OPC). This bit is set with the results of the execution of the *OPC command. It indicates that all pending operations have been completed.

When an event occurs, the SESR bit corresponding to the event is set, resulting in the event being stacked in the Error/Event Queue. The SBR OAV bit is also set. If the bit corresponding to the event has also been set in the ESER, the SBR ESB bit is also set. When a message is sent to the Output Queue, the SBR MAV bit is set.

Operation status block

The operation status block contains conditions that are part of the instrument's normal operation. It consists of the following registers

- Operation Condition Register (OCR)
- Operation Positive/ Negative Transition Register (OPTR/ONTR)
- Operation Event Register (OEVR)
- Operation Enable Register (OENR)

These registers are made up of the same bits defined in the following table and figure. Use the STATus:OPERation commands to access the operation status register set.

15	14	13	12	11	10	9	8	7	6	5 WFT	4	3	2	1	0 CAL

Figure 3-5: Operation status register

Table 3-4: Operation status register bit definition

Bit	Description
15	Always zero (0).
14 - 6	Not used.
5	Waiting for trigger (WFT). Indicates that the instrument is waiting for a trigger event to occur.

Table 3-4: Operation status register bit definition (cont.)

Bit	Description
4 - 1	Not used.
0	Calibrating (CAL). Indicates that the instrument is currently performing a calibration.

When the specified state changes in the OCR, its bit is set or reset. This change is filtered with a transition register, and the corresponding bit of the OEVR is set. If the bit corresponding to the event has also been set in the OENR, the SBR OSS bit is also set.

Questionable status block

The questionable status register set contains bits which give an indication of the quality of various aspects of the signal together with the fanned out registers as described in the next subsections. It consists of the following registers

- Questionable Condition Register (QCR)
- Questionable Positive/Negative Transition Register (QPTR/QNTR)
- Questionable Event Register (QEVR)
- Questionable Enable Register (QENR)

These registers are made up of the same bits defined in the following table and figure. Use the STATus:QUEStionable commands to access the questionable status register set.

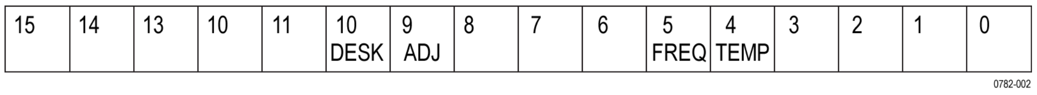


Figure 3-6: Questionable status register

Table 3-5: Questionable status register bit definition

Bit	Description
15	Always zero (0).
14 – 11	Not used.
10	DESKew Deskew calibration required due to temperature out of range.
9	ADJust (ADJ). External clock adjustment required.
8 – 6	Not used
5	FREQuency (FREQ). Using External Reference or frequency is out of range.

Table 3-5: Questionable status register bit definition (cont.)

Bit	Description
4	TEMPerature (TEMP). Calibration required due to instrument temperature change.
3 – 0	Not used.

When the specified state changes in the QCR, its bit is set or reset. This change is filtered with a transition register, and the corresponding bit of the QEVR is set. If the bit corresponding to the event has also been set in the QENR, the SBR QSS bit is also set.

Queues

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

Output queue

The output queue is a FIFO (first in, first out) queue and holds response messages to queries, where they await retrieval. When there are messages in the queue, the SBR MAV bit is set.

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

Event queue

The event queue is a FIFO queue and stores events as they occur in the analyzer. If more than 32 events occur, event 32 will be replaced with event code -350 ("Queue Overflow"). The error code and text are retrieved using the SYSTem:ERRor queries.

Status and event processing sequence

The following figure shows an outline of the sequence for status and event processing.

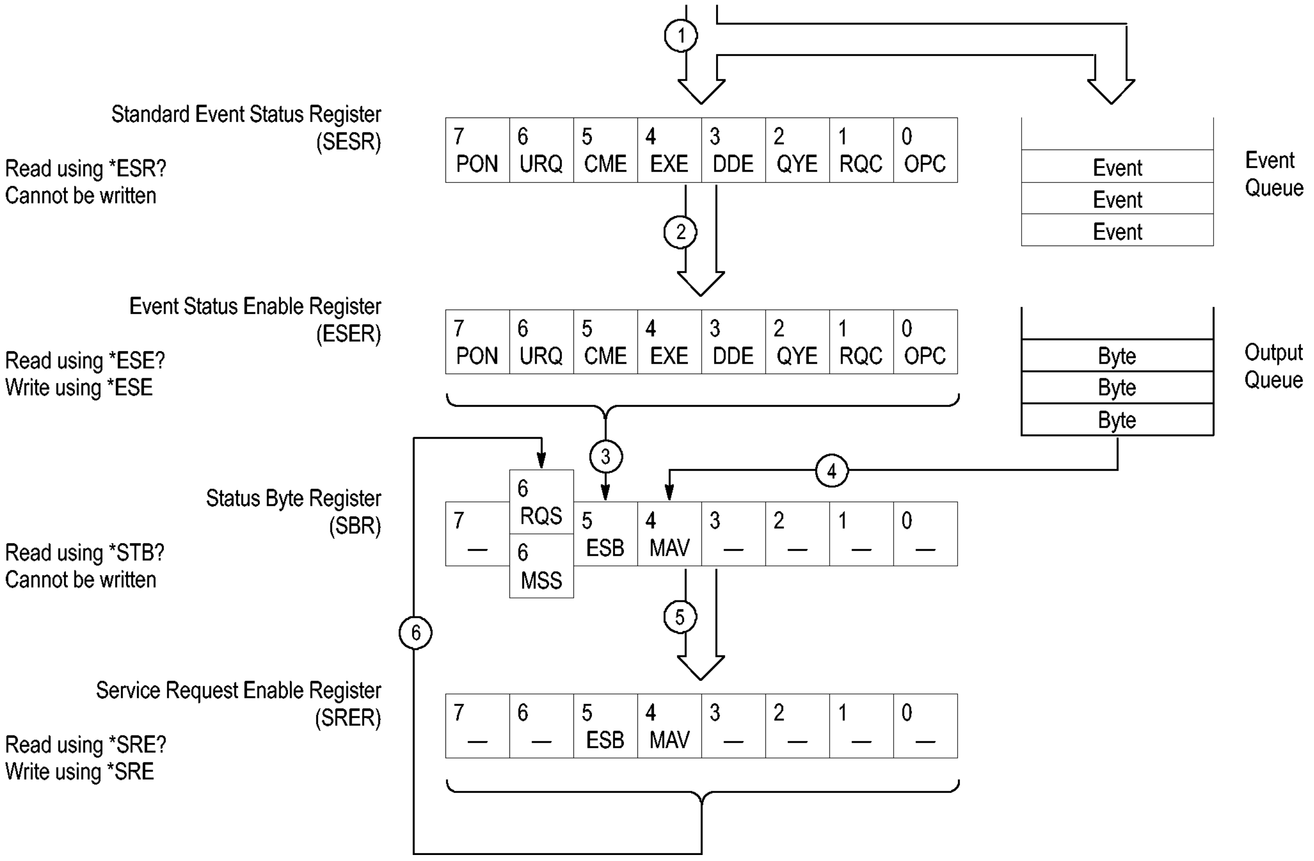


Figure 3-7: Status and event processing sequence

1. If an event has occurred, the SESR bit corresponding to that event is set and the event is placed in the event queue.
2. A bit corresponding to that event in the ESER has is set.
3. The SBR ESB bit is set to reflect the status of the ESER.
4. When a message is sent to the output queue, the SBR MAV bit is set.
5. Setting either the ESB or MAV bits in the SBR sets the respective bit in the SRER.
6. When the SRER bit is set, the SBR MSS bit is set and a service request is generated when using the GPIB interface.

Synchronizing execution

Almost all commands are executed in the order in which they are sent from the controller. However, some commands perform data analysis in another thread, and another command can thus be executed concurrently. These types of commands are called overlapping commands. (See page 2-9, *Sequential, blocking, and overlapping commands*.)

Some examples of these types of commands include the following.

- AWGControl:RUN
- CLOCk:JITTer
- MMEMory:SAVE:WFMX

You have two options to achieve command synchronization.

- Using the status and event reporting function
- Using synchronizing commands

Using the status and event reporting function

In the following example, the Operation Condition Register (OCR) is being used to provide synchronization.

```
STATus:OPERation:NTRansition 32
    // Set the filter of the OCR Waiting for Trigger bit
STATus:OPERation:ENABle 32
    // Enable the filter of the OCR Waiting for Trigger bit
*SRE 128
    // Set the SRER OSS bit
```

The command waits for generation of SRQ.

Using synchronizing commands

The IEEE-488.2 common commands include the following synchronizing commands

- *OPC
- *OPC?
- *WAI

Using the *OPC command. The *OPC command causes the AWG to sense the internal flag referred to as the “No-Operation-Pending” flag. (An on-going overlapped command would be an operation that is pending.) When the pending operation has completed, the Operation Complete (OPC) bit in the Event Status Register (ESR) is set. The user can poll the ESR register (*ESR?) or enable the service request process to be notified.

Using the *OPC? query. The *OPC? query causes the AWG to sense the internal flag referred to as the “No-Operation-Pending flag (same as the *OPC command).

When the pending operation has completed, a “1” will be returned to the client. This query does not use the ESR register and the service request process does not work.

Using the *WAI command. The *WAI command causes the AWG to sense the same internal flag, referred to as the No-Operation-Pending” flag. The *WAI command prevents any command or query from executing until any pending operation completes.

Error messages and codes

Error codes with negative values are SCPI standard codes. Error codes with positive values are unique to the AWG70000 series Arbitrary Waveform Generators.

Event codes and messages can be obtained by using the queries SYSTem:ERRor? and SYSTem:ERRor:ALL? These are returned in the following format

Command errors

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

Table 3-6: Command errors

Error code	Error message
-100	Command
-101	Invalid character
-102	Syntax
-103	Invalid separator
-104	Data type
-105	GET not allowed
-108	Parameter not allowed
-109	Missing parameter
-110	Command header
-111	Header separator
-112	Program mnemonic too long
-113	Undefined header
-114	Header suffix out of range
-120	Numeric data
-121	Invalid character in number
-123	Exponent too large
-124	Too many digits
-128	Numeric data not allowed
-130	Suffix
-131	Invalid suffix
-134	Suffix too long
-138	Suffix not allowed

Table 3-6: Command errors (cont.)

Error code	Error message
-140	Character data
-141	Invalid character data
-144	Character data too long
-148	Character data not allowed
-150	String data
-151	Invalid string data
-158	String data not allowed
-160	Block data
-161	Invalid block data
-168	Block data not allowed
-170	Expression
-171	Invalid expression
-178	Expression data not allowed
-180	Macro
-181	Invalid outside macro definition
-183	Invalid inside macro definition
-184	Macro parameter

Execution errors

These error codes are returned when an error is detected while a command is being executed.

Table 3-7: Execution errors

Error code	Error message
-200	Execution
-201	Invalid while in local
-202	Settings lost due to RTL
-210	Trigger
-211	Trigger ignored
-212	Arm ignored
-213	Init ignored
-214	Trigger deadlock
-215	Arm deadlock

Table 3-7: Execution errors (cont.)

Error code	Error message
-220	Parameter
-221	Settings conflict
-222	Data out of range
-223	Too much data
-224	Illegal parameter value
-225	Out of memory
-226	Lists not same length
-230	Data corrupt or stale
-231	Data questionable
-240	Hardware
-241	Hardware missing
-250	Mass storage
-251	Missing mass storage
-252	Missing media
-253	Corrupt media
-254	Media full
-255	Directory full
-256	Filename not found
-257	Filename
-258	Media protected
-260	Execution expression
-261	Math in expression
-270	Execution macro
-271	Macro syntax
-272	Macro execution
-273	Illegal macro label
-274	Execution macro parameter
-275	Macro definition too long
-276	Macro recursion
-277	Macro redefinition not allowed
-278	Macro header not found
-280	Program
-281	Cannot create program
-282	Illegal program name
-283	Illegal variable name
-284	Program currently running
-285	Program syntax

Table 3-7: Execution errors (cont.)

Error code	Error message
-286	Program runtime
-290	Memory use
-291	Out of memory
-292	Referenced name does not exist
-293	Referenced name already exists
-294	Incompatible type

Device specific errors

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

Table 3-8: Device specific errors

Error code	Error message
-300	Device specific or sequence step error
-310	System
-311	Memory
-312	PUD memory lost
-313	Calibration memory lost
-314	Save/Recall memory lost
-315	Configuration memory lost
-320	Storage fault
-321	Out of memory
-330	Self test failed
-340	Calibration failed
-350	Queue overflow
-360	Communication
-361	Parity in program message
-362	Framing in program message
-363	Input buffer overrun

Query and system errors

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

Table 3-9: Query errors

Error code	Error message
-400	Query error
-410	Query interrupted
-420	Query unterminated
-430	Query deadlocked
-440	Query unterminated after indefinite period
-500	Power on
-600	User request
-700	Request control
-800	Operation complete

AWG70000A series error codes

These error codes and messages are unique to the AWG70000A Series instruments.

Table 3-10: Device errors

Error code	Error message
500	Calibration in process.
501	Waiting for trigger.
550	Lost frequency lock with External Reference source.
551	External Reference frequency out of range.
552	Calibration recommended, temperature change.
553	Ext Clk adjustment recommended for frequency change.
554	Ext Clk adjustment recommended for temperature change.
555	Deskew Calibration recommended.
556	Synchronization Adjust recommended on master.
557	Sync Clock unlocked. Lost frequency lock with Clock In provided by the system – unable to play.
558	Sync Frequency out of range. Clock In frequency is higher or lower than the specified range or the value specified by the system.
559	Configuration recommended on Master. Configure the system on the Master or adjust if already configured to properly synchronize the system.
1000	Waveform allocation failed.

Table 3-10: Device errors (cont.)

Error code	Error message
1001	Registry write failed.
1002	Sequencing not enabled, Option 03 (Sequencing) not enabled – unable to complete the operation.
1003	Firmware load failed.
1004	Option system failed.
1100	Function generator failed.
1102	Function generator frequency too high.
1103	Function generator frequency too low.
1104	Function generator hardware failed.
1200	Load failure, unable to load waveform or sequence.
1201	Waveform load max length error, waveform length exceeds maximum samples - unable to load waveform. Maximum length is based on sample rate and options.
1202	Waveform load min length error, waveform length less than minimum samples - unable to load waveform. Use Modify waveform to increase the number of waveform points by adding points or repeating the waveform.
1203	Waveform load granularity error, length is not divisible by granularity - unable to load waveform. Use Modify waveform to add or subtract one point or repeat the waveform for 2 cycles.
1204	Play failed, no waveform assigned.
1205	No asset assigned to channel.
1206	Play failed, resampled waveform exceeds maximum. Use Modify waveform to decrease the number of waveform points.
1207	Play failed, resampled waveform too small. Use Modify waveform to increase the number of waveform points.
1208	Resampled waveform load granularity.
1209	Play failed, hardware failure.
1210	Play failed to stop, hardware failure.
1211	Failed to load, hardware failure.
1212	Sample rate not available, requested sample rate is not available; sample rate set to nearest value.
1213	Failed to load sequence, sequence step count exceeds hardware limit.
1214	Failed to load sequence, sequence step has no asset assigned.
1215	Failed to load sequence, repeat count of Sequence step exceeds hardware limit.

Table 3-10: Device errors (cont.)

Error code	Error message
1216	Failed to load sequence, sequence step contains invalid Goto step.
1217	Failed to load sequence, sequence step contains invalid Event Jump step.
1218	Failed to load sequence, pattern jump table contains invalid jump target.
1219	Hardware error, unable to load waveform or sequence due to hardware error.
1220	Empty sequence, sequence is empty - unable to load.
1221	Failed to load sequence, sequence step must contain waveforms of equal length.
1222	Failed to load sequence, pattern jump table size exceeds hardware limit.
1223	Failed to load sequence, total waveform(s) length exceeds maximum samples.
1224	Failed to find sequence, no sequence definition was found in the file.
1300	Unknown exception, unable to save file.
1301	File save error, unknown error - unable to save waveform.
1302	File restore error, unknown error - unable to open asset from setup file.
1303	Unknown exception, unable to open waveform.
1304	Recall waveform failed, duplicate name.
1307	File access failed.
1308	Recall waveform failed, missing parameter.
1309	Recall waveform failed, unsupported number of bits.
1310	Recall waveform failed, invalid marker type.
1311	Recall waveform failed, invalid marker data length.
1312	Recall waveform failed, waveform name and/or data not found.
1313	Recall waveform failed, unsupported waveform file type.
1314	Recall waveform failed, invalid sample data.
1315	Recall waveform failed, unable to read sample data.
1316	Recall waveform failed, unable to read Matlab HDF5 data set.
1317	Recall waveform failed, invalid IQ data format.
1318	Recall waveform failed, invalid DPX spectral data format.
1319	Recall waveform failed, invalid RSA header format.
1320	Recall waveform failed, data length error.
1321	Recall waveform failed, invalid data format.
1322	Recall waveform failed, invalid marker data format.

Table 3-10: Device errors (cont.)

Error code	Error message
1323	Recall waveform failed, invalid file extension.
1324	Recall waveform failed, invalid file header.
1325	Recall waveform failed, file type unknown.
1326	Recall waveform failed, file version not supported.
1327	Recall waveform failed, no waveform data.
1328	Asset not found, unable to import asset(s).
1329	Recall waveform failed, unable to open waveform from RSA file.
1330	Recall waveform failed, waveform format not supported.
1331	Invalid operation.
1332	Read failed, unable to open file.
1333	Export failed to write file.
1334	Recall waveform failed, unable to read filed.
1335	Export failed, out of disk space.
1336	File not found.
1337	File format error, file format not valid - unable to open file.
1338	Failed to delete file.
1340	Invalid save type, save type not valid - unable to save file.
1341	Asset name error, asset list already has an asset with that name - unable to save file.
1342	Asset not found, item is not in the Asset List - unable to save file.
1343	Recall failed, IQ waveform error.
1344	Restore setup failed, unable to open setup file.
1345	Error in file format or data, unable to restore the sequence and it's assets.
1346	Subsequences not supported, restored subsequences will be added to the Sequence List, but the sequence steps they occupied will be shown as Empty.
1347	Missing asset file(s), waveform or sequence file(s) not found; shown as Empty in the sequence table.
1348	Restore pattern table error, Pattern Jump table has too many rows; Restored the first 256 patterns only.
1600	Timing error, unable to change clock setting.
1601	Timing error, lost timing lock.
1602	Channel error, unable to change channel parameter.
1603	USB lock/unlock failed. Administrator permissions are required to lock or unlock the USB ports. Check the Windows security settings or contact your network administrator.
1604	Force Jump error, unable to force jump to specified step.

Table 3-10: Device errors (cont.)

Error code	Error message
1605	External Clock adjustment failed, check the Clock In signal.
1606	External Clock error, clock In differs from external clock adjustments - Check the Clock In signal or Adjust.
1606	External Clock error, Clock In differs from the external clock adjustments. Check the Clock In signal or Adjust.
1700	Resample failed, sample rate value not found. The waveform did not include the recommended sample rate.
1701	Failed to resample waveform.
1702	Resampling ratio too small.
1703	Resampling ratio too large.
1704	Shift/rotate failed.
1750	Step number exceeds max, exceeds max number of steps - failed to add step(s).
1751	Invalid step, invalid step specified.
1752	Add step(s) failure, failed to add step(s) to sequence.
1753	Insert step(s) failure, failed to insert step(s) to sequence.
1754	Remove step(s) failure, failed to remove step(s) from sequence.
1755	Failed to add track, exceeded maximum number of tracks.
1756	Failed to remove track, sequence must have at least one track.
1757	Invalid track number, invalid track number specified.
1758	Add track error, failed to add track.
1759	Remove track error, failed to remove track.
1760	Sequence name in use, name already used in Sequence List - unable to create sequence.
1761	Sequence creation failed, unable to create sequence.
1762	Paste error, clipboard values do not match paste area data type(s).
1800	Invalid name, invalid name or handle for asset.
1801	Renaming error, no name given - unable to rename asset.
1802	Asset name in use, name already used in list - unable to rename asset.
1803	Rename failed, linked file missing.
1804	File name in use, unable to rename asset.
1903	Calibrations are still running. Abort and try again.
2000	AWGSYNC01 Communication failed.
2001	AWGSYNC01 Connection failure.
2002	AWGSYNC01 FPGA Update failure.

Table 3-10: Device errors (cont.)

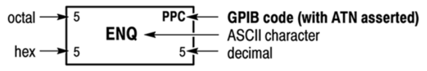
Error code	Error message
2003	AWGSYNC01 Configuration failure.
2004	AWGSYNC01 Deskew Calibration failure.
2005	Failed to set sample rate on Port 2.
2006	Failed to set sample rate on Port 3.
2007	Failed to set sample rate on Port 4.
2008	AWGSYNC01 time out failure.
2009	AWGSYNC01 alignment failure.
2010	AWGSYNC01 Play failure.
2011	AWGSYNC01 Configuration failure.
2012	AWGSYNC01 Adjust failure.
2013	AWGSYNC01 missing configuration failure.
2014	AWGSYNC01 missing configuration failure.
2015	AWGSYNC01 synchronization failure.
2016	AWGSYNC01 Deskew Calibration failure.
2017	AWGSYNC01 Deskew Calibration failure.

Appendices

Appendix A: Character charts

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: Raw socket specification

TCP/IP is used as the network protocol, and the port number is variable. Commands can be sent from the application program through the TCP/IP socket interface, and queries can be received through the interface.

- The Line Feed (LF) code is needed as a terminator at the end of a message.
- The IEEE 488.1 standard (for example, Device Clear or Service Request) is not supported.
- The Message Exchange Control Protocol in the IEEE 488.2 is not supported. However, common commands such as *ESE and the event handling features are supported.
- The Indefinite format (the block start at #0) in the <ARBITRARY BLOCK PROGRAM DATA> of the IEEE 488.2 is not supported.

Appendix C: Factory initialization settings

Commands affected by a factory initialization (*RST command) are listed in the following table and are also noted in their command description.

Table C-1: Factory initialization settings

Command	Default value
ACTive:MODE	NORMal
AWGControl:INTerleave:ADJustment:AMPLitude	0%
AWGControl:INTerleave:ADJustment:PHASe	0°
AWGControl:RMODe	CONTinuous
CALibration:LOG:DETails	0 (off)
CALibration:LOG:FAILuresonly	0 (off)
CLOCk:ECLock:DIVider	1
CLOCk:ECLock:MULTiplier	1
CLOCk:EREFerence:DIVider	1
CLOCk:EREFerence:FREQuency	35 MHz
CLOCk:EREFerence:MULTiplier	1
CLOCk:JITTer	0 (off)
CLOCk:OUTPut[:STATe]	0 (off)
CLOCk:PHASe[:ADJust]	0°
CLOCk:SOURce	INTernal
CLOCk:SOUT[:STATe]	0 (off)
CLOCk:SRATe	AWG70001A 50 GS AWG70002A 25 GS
DIAGnostic:CONTrol:COUNt	0
DIAGnostic:CONTrol:HALT	0 (off)
DIAGnostic:CONTrol:LOOP	ONCE
DIAGnostic:LOG:FAILuresonly	0 (off)
DIAGnostic:TYPE	NORMal
DISPlay[:PLOT][:STATe]	1 (on)
FGEN[:CHANnel[n]]:AMPLitude	500 mV
FGEN[:CHANnel[n]]:FREQuency	1.2 MHz
FGEN[:CHANnel[n]]:DCLevel	0 V
FGEN[:CHANnel[n]]:HIGH	250 mV
FGEN[:CHANnel[n]]:LOW	–250 mV
FGEN[:CHANnel[n]]:OFFSet	0 V
FGEN[:CHANnel[n]]:PHASe	0°
FGEN[:CHANnel[n]]:SYMMetry	100%
FGEN[:CHANnel[n]]:TYPE	SINE

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

Command	Default value
FGEN:COUPle:AMPLitude	0 (off)
INSTrument:COUPle:SOURce	0 (off)
INSTrument:MODE	AWG
MMEMory:OPEN:PARameter:NORMalize	NONE
OUTPut:OFF	0 (off)
OUTPut[n][:STATe]	0 (off)
OUTPut[n]:SVALue[:ANALog][:STATe]	ZERO
OUTPut[n]:SVALue:MARKer[1|2]	LOW
OUTPut[n]:WVALue[:ANALog][:STATe]	ZERO
OUTPut[n]:WVALue:MARKer[1|2]	LOW
SLISt:SEQuence:EVENt:JTIMing	END
SLISt:SEQuence:EVENt:PJUMp:ENABle	0 (off)
SLISt:SEQuence:RFLag	1 (on)
SLISt:SEQuence:STEP[n]:EJINput	0 (off)
[SOURce]:FREQuency[:CW][:FIXed]	8 GHz
[SOURce[n]]:DAC:RESolution	10
[SOURce[n]]:MARKer[1|2]:DELay	0 seconds
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate][:AMPLitude]	1 V
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:HIGH	1 V
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:LOW	0 V
[SOURce[n]]:MARKer[1|2]:VOLTage[:LEVel][:IMMediate]:OFFSet	500 mV
[SOURce]:RCCouple	0 (off)
[SOURce[n]]:RMODe	CONTinuous
[SOURce[n]]:SKEW	0 seconds
[SOURce[n]]:TINPut	ATRigger
[SOURce[n]]:VOLTage[:LEVel][:IMMediate][:AMPLitude]	500 mV
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:HIGH	250 mV
[SOURce[n]]:VOLTage[:LEVel][:IMMediate]:LOW	–250 mV
SYNChronize:ENABle	0 (off)
SYSTem:ERRor:DIALog	1 (enabled)
TRIGger[:SEQuence]:IMPedance	50 Ω
TRIGger[:SEQuence]:LEVel	1.4 V
TRIGger[:SEQuence]:MODE	ASYNchronous

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

Command	Default value
TRIGger[:SEQuence]:SLOPe	POSitive.
TRIGger[:SEQuence]:SOURce	EXTernal
TRIGger[:SEQuence]:WVALue	ZERO

Appendix D: Master & slave operation

The AWGSYNC01 Synchronization Hub provides a means of synchronizing multiple AWG70000 series instruments in a complex system. The AWG70000 series instruments can be used as masters or slaves. This appendix lists commands that cannot be modified when used in a master-slave configuration.

Operation with the AWG70000 as a master

When synchronization is enabled and the AWG70000 is the master, the following commands have limited functins. Refer to the individual command for more information.

CLOCk:OUTPut[:STATe]
TRIGger[:SEQuence]:MODE

Operation with the AWG70000 as a slave

When synchronization is enabled and the AWG70000 is a slave, the following commands will not cause any changes and will generate an error message. Refer to the individual command for more information.

*TRG
[SOURce]:FREQuency[:CW]|[:FIXed]
[SOURce]:ROSCillator:MULTiplier
AWGControl:RUN[:IMMediate]
AWGControl:STOP[:IMMediate]
AWGControl[:CLOCk]:DRATe
AWGControl[:CLOCk]:SOURce
CLOCk:ECLock:DIVider
CLOCk:ECLock:FREQuency
CLOCk:ECLock:FREQuency:ADJust
CLOCk:ECLock:FREQuency:DETect
CLOCk:ECLock:MULTiplier
CLOCk:EREFerence:DIVider
CLOCk:EREFerence:FREQuency
CLOCk:EREFerence:FREQuency:DETect
CLOCk:EREFerence:MULTiplier
CLOCk:JITTer
CLOCk:OUTPut[:STATe]
CLOCk:SOURce
CLOCk:SRATe
TRIGger[:SEQuence][:IMMediate]
TRIGger[:SEQuence]:IMPedance
TRIGger[:SEQuence]:INTerval
TRIGger[:SEQuence]:LEVel

TRIGger[:SEQuence]:MODE
TRIGger[:SEQuence]:SLOPe
TRIGger[:SEQuence]:SOURce

Operation with the AWG70000 as a master or slave and system is not idle

When synchronization is enabled and the AWG70000 is either a master or a slave, the following commands will not cause any changes and will generate an error message while the system is playing or not idle. Refer to the individual command for more information.

[SOURce[n]]:CASSet:SEQuence
[SOURce[n]]:CASSet:WAVeform
[SOURce[n]]:DAC:RESolution
[SOURce[n]]:WAVeform
OUTPut[n]:WVALue:MARKer[1|2]
OUTPut[n]:WVALue[:ANALog][:STATe]
SLISt:SEQuence:DELete
SLISt:SEQuence:EVENt:PJUMp:DEFine
SLISt:SEQuence:EVENt:PJUMp:ENABle
SLISt:SEQuence:STEP[n]:EJINput
SLISt:SEQuence:STEP[n]:EJUMp
SLISt:SEQuence:STEP[n]:TFLag[m]:AFLag
SLISt:SEQuence:STEP[n]:TFLag[m]:BFLag
SLISt:SEQuence:STEP[n]:TFLag[m]:CFLag
SLISt:SEQuence:STEP[n]:TFLag[m]:DFLag
SLISt:SEQuence:STEP[n]:GOTO
SLISt:SEQuence:STEP:RCOunt:MAX?
SLISt:SEQuence:STEP[n]:TASSet:SEQuence
SLISt:SEQuence:STEP[n]:TASSet[m]:WAVeform
SLISt:SEQuence:STEP[n]:WINPut
WLISt:WAVeform:DELete
WLISt:WAVeform:NORMalize
WLISt:WAVeform:RESample
WLISt:WAVeform:SHIFt

Index