VB8000

Arbitrary Waveform Generator
Communication Interface

USER'S MANUAL

YOKOGAWA
Yokogawa Electric Corporation

IM 703150-11E
1st Edition

Foreword

Thank you for purchasing the Arbitrary Waveform Generator VB8000.
This Communication Interface User's Manual describes the functions of the GP-IB and serial interfaces and commands. To ensure correct use, please read this manual thoroughly before operation.
Keep this manual in a safe place for quick reference in the event a question arises.
The following three manuals, including this one, are provided as manuals for the VB8000. Read them along with this manual.

Manual Title	Manual No.	Description
VB8000 Arbitrary Waveform Generator User's Manual	IM 703150-01E	Explains all functions and procedures of the VB8000 excluding the communication functions.
VB8000 Arbitrary Waveform Generator Communication Interface User's Manual	IM 703150-11E	This manual. Explains the communication functions of the GP-IB and serial (RS-232) interfaces.
File Conversion Utility Software for VB8000 User's Manual	IM 703150-61E	Explains how to use the software that converts files into data that the VB8000 can use.

Notes

- The contents of this manual are subject to change without prior notice as a result of continuing improvements to the instrument's performance and functions.
- Every effort has been made in the preparation of this manual to ensure the accuracy of its contents. However, should you have any questions or find any errors, please contact your nearest YOKOGAWA dealer as listed on the back cover of this manual.
- Copying or reproducing all or any part of the contents of this manual without YOKOGAWA's permission is strictly prohibited.

Trademarks

- Microsoft, MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe and Acrobat are trademarks of Adobe Systems Incorporated.
- Other company and product names are trademarks or registered trademarks of their respective holders.

Revisions

1st Edition: November 2000

Disk No. CM12
1st Edition : November 2000 (YK)

How to Use This Manual

Structure of the Manual

This User's Manual consists of the following sections:

Chapter 1 GP-IB Interface
Describes the functions and specifications of the GP-IB interface.

Chapter 2 Serial Interface
Describes the functions and specifications of the serial interface.

Chapter 3 Before Programming
Describes the syntax used to transmit commands.

Chapter 4 Commands
Describes each command that is available.

Chapter 5 Status Report
Describes the status byte, various registers, queues, and other information.

Chapter 6 Sample Program
Introduces a sample program written in Quick-BASIC using an IBM-compatible PC (the GP-IB board that is used is AT-GPIB/TNT IEEE-488.2 by National Instruments).

Appendix
Describes reference material such as an ASCII character code table.

Index
Index of contents.

Conventions Used in This Manual

- **Conventions**

Type	Symbol	Meaning
Unit	k K	1000 Example: 100 kHz (clock frequency) 1024 Example: 640 KB (Storage capacity of floppy disks)
Notes	***Note***	Provides important information for the proper operation of the instrument.
Key	[Communication]	Expresses soft keys that appear on the screen.

- **Symbols Used in the Syntax**

The following table indicates symbols that are used in the syntax mainly in chapter 4. These symbols are referred to as BNF (Backus-Naur Form) symbols. For details on the data, see pages 3-5 and 3-6.

Symbol	Meaning	Example	Entry Example
<>	Defined value	OUTPut<x> <x>=1 to 4	→OUTPUT2
{}	Select from values given in {}	CLOCk:INPut{INTernal|EXTernal}?	→CLOCK:INPUT:INTERNAL
|	Exclusive OR		

Contents

Foreword i
How to Use This Manual iii

Chapter 1 GP-IB Interface
1.1 Names and Function of Sections 1-1
1.2 Connecting the GP-IB Cable 1-2
1.3 GP-IB Interface Functions 1-3
1.4 GP-IB Interface Specifications 1-4
1.5 Setting the Address 1-5
1.6 Responses to Interface Messages 1-6

Chapter 2 Serial Interface
2.1 Names and Function of Sections 2-1
2.2 Specifications and Functions of the Serial Interface 2-2
2.3 Connection via Serial Interface 2-3
2.4 Combination of Handshaking Methods 2-5
2.5 Combination of Data Formats 2-7
2.6 Setting Serial Communications 2-8

Chapter 3 Before Programming
3.1 Messages 3-1
3.2 Commands 3-3
3.3 Responses 3-5
3.4 Data 3-5
3.5 Synchronization with the Controller 3-7

Chapter 4 Commands
4.1 A List of Commands 4-1
4.2 CLOCk Group 4-5
4.3 COMMunicate Group 4-7
4.4 FILE Group 4-8
4.5 MISC Group 4-13
4.6 OUTPut Group 4-16
4.7 STARt Group 4-22
4.8 STATus Group 4-23
4.9 STOP Group 4-24
4.10 SYSTem Group 4-25
4.11 Common Command Group 4-26

Chapter 5 Status Report
5.1 Status Reports 5-1
5.2 Status Byte 5-2
5.3 Standard Event Register 5-3
5.4 Extended Event Register 5-4
5.5 Output Queue and Error Queue 5-5

Chapter 6 Sample Program

6.1 Before Programming 6-1
6.2 Setting Waveform Generation Conditions for Each Output 6-2

Appendix

Appendix 1 ASCII Character Codes App-1
Appendix 2 Error Messages App-2
Appendix 3 About the IEEE.488.2-1992 Standard App-5

Index

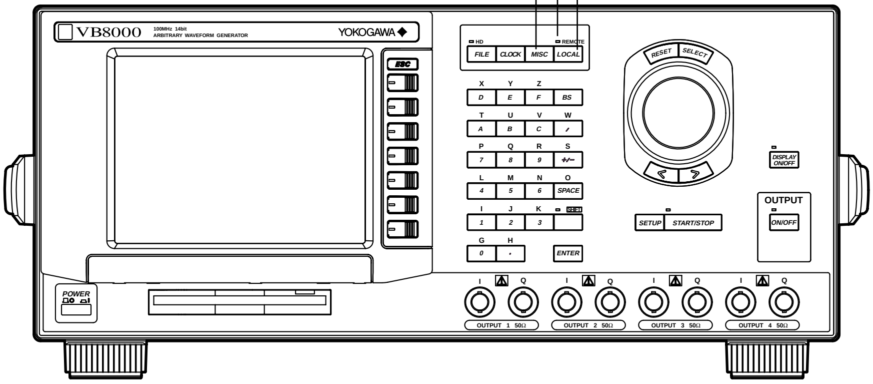
1.1 Names and Function of Sections

Front Panel

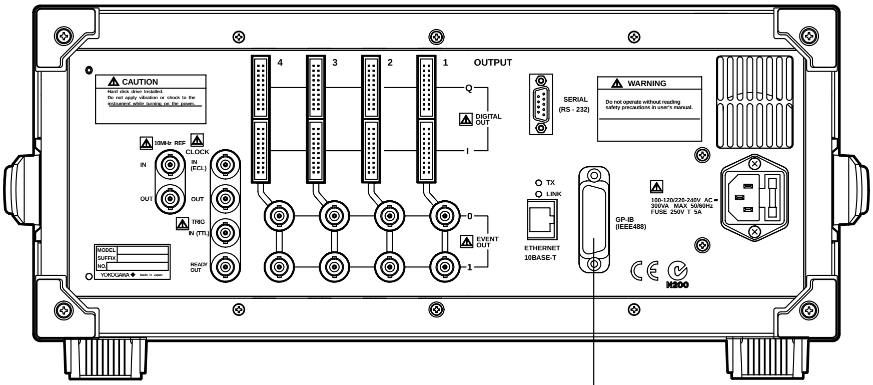
MISC key
Press this key to configure communications.

REMOTE indicator
Turns ON when the VB8000 is in the remote mode through communications.

LOCAL key
Press this key to clear the remote mode through communications and enter the local mode in which key operations are enabled.



Rear Panel


CAUTION
Hard disk drive installed.
Do not apply vibration or shock to the instrument while turning on the power.
OUTPUT
DIGITAL OUT
EVENT OUT
10MHz REF
CLOCK
IN (ECL)
TRIG IN (TTL)
READY OUT
SERIAL (RS - 232)
WARNING
Do not operate without reading safety precautions in user's manual.
TX
LINK
ETHERNET 10BASE-T
GP-IB (IEEE488)
100-120/220-240V AC 300VA MAX 50/60Hz FUSE 250V T 5A
MODEL
SUFFIX
NO.
YOKOGAWA Made in Japan

GP-IB connector
Connector used to connect the VB8000 to the controller (PC) using a GP-IB cable.

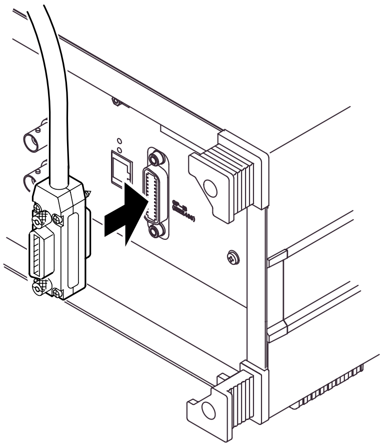
1.2 Connecting the GP-IB Cable

GP-IB Cable

The GP-IB connector used on this instrument is a 24-pin connector that conforms to the IEEE St'd 488-1978. Use a GP-IB cable that conforms to this standard.

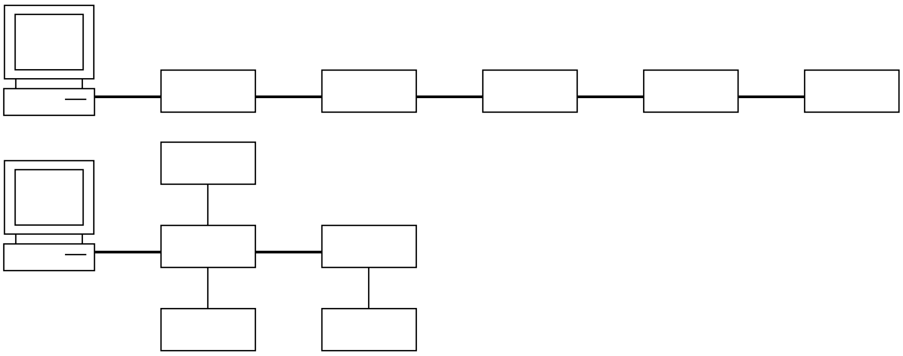
Connection Procedure

Connect the cable as shown below.



Precautions to Be Taken during Connection

- Firmly tighten the screws on the GP-IB cable connector.
- Multiple devices can be connected to a single GP-IB system. However, no more than 15 devices (including the controller) can be connected to a single system.
- When connecting multiple devices, each device must have its own unique address.
- Use a cable of length 2 m or less for connecting the devices.
- Make sure the total cable length does not exceed 20 m.
- When communicating, have at least two-thirds of the devices turned ON.
- When connecting multiple devices, connect them in a star or linear configuration (see the figure below). Loop and parallel configurations are not allowed.



CAUTION

When connecting or disconnecting communication cables, make sure to turn OFF the PC and the VB8000. Otherwise, erroneous operation or damage to the internal circuitry may result.

1.3 GP-IB Interface Functions

GP-IB Interface Functions

Listener function

- All of the information that you can set with the panel keys can be set through the GP-IB interface except for turning ON/OFF the power and setting the communication parameters.
- Receives commands from a controller requesting the output of setup information, waveform data, and other information.
- Also receives status report commands.

Talker function

- Outputs setup information, waveform data, and other information.

Note

Talk-only, listen-only, and controller functions are not available on this instrument.

Switching between Remote and Local Modes

When switching from local to remote mode

Receiving an REN (Remote Enable) message from the controller when the instrument is in the local mode causes the instrument to switch to the remote mode.

- The REMOTE indicator turns ON (see page 1-1).
- All keys other than the LOCAL key are locked.
- The settings that existed in the local mode are maintained even when the instrument switches to the remote mode.

When switching from remote to local mode

Pressing the LOCAL key when the instrument is in the remote mode causes the instrument to switch to the local mode. However, this act is invalid if the instrument has been set to Local Lockout mode (see page 1-6) by the controller.

- The REMOTE indicator turns OFF.
- Key operations are enabled.
- The settings that existed in the remote mode are maintained even when the instrument switches to the local mode.

1.4 GP-IB Interface Specifications

GP-IB Interface Specifications

Electrical and mechanical specifications: Conforms to IEEE St'd 488-1978

Functional specifications: See table below.

Protocol: Conforms to IEEE St'd 488.2-1992

Code: ISO (ASCII) code

Mode: Addressable mode

Address setting: The address can be set in the range from 0 to 30 on the GP-IB setting screen that is played using the MISC key.

Clear remote mode: Clear remote mode by pressing the LOCAL key (except when Local Lockout is enabled by the controller).

Functional specifications

Function	Subset Name	Description
Source handshaking	SH1	Full source handshaking capability
Acceptor handshaking	AH1	Full acceptor handshaking capability
Talker	T6	Basic talker capability, serial polling, untalk on MLA (My Listen Address), and no talk-only capability
Listener	L4	Basic listener capability, unlisten on MTA (My Talk Address), and no listen-only capability.
Service request	SR1	Full service request capability
Remote local	RL1	Full remote/local capability
Parallel polling	PP0	No parallel polling capability
Clear device	DC1	Full device clear capability
Device trigger	DT1	Full device trigger capability
Controller	C0	No controller functions
Electrical characteristic	E1	Open collector

1.5 Setting the Address

Function

Carry out the following settings when using a controller to set information that can be specified through key operation on the VB8000 or when outputting setup data or output waveform data to the controller.

Setting the address

Set the address of the VB8000 within the following range for the addressable mode:
0 to 30
Each device that can be connected via GP-IB has a unique address within the GP-IB system. This address is used to distinguish the device from others. Therefore, when you connect the VB8000 to a PC, for example, make sure to assign a unique address to the VB8000.

Note

Do not change the address while the controller or other devices are using the GP-IB system.

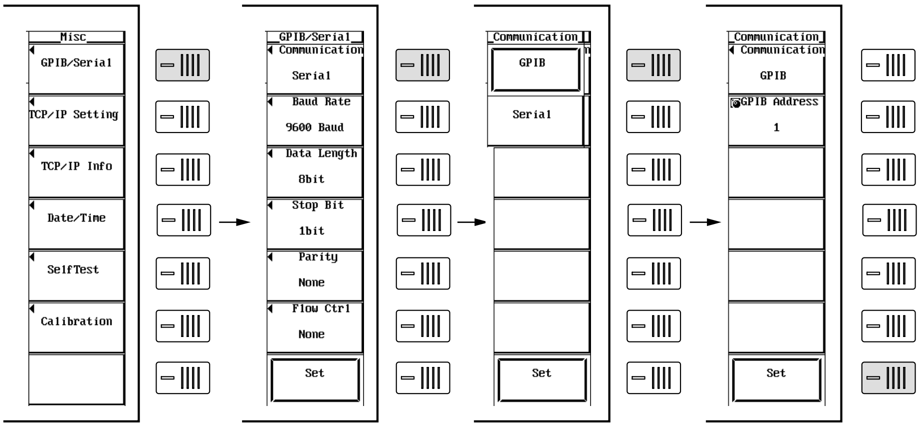
Procedure

Displaying the GP-IB menu

1. Press MISC.
2. Press the [GPIB/Serial] soft key.
3. Press the [Communication] soft key to display the Communication Interface Selection menu.
4. Press the [GPIB] soft key.

Setting the address

5. Use the rotary knob or numerical keys to set [GPIB Address].
6. Press the [Set] soft key to confirm the GP-IB address.



1.6 Responses to Interface Messages

Responses to Interface Messages

Responses to a uni-line message

- **IFC (Interface Clear)**
  Clears the talker and listener functions. Stops output if data are being output.
- **REN (Remote Enable)**
  Switches between the remote and local modes.

  IDY (Identify) is not supported.

Responses to a multi-line message (address command)

- **GTL (Go To Local)**
  Switches to the local mode.
- **SDC (Selected Device Clear)**
  - Clears the program message (command) being received and the output queue (see page 5-5).
  - `*OPC` and `*OPC?` commands in execution are void.
  - The `*WAI` and `COMMunicate:WAIT` commands are immediately terminated.
- **GET (Group Execute Trigger)**
  Same operation as the `*TRG` command.

  PPC (Parallel Poll Configure) and TCT (Take Control) are not supported.

Responses to a multi-line message (universal command)

- **LLO (Local Lockout)**
  Disables the LOCAL key on the front panel to prohibit switching to the local mode.
- **DCL (Device Clear)**
  Same operation as the SDC message.
- **SPE (Serial Poll Enable)**
  Sets the talker function on all devices on the bus to serial polling mode. The controller polls the devices in order.
- **SPD (Serial Poll Disable)**
  Clears the serial polling mode of the talker function on all devices on the bus.

  PPU (Parallel Poll Unconfigure) is not supported.

What Is an Interface Message

Interface messages are also referred to as interface commands or bus commands. They are commands that are issued by the controller. They are classified as follows:

Uni-line messages

A single control line is used to transmit uni-line messages. The following three types of messages are available:

- IFC (Interface Clear)
- REN (Remote Enable)
- IDY (Identify)

Multi-line messages

Eight data lines are used to transmit multi-line messages. The messages are classified as follows:

- **Address command**

  These commands are valid when the instrument is designated as a listener or as a talker. The following five commands are available:

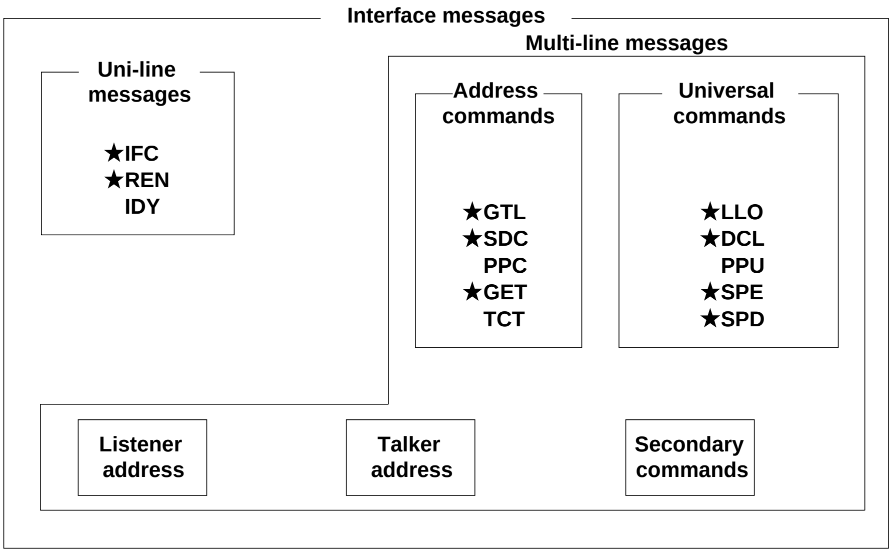
  Commands that are valid on an instrument that is designated as a listener
  - GTL (Go To Local)
  - SDC (Selected Device Clear)
  - PPC (Parallel Poll Configure)
  - GET (Group Execute Trigger)

  Commands that are valid on an instrument that is designated as a talker
  - TCT (Take Control)

- **Universal command**

  These commands are valid on all instruments regardless of the listener and talker designations. The following five commands are available:
  - LLO (Local Lockout)
  - DCL (Device Clear)
  - PPU (Parallel Poll Unconfigure)
  - SPE (Serial Poll Enable)
  - SPD (Serial Poll Disable)

  In addition, listener address, talker address, and secondary commands are also considered interface messages.



Interface messages that VB8000 supports are indicated with Åö marks.

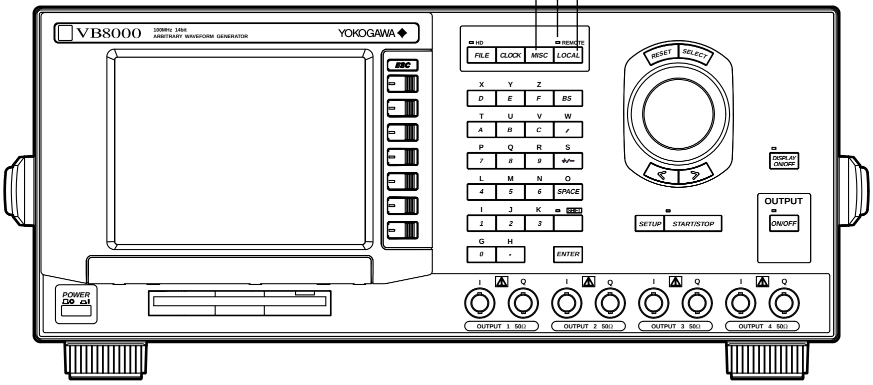
Note

The differences between SDC and DCL

In multi-line messages, SDC messages are those that require talker or listener designation and DCL messages are those that do not require the designation. Therefore, SDC messages are directed at a particular instrument while DCL messages are directed at all instruments on the bus.

2.1 Names and Function of Sections

Front Panel

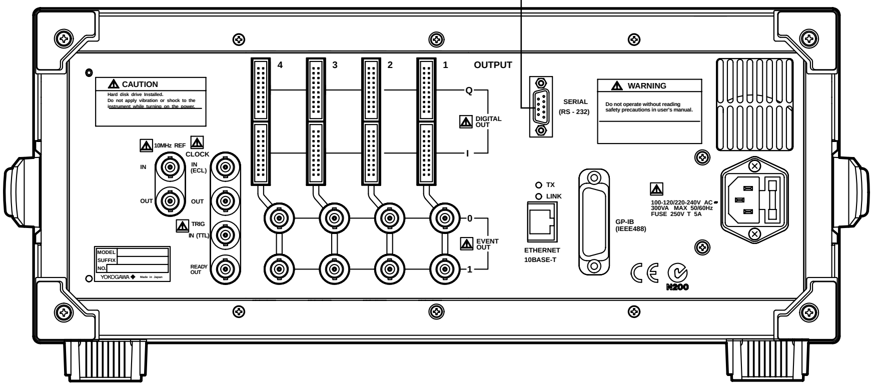


MISC key
Press this key to configure communications.

REMOTE indicator
Turns ON when the VB8000 is in the remote mode (controlled via communications).

LOCAL key
Press this key to clear the remote mode (controlled via communications) and enter the local mode in which key operations are enabled.

Rear Panel


CAUTION
Hard disk drive installed.
Do not apply vibration or shock to the instrument while turning on the power.
OUTPUT
4
3
2
1
Q
I
DIGITAL OUT
EVENT OUT
SERIAL (RS - 232)
WARNING
Do not operate without reading safety precautions in user's manual.
10MHz REF
CLOCK
IN
OUT
IN (ECL)
TRIG IN (TTL)
READY OUT
TX
LINK
ETHERNET 10BASE-T
GP-IB (IEEE488)
100-120/220-240V AC 300VA MAX 50/60Hz FUSE 250V T 5A
MODEL
SUFFIX
NO.
YOKOGAWA Made in Japan
N200

Serial interface connector (EIA-574 (former 9-pin EIA-232 (RS-232))
Connector used to connect the VB8000 to the controller (PC) using an interface cable. For the connection procedure, see section 2.3.

2.2 Specifications and Functions of the Serial Interface

Reception Functions

You can specify the same settings as those specified by front panel key operations.
Receives output requests for waveform data, setup information, and error codes.

Transmission Functions

Able to output waveform data.
Able to output setup information and the status byte.
Able to output error codes that are generated.

Serial Interface Specifications

Electrical characteristics:	Conforms to EIA-574 (9-pin EIA-232 (RS-232)
Connection method:	Point-to-point
Transmission mode:	Full-duplex
Synchronization:	Start-stop synchronization
Baud rate:	1200, 2400, 4800, 9600, 19200, and 38400
Start bit:	Fixed to 1 bit
Data length:	7 or 8 bits
Parity:	Even, odd, or no parity
Stop bit:	1 or 2 bits
Connector:	DELC-J9PAF-13L6 (JAE or equivalent)
Hardware handshaking:	Select whether to fix the CA and CB signals to TRUE or use the signals for flow control.
Software handshaking:	Select whether to use the X-ON and X-OFF signals to control the transmission data or both transmission and reception data. X-ON (ASCII 11H) X-OFF (ASCII 13H)
Received buffer length:	256 bytes

Switching between Remote and Local Modes

When switching from local to remote mode

If the VB8000 receives a "`:COMMunicate:REMote ON`" command from the PC when it is in the local mode, it switches to the remote mode.

- The REMOTE indicator turns ON.
- All keys except the LOCAL key are disabled.
- The settings that existed in the local mode are maintained even when the VB8000 switches to the remote mode.

When switching from remote to local mode

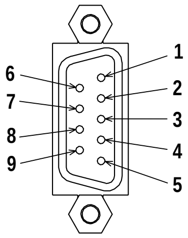
Pressing the LOCAL key when the VB8000 is in the remote mode causes the instrument to switch to the local mode. However, this is void when the VB8000 has received a "`:COMMunicate:LOCKout ON`" command from the PC (local lockout condition). When the VB8000 receives a "`:COMMunicate:REMote OFF`" command from the PC, the VB8000 switches to the local mode regardless of the local lockout condition.

- The REMOTE indicator turns OFF.
- Key operations are enabled.
- The settings that existed in the remote mode are maintained even when the instrument switches to the local mode.

2.3 Connection via Serial Interface

When you connect the VB8000 to a PC, you must set the VB8000 so that the handshaking method, data transfer rate, data format, etc. match those on the PC side. For details on the settings, see the following pages. In addition, use an interface cable that meets the specifications of the VB8000.

Connector and Signal Names



DELC-J9PAF-13L6 or equivalent

2 RD (Received Data): Received data from the PC.
Signal direction ..... input

3 SD (Send Data): Transmitted data to the PC.
Signal direction ..... output

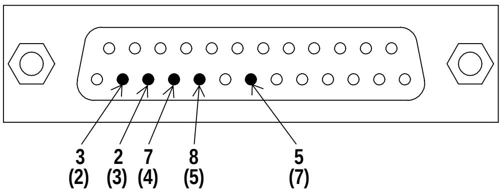
5 SG (Signal Ground): Signal ground.

7 RS (Request to Send): Handshaking used to receive data from the PC.
Signal direction ..... output

8 CS(Clear to Send): Handshaking used to send data to the PC.
Signal direction ..... input

* Pins 1, 4, 6, and 9 are not used.

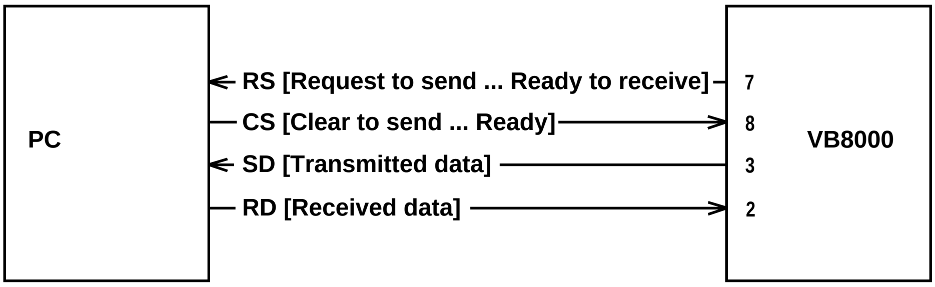
9-pin to 25-pin Adapter and Signal Names



The numbers inside the parentheses are pin numbers for the 25-pin connector.

Signal Direction

The figure below shows the direction of the signals used by the serial interface of the VB8000.


PC
RS [Request to send ... Ready to receive]
CS [Clear to send ... Ready]
SD [Transmitted data]
RD [Received data]
7
8
3
2
VB8000

RS-232 Standard Signals and Their JIS and CCITT Abbreviations

Signal Table

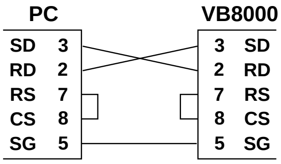
Pin number (9-pin connector)	Symbol			Name
	RS-232	CCITT	JIS	
5	AB (GND)	102	SG	Signal ground
3	BA (TXD)	103	SD	Transmitted data
2	BB (RXD)	104	RD	Received data
7	CA (RTS)	105	RS	Request to send
8	CB (CTS)	106	CS	Clear to send

Signal Wiring Example

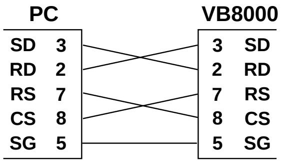
The pin numbers are for the 9-pin connector.

In general, use a cross cable.

- OFF-OFF/XON-XON



- Hard(CS-RS)



2.4 Combination of Handshaking Methods

When using the serial interface for transferring data, it is necessary for equipment on both sides to agree on a set of rules to ensure the proper transfer of data. The set of rules is called handshaking. Because there are various handshaking methods that can be used between the VB8000 and the PC, one must make sure that the same method is chosen by both the VB8000 and the PC.

You can choose any of the four methods in the table below.

Table of Handshaking Methods (○...indicates that it is supported)

Handshaking Method		Data Transmission Control (Control used to send data to a PC)			Data Reception Control (Control used to receive data from a PC)		
		Software handshaking	Hardware handshaking	No handshaking	Software handshaking	Hardware handshaking	No handshaking
		Stops transmission when X-OFF is received. Resume when X-ON is received.	Stops transmission when CB (CTS) is false. Resume when it is true.		Send X-OFF when the received data buffer is 3/4th filled. Send X-ON when the received data buffer becomes 1/4th filled.	Set CA (RTS) to False when the received data buffer is 3/4th filled. Set to True when the received data buffer becomes 1/4th filled.	
	VB8000 menu						
OFF-OFF	None			○			○
XON-XON	Xon	○			○		
CS-RS	Hard		○			○	

OFF-OFF

Data transmission control

There is no handshaking between the VB8000 and the PC. The “X-OFF” and “X-ON” signals are treated as data, and the CS signal is ignored.

Data reception control

There is no handshaking between the VB8000 and the PC. When the received buffer becomes full, all overflow data are discarded.

RS = True (fixed).

XON-XON

Data transmission control

Software handshaking is performed between the VB8000 and the PC. When an “X-OFF” code is received while sending data to the PC, the instrument stops the data transmission. When it receives the next “X-ON” code, it resumes the data transmission. The CS signal received from the PC is ignored.

Data reception control

Software handshaking is performed between the VB8000 and the PC. When the free area of the receive buffer decreases to 64 bytes, the VB8000 sends an “X-OFF” code. When the free area increases to 192 bytes, it sends an “X-ON” code.

RS = True (fixed).

CS-RS

Data transmission control

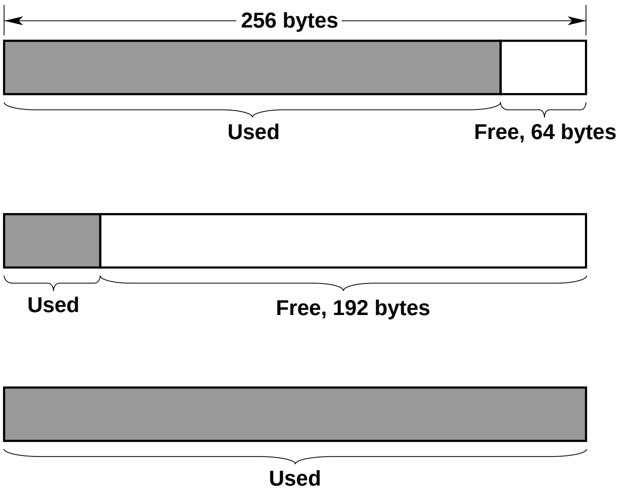
Hardware handshaking is performed between the VB8000 and the PC. When the CS signal becomes False while sending data to the PC, the instrument stops the data transmission. When the CS signal becomes True, it resumes the data transmission. The “X-OFF” and “X-ON” signals are treated as data.

Data reception control

Hardware handshaking is performed between the VB8000 and the PC. When the free area of the receive buffer decreases to 64 bytes, the instrument sets “RS=False.” When the free area increases to 192 bytes, it sets “RS=True.”

Precautions Regarding Data Receiving Control

When handshaking is used to control the reception of data, data may still be sent from the PC even if the free space in the receive buffer drops below 64 bytes. In this case, after the receive buffer becomes full, the excess data will be lost, whether or not handshaking is in effect. Data storage of data resumes when there is free space in the buffer.



When handshaking is used, data reception will stop when the free space in the buffer drops to 64 bytes due to the inability to keep up with the data transfer.

After data reception stops, data continue to be passed to the internal program. When the free space in the buffer increases to 192 bytes, data reception resumes.

If the buffer becomes full, data that overflow are discarded regardless of the handshaking.

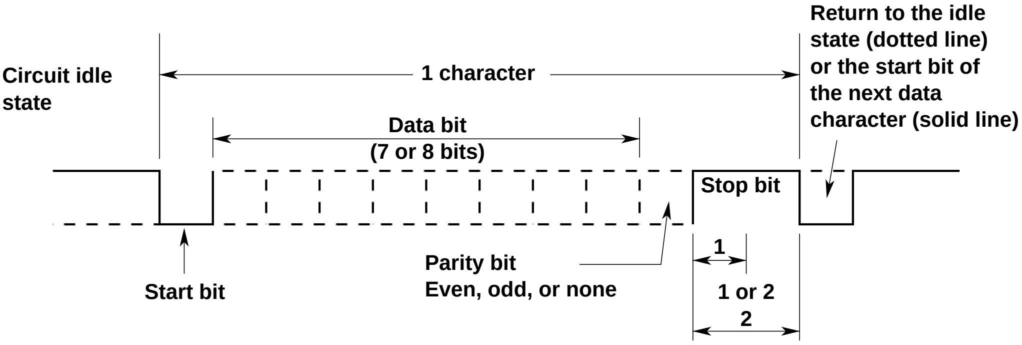
Data Receiving Control through Handshaking

Note

The PC program must be designed so that the received buffers of both the VB8000 and the PC do not become full.

2.5 Combination of Data Formats

The serial interface of the VB8000 performs communications using start-stop synchronization. In start-stop synchronization, characters are transmitted one at a time. Each character consists of a start bit, data bits, a parity bit, and a stop bit (see the figure below).



2.6 Setting Serial Communications

Function

Carry out the following settings when using a controller to set information that can be specified through key operation on the VB8000 or when outputting setup data or output waveform data to the controller.

Selecting the baud rate

Select the baud rate from the following:
1200, 2400, 4800, 9600, 19200, and 38400

Selecting the data length

Select the data length from the following:
7 bit or 8 bit

Selecting the stop bit

Select the stop bit from the following:
1 bit or 2 bit

Selecting the parity bit

Select the parity bit from the following:
None, Odd, or Even

Selecting the handshaking method

Select the transmit data control and receive data control from the following:
Non, Xon, or Hard

Note

The terminator that is used when the VB8000 transmits data is fixed to [CR+LF].

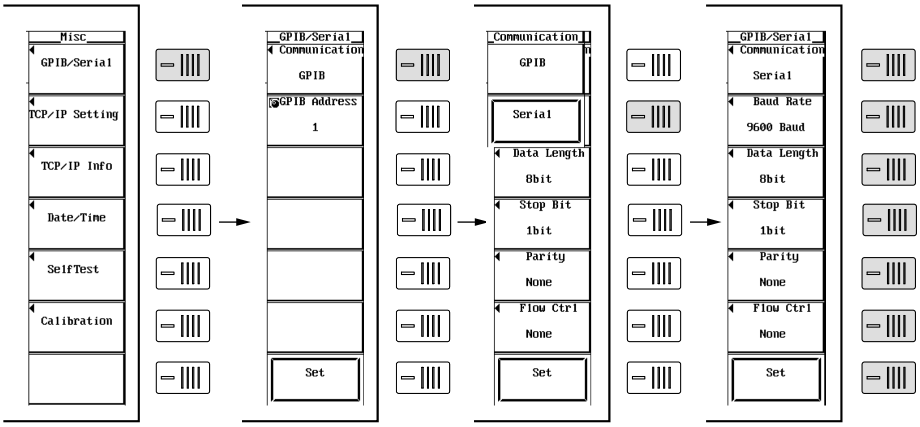
Procedure

Displaying the serial communication (RS-232) menu

1. Press MISC.
2. Press the [GPIB/Serial] soft key.
3. Press the [Communication] soft key to display the Communication Interface Selection menu.
4. Press the [Serial] soft key.

Selecting the baud rate, data length, etc.

5. Press each of the [Baud Rate], [Data Length], [Stop Bit], [Parity], and [Flow Ctrl] (handshaking method) soft keys and select each item.
6. Press the [Set] soft key to confirm the setting.



Chapter 3 Before Programming

3.1 Messages

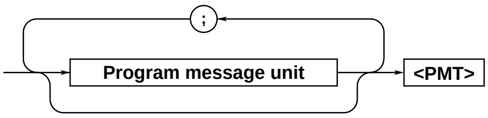
Messages

Messages are used to exchange information between the controller and the instrument. Messages that are sent from the controller to the instrument are called program messages and messages that are sent back from the instrument to the controller are called response messages.

If a program message contains a command that requests a response (a query), the instrument returns a response message upon receiving the program message. A single response message is always returned in response to a single program message.

Program Messages

The program message format is shown below.



<Program Message Unit>

A program message consists of one or more program message units; each unit corresponds to one command. The instrument executes the received commands in order.

Each program message unit is separated by a semicolon (;).

For details regarding the format of the program message unit, see the next section.

Example

$\underbrace{\texttt{:CLOCk:REFerence INTernal}}_{\text{Unit}}\texttt{;}\underbrace{\texttt{TRIGger EXTernal}}_{\text{Unit}}\texttt{<PMT>}$

<PMT>

PMT is a program message terminator. The following three types of terminators are available:

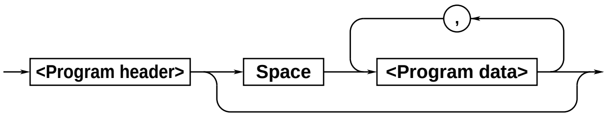
NL (New Line) : Same as LF (Line Feed). ASCII code "0AH"

^END : END message that is defined in IEEE488.1 (EOI signal)
(The data byte that is sent simultaneously with the END message is the last data of the program message.)

NL^END : NL with an END message added
(NL is not included in the program message.)

- **Program Message Unit Format**

  The program message unit format is shown below.



<Program Header>

The program header indicates the command type. For details, see page 3-3.

<Program Data>

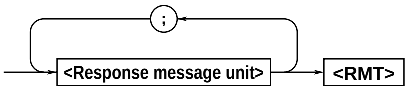
If certain conditions are required in executing a command, program data are added. A space (ASCII code "20H") separates the program data from the header. If there are multiple sets of program data, they are separated by commas (,).

For details, see page 3-5.

Example $\underbrace{\texttt{:CLOCk:REFerence}}_{\text{Header}}\ \underbrace{\texttt{INTernal}}_{\text{Data}}\texttt{<PMT>}$

Response Messages

The response message format is shown below.



<Response Message Unit>

A response message consists of one or more response message units; each response message unit corresponds to one response.

Response message units are separated by a semicolon (;).

For details regarding the format of the response message unit, see the next section.

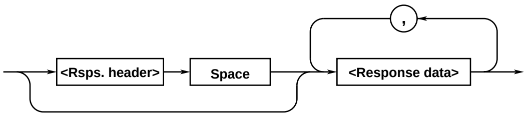
Example

$\underbrace{\texttt{:CLOCk:REFerence INTernal}}_{\text{Unit}}\texttt{;}\underbrace{\texttt{TRIGger EXTernal}}_{\text{Unit}}\texttt{<RMT>}$

<RMT>

<RMT> is a response message terminator. It is NL^END.

- **Response Message Unit Format**
  The response message unit format is shown below.



<Response Header>
A response header sometimes precedes the response data. A space separates the data from the header. For details, see page 3-4.

<Response Data>
Response data contain the content of the response. If there are multiple sets of response data, they are separated by commas (,). For details, see page 3-5.

Example

```
100.00E-03<RMT>    :CLOCK:INPUT INTERNAL<RMT>
     Data              Header      Data
```

If there are multiple queries in a program message, responses are made in the same order as the queries. In most cases, a single query returns a single response message unit, but there are a few queries that return multiple units. The first response message unit always corresponds to the first query, but the nth response unit may not necessarily correspond to the nth query. Therefore, if you want to make sure that every response is retrieved, divide the program messages into individual messages.

Precautions to Be Taken when Transferring Messages

- If a program message that does not contain a query is sent, the next program message can be sent at any time.
- If a program message that contains a query is sent, a response message must be received before the next program message can be sent. If the next program message is sent before the response message is received in its entirety, an error occurs. The response message that was not received is discarded.
- If the controller tries to receive a response message when there is none, an error occurs. If the controller tries to receive a response message before the transmission of the program message is complete, an error occurs.
- If a program message containing multiple message units is sent, and the message contains incomplete units, the instrument will attempt to execute the ones that are believed to be complete. However, these attempts may not always be successful. In addition, if the message contains queries, the responses may not be returned.

Deadlock

The instrument can store in its buffer program and response messages of length 1024 bytes or more (The number of available bytes varies depending on the operating conditions). When both the transmit and receive buffers become full at the same time, the instrument can no longer continue its communication operation. This state is called a deadlock. In this case, operation can be resumed by discarding the program message.
Deadlock will not occur if the program message (including the <PMT>) is kept below 1024 bytes. Furthermore, deadlock never occurs if a program message does not contain a query.

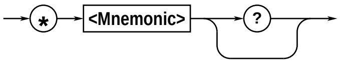
3.2 Commands

Commands

There are three types of commands (program headers) that are sent from the controller to the instrument. They differ in their program header formats.

Common Command Header

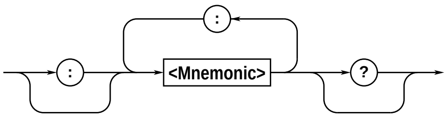
Commands that are defined in the IEEE 488.2-1992 are called common commands. The header format of a common command is shown below. An asterisk (*) is always placed in the beginning of a command.



An example of a common command: `*CLS`

Compound Header

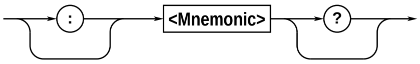
Dedicated commands used by the instrument are classified and arranged in a hierarchy according to their functions. The format of a compound header is shown below. A colon (`:`) must be used to specify a lower hierarchy.



An example of a compound header: `MISC:TCPIp`

Simple Header

These commands are functionally independent and do not have a hierarchy. The format of a simple header is shown below.



An example of a simple header: `STARt`

Note

A <mnemonic> is a character string made up of alphanumeric characters.

When Concatenating Commands

Command Group

A command group is a group of commands that have common compound headers arranged in a hierarchy. A command group may contain sub-groups.

Example Group of commands related to the clock

```
:CLOCk?
:CLOCk:FREQuency
:CLOCk:REFerence
:CLOCk:INPut
:CLOCk:TRIGger
```

When Concatenating Commands of the Same Group

The instrument stores the hierarchical level of the command that is currently being executed, and performs analysis on the assumption that the next command sent will also belong to the same level. Therefore, common header sections can be omitted for commands belonging to the same group.

Example `:CLOCk:INPut EXTernal;`
`TRIGger INTernal<PMT>`

When Concatenating Commands of Different Groups

If the following command does not belong to the same group, a colon (`:`) is placed in front of the header.

Example `:CLOCk:INPut EXTernal;:SYSTem:`
`LCD ON<PMT>`

When Concatenating Simple Headers

If a simple header follows another command, a colon (`:`) is placed in front of the simple header.

Example `:CLOCk:INPut EXTernal;:STARt<PMT>`

When Concatenating Common Commands

Common commands that are defined in the IEEE 488.2-1992 are independent of hierarchy. Colons (`:`) are not needed before a common command.

Example `:CLOCk:INPut EXTernal;*CLS;`
`TRIGger INTernal<PMT>`

When Separating Commands with <PMT>

If a terminator is used to separate two commands, each command is a separate message. Therefore, the common header must be specified for each command even when commands belonging to the same command group are being concatenated.

Example `:CLOCk:INPut EXTernal<PMT>:CLOCk:`
`TRIGger INTernal<PMT>`

Upper-level Query

An upper-level query is a query in which a question mark (?) is appended to the highest level command of a group. Execution of an upper-level query allows all settings that can be specified in the group to be received at once. Some query groups which are comprised of more than three hierarchical levels can output all the lower level settings.

Example `:SYSTEM?<PMT> → :SYSTEM:OUTPUT ON; LCD ON`

The response to an upper-level query can be transmitted as a program message back to the instrument. In this way, the settings that existed when the upper-level query was made can be restored. However, some upper-level queries will not return setup information that is not currently in use. It is important to remember that not all the group's information is necessarily returned as part of a response.

Header Interpretation Rules

The instrument interprets the header that is received according to the rules below.

- Upper-case and lower-case letters of a mnemonic are treated the same.
  Example "SYSTem" can also be written as "system" or "SYSTem"
- The lower-case section of the header can be omitted.
  Example "SYSTem" can also be written as "SYSTEM" or "SYST"
- The question mark (?) at the end of a header indicates that it is a query. The question mark (?) cannot be omitted.
  Example The shortest abbreviation for "SYSTem?" is "SYST?"
- If the <x> (value) at the end of a mnemonic is omitted, it is interpreted as a 1.
  Example If "OUTPut<x>" is written as "OUTP," it means "OUTPut1."

3.3 Responses

When the controller sends a message unit that has a question mark (?) in its program header (query), the instrument returns a response message to the query. A response message is returned in one of the following two forms.

- Response consisting of a header and data
  If the response can be used as a program message without any change, it is returned with a command header attached.
  Example `:OUTPUT1:FILTER?<PMT>→:OUTPUT1: FILTER F30MHZ<RMT>`

- Response consisting of data only
  If the response cannot be used as a program message unless changes are made to it (query-only command), only the data section is returned. However, there are query-only commands that return responses with the header attached.
  Example `:STATus:ERRor?<PMT>→ 0,"No error"<RMT>`

When You Wish to Return a Response without a Header

Responses that return both header and data can be set so that only the data section is returned. Use the "COMMunicate:HEADer" command for this task.

Abbreviated Form

The response header is normally returned with the lower-case section removed. You can change this so that the response header is in the full form. Use the "COMMunicate:VERBose" command for this task.

3.4 Data

Data

Data contain conditions and values that are written after the header. A space is used to separate the header and data. Data are classified as follows:

Data	Description
<Decimal>	Value expressed as a decimal number (Example: Delay time of waveform generation →`OUTPut1:DELay 100`)
<Voltage>< Frequency>	Physical value (Example: Clock frequency→`CLOCk:FREQuency`)
<Register>	Register value expressed as either binary, octal, decimal or hexadecimal. (Example: Extended event register value →`STATus:EESE #HFE`)
<Character data>	Predefined character string (mnemonic). Selectable from { } (Example: Filter selection →`OUTPut1:FILTer {F90Mhz|F30Mhz|F6Mhz}`)
<Boolean>	Indicates ON and OFF. Use "`ON`," "`OFF`," or a value. (Example: Turn ON remote control →`COMMunicate:REMote ON`)
<Character string data>	Arbitrary character string (Example: Waveform label of OUTPUT →`OUTPut1:LABel "WAVE 1"`)
<Block data>	Data containing 8-bit arbitrary values (Example: Response to acquired waveform data→`FILE:UPLoad:TRANs#800004101......`)

<Decimal>

<Decimal> indicates a value expressed as a decimal number, as shown in the table below. Decimal values are given in the NR form as specified in the ANSI X3.42-1975.

Symbol	Description	Example
<NR1>	Integer	`125  -1  +1000`
<NR2>	Fixed point number	`125.0  -.90  +001.`
<NR3>	Floating point number	`125.0E+0  -9E-1  +.1E4`
<NRf>	Any of the forms <NR1> to <NR3> is allowed.	

- The instrument can receive decimal values that are sent from the controller in any of the forms, <NR1> to <NR3>. This is represented by <NRf>.
- For response messages that the instrument returns to the controller, a specific form <NR1> to <NR3> is defined for each query. The same form is used regardless of the size of the value.
- For the <NR3> format, the "+" sign after the "E" can be omitted. However, the "-" sign cannot be omitted.
- If a value outside the setting range is entered, the value will be changed to the closest value inside the range.
- If a value has more significant digits than the available resolution, the value is rounded.

<Voltage> and <Frequency>

<Voltage> and <Frequency> indicate decimal values that have physical dimensions. <Multiplier> or <Unit> can be attached to the <NRf> format that was described earlier. They are expressed in one of the following forms:

Form	Example
<NRf><Multiplier><Unit>	5MV
<NRf><Unit>	5E-3V
<NRf><Multiplier>	5M
<NRf>	5E-3

- **<Multiplier>**

  <Multipliers> given in the following table can be used:

Symbol	Word	Multiplier
EX	Exa	10^{18}
PE	Peta	10^{15}
T	Tera	10^{12}
G	Giga	10^{9}
MA	Mega	10^{6}
K	Kilo	10^{3}
M	Milli	10^{-3}
U	Micro	10^{-6}
N	Nano	10^{-9}
P	Pico	10^{-12}
F	Femto	10^{-15}

- **<Unit>**

  <Unit> given in the following table can be used:

Symbol	Word	Description
V	Volt	Voltage
HZ	Hertz	Frequency
MHZ	Megahertz	Frequency

- <Multiplier> and <Unit> are not case sensitive.
- "U" is used to indicate the micro "µ ."
- "MA" is used for Mega to distinguish it from Milli. The only exception is Megahertz which is expressed as "MHZ." Therefore, the "M (Milli)" multiplier cannot be used for frequencies.
- If both <Multiplier> and <Unit> are omitted, the default unit is used.
- Response messages are always in the <NR3> form. Response messages are returned using the default unit without the <Multiplier> or <Unit>.

<Register>

<Register> indicates an integer that can be expressed not only in <Decimal> notation, but also <Hexadecimal>, <Octal>, or <Binary>. <Register> is used when each bit of the value has a particular meaning. They are expressed in one of the following forms:

Form	Example
<NRf>	1
#H<Hexadecimal value made up of the digits 0 to 9 and A to F>	#H0F
#Q<Octal value made up of the digits 0 to 7>	#Q777
#B<Binary value made up of the digits 0 and 1>	#B001100

- <Register> is not case sensitive.
- Response messages are always returned in the <NR1> form.

<Character data>

<Character Data> are predefined character strings (mnemonic). They are mainly used to indicate options. One of the character strings given in brackets {} is chosen. The data interpretation is the same as the description given in “Header Interpretation Rules” on page 3-4.

Form	Example
{INTernal|EXTernal}	INTERNAL

- As with the header, the "COMMunicate:VERBose" command can be used to select whether to return the response in the full form or in the abbreviated form.
- The "COMMunicate:HEADer" setting does not affect the <character data>.

<Boolean>

<Boolean> are data that indicate ON or OFF. They are expressed in one of the following forms:

Form	Example
{ON|OFF|<NRf>}	ON OFF 1 0

- When <Boolean> is expressed in the <NRf> form, "OFF" is selected if the rounded integer value is "0," and ON for all other cases.
- A response message is always returned with a "1" if the value is ON and "0" if the value is OFF.

<Character string data>

Unlike the predefined character strings of <Character data>, <Character string data> is an arbitrary character string. The character string is enclosed in single quotation marks (') or double quotation marks (").

Form	Example
<Character string data>	'ABC' "IEEE488.2-1987"

- If the character string contains a double quotation mark ("), it is represented by (""). This rule also applies to a single quotation mark (').
- A response message is always enclosed in double quotation marks (").
- Because <Character string data> is an arbitrary character string, if the last single quotation mark (') or double quotation mark (") is missing, the instrument may assume that the remaining program message units are part of the <Character string data> and may not detect the error.

<Block data>

<Block data> are data containing 8-bit arbitrary values. They are only used in response messages on the VB8000. The syntax is as follows:

Form	Example
#N<N-digit decimal number> <Series of data bytes>	#800000010ABCDEFGHIJ

- #N
Indicates that the data are <Block data>. "N" indicates the number of succeeding data bytes (digits) in ASCII code characters.
- <N-digit decimal number>
Indicates the number of bytes of data (example: 00000010 = 10 bytes).
- <Series of data bytes>
Expresses the actual data (example: ABCDEFGHIJ).
- Data are 8-bit values (0 to 255) that are allowed. Therefore, care must be taken on the controller side, because ASCII code "0AH" that indicates "NL" can be a part of the data.

3.5 Synchronization with the Controller

The VB8000 does not support overlap commands, which allows the execution of the next command to start before the execution of the previous command is completed. If multiple sequential commands are sent consecutively, the execution of the next command is held until the execution of the previous command is completed.

Chapter 4 Commands

4.1 A List of Commands

Command	Functions	Page
CLOCk Group		
`:CLOCk?`	Queries all settings related to the clock and trigger.	4-5
`:CLOCk:FREQuency`	Sets the clock frequency or queries the current setting.	4-5
`:CLOCk:INPut`	Selects internal or external clock input or queries the current setting.	4-5
`:CLOCk:REFerence`	Selects internal or external 10-MHz reference signal or queries the current setting.	4-5
`:CLOCk:TRIGger`	Selects internal or external start trigger or queries the current setting.	4-5
COMMunicate Group		
`:COMMunicate?`	Queries all settings related to communications.	4-6
`:COMMunicate:HEADer`	Sets whether or not to attach a header to the response data or queries the current setting.	4-6
`:COMMunicate:LOCKout`	Sets or clears local lockout.	4-6
`:COMMunicate:REMote`	Switch between remote and local modes.	4-7
`:COMMunicate:STATus?`	Queries line-specific status.	4-7
`:COMMunicate:VERBose`	Sets whether to use abbreviated or unabbreviated form for response data.	4-7
`:COMMunicate:WAIT`	Waits for an extended event to occur.	4-7
`:COMMunicate:WAIT?`	Creates a response for the specified extended event.	4-7
File Group		
`:FILE?`	Queries all settings related to files.	4-9
`:FILE:DELete?`	Queries all settings related to the deletion of files.	4-9
`:FILE:DELete:EXECute`	Executes deleting of files.	4-9
`:FILE:DELete:NAME`	Sets the files to be deleted or queries the current setting.	4-9
`:FILE:EXPort?`	Queries all settings related to exporting.	4-9
`:FILE:EXPort:DSTName`	Sets the export destination file name or queries the current setting.	4-10
`:FILE:EXPort:EXECute`	Executes the export operation.	4-10
`:FILE:EXPort:SRCName`	Sets the name of the file to be exported or queries the current setting.	4-10
`:FILE:EXPort:TYPE`	Sets the type of file to be exported or queries the current setting.	4-10
`:FILE:FDFormat?`	Queries all settings related to the format of floppy disks.	4-10
`:FILE:FDFormat:EXECute`	Executes the floppy disk format.	4-10
`:FILE:FDFormat:TYPE`	Sets the format type of the floppy disk or queries the current setting.	4-10
`:FILE:IMPort?`	Queries all settings related to importing.	4-10
`:FILE:IMPort:DSTName`	Sets the import destination file name or queries the current setting.	4-10
`:FILE:IMPort:EXECute`	Executes the import operation.	4-11
`:FILE:IMPort:SRCName`	Sets the name of the file to be imported or queries the current setting.	4-11
`:FILE:IMPort:TYPE`	Sets the type of file to be imported or queries the current setting.	4-11
`:FILE:LOAD?`	Queries all settings related to the loading of settings of each OUTPUT or all OUTPUTs.	4-11
`:FILE:LOAD:EXECute`	Executes the loading of settings of each OUTPUT or all OUTPUTs.	4-11
`:FILE:LOAD:NAME`	Sets the name of the file to be loaded or queries the current setting.	4-11
`:FILE:LOAD:TYPE`	Sets the load destination OUTPUT or all OUTPUTs or queries the current setting.	4-11
`:FILE:MKDir?`	Queries all settings related to directory creation.	4-11
`:FILE:MKDir:EXECute`	Creates the directory.	4-11
`:FILE:MKDir:NAME`	Sets the name of the directory to be created or queries the current setting.	4-11
`:FILE:SAVE?`	Queries all settings related to the saving of settings of each OUTPUT or all OUTPUTs.	4-12
`:FILE:SAVE:EXECute`	Executes the saving of settings of each OUTPUT or all OUTPUTs.	4-12
`:FILE:SAVE:NAME`	Sets the save destination file name or queries the current setting.	4-12
`:FILE:SAVE:TYPE`	Sets the OUTPUT or all OUTPUTs to be saved or queries the current setting.	4-12
`:FILE:UPLoad:NAME`	Sets the name of the file of the data transfer destination or queries the current setting.	4-12

Command	Functions	Page
:FILE:UPLoad:TRANs	Executes data transfer via communications.	4-12
MISC Group		
:MISC?	Queries all settings related to the MISC group.	4-13
:MISC:CALibration	Executes calibration.	4-13
:MISC:DATE	Sets the date or queries the current setting.	4-14
:MISC:TCPIp?	Queries all settings related to TCP/IP.	4-14
:MISC:TCPIp:DISPlay?	Queries all the TCP/IP settings that the instrument is using.	4-14
:MISC:TCPIp:DISPlay:DHCP?	Queries whether or not a DHCP server is being used.	4-14
:MISC:TCPIp:DISPlay:GATeway?	Queries the current gateway.	4-14
:MISC:TCPIp:DISPlay:HSTName?	Queries the current host name.	4-14
:MISC:TCPIp:DISPlay:IPADdress?	Queries the current IP address.	4-14
:MISC:TCPIp:DISPlay:MACaddress?	Queries the current MAC address.	4-14
:MISC:TCPIp:DISPlay:NETMask?	Queries the current netmask value.	4-14
:MISC:TCPIp:SETTing?	Queries all TCP/IP settings that are activated when the execution command is issued.	4-14
:MISC:TCPIp:SETTing:DHCP	Sets whether or not to use a DHCP server or queries the current setting.	4-14
:MISC:TCPIp:SETTing:GATeway	Sets the gateway or queries the current setting.	4-15
:MISC:TCPIp:SETTing:HSTName	Sets the host name or queries the current setting.	4-15
:MISC:TCPIp:SETTing:IPADdress	Sets the IP address or queries the current setting.	4-15
:MISC:TCPIp:SETTing:NETMask	Sets the netmask or queries the current setting.	4-15
:MISC:TCPIp:SETTing:PASSwd	Sets the password or queries the current setting.	4-15
:MISC:TCPIp:SETTing:SET	Executes the setting of TCP/IP.	4-15
:MISC:TIME	Sets the time or queries the current setting.	4-15
OUTPut Group		
:OUTPut<x>?	Queries all settings related to the OUTPUT.	4-17
:OUTPut<x>:ATT	Sets the attenuator or queries the current setting.	4-17
:OUTPut<x>:DELay	Sets the delay when generating waveforms or queries the current setting.	4-17
:OUTPut<x>:DFORmat	Sets the digital out format or queries the current setting.	4-17
:OUTPut<x>:DOUT	Turns On/Off digital out or queries the current setting.	4-18
:OUTPut<x>:FILTer	Sets the type of low-pass filter or queries the current setting.	4-18
:OUTPut<x>:LABel	Sets the label or queries the current setting.	4-18
:OUTPut<x>:LOAD	Executes the load operation from the setup file.	4-18
:OUTPut<x>:IQGAin	Sets the I/Q gain ratio or queries the current setting.	4-18
:OUTPut<x>:OFFSet?	Queries all settings related to the offset.	4-18
:OUTPut<x>:OFFSet:ICOARse	Sets the offset on the I side of the single-ended output model or queries the current setting.	4-18
:OUTPut<x>:OFFSet:ICOMm	Sets the common-mode offset on the I side of the differential output model or queries the current setting.	4-18
:OUTPut<x>:OFFSet:IDIFf	Sets the differential offset on the I side of the differential output model or queries the current setting.	4-18
:OUTPut<x>:OFFSet:IFINe	Sets the offset on the I side of the single-ended output model or queries the current setting.	4-18
:OUTPut<x>:OFFSet:QCOARse	Sets the offset on the Q side of the single-ended output model or queries the current setting.	4-19
:OUTPut<x>:OFFSet:QCOMm	Sets the common-mode offset on the Q side of the differential output model or queries the current setting.	4-19
:OUTPut<x>:OFFSet:QDIFf	Sets the differential offset on the Q side of the differential output model or queries the current setting.	4-19
:OUTPut<x>:OFFSet:QFINe	Sets the offset on the Q side of the single-ended output model or queries the current setting.	4-19
:OUTPut<x>:PHASe	Sets the phase or queries the current setting.	4-19
:OUTPut<x>:QUADrature	Sets the quadrature offset or queries the current setting.	4-19
:OUTPut<x>:WAVeform?	Queries all settings related waveform data.	4-19
:OUTPut<x>:WAVeform:LOAD	Executes the loading of the waveform group list.	4-19

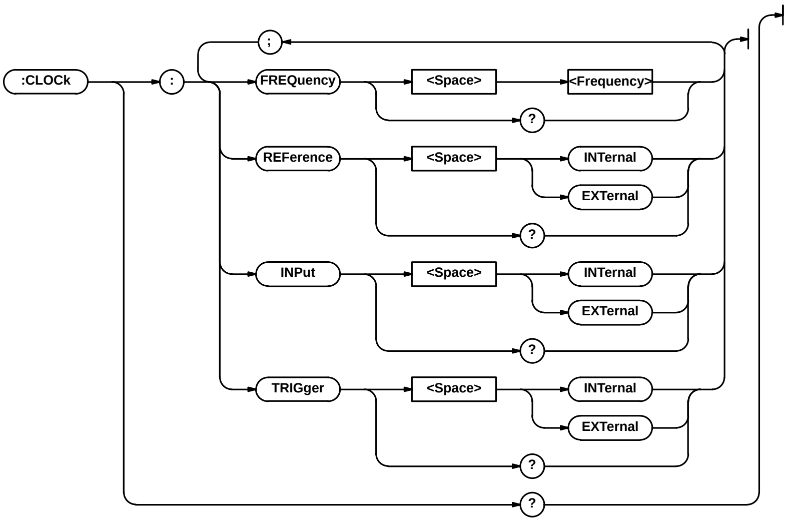
Command	Functions	Page
:OUTPut<x>:WAVeform:SAVE	Executes the saving of the waveform group list.	4-19
:OUTPut<x>:WAVeform:SELect?	Queries all settings related the selection of waveform data.	4-20
:OUTPut<x>:WAVeform:SELect:ALLLoad	Executes the loading of all waveform data that are registered in the specified OUTPUT.	4-20
:OUTPut<x>:WAVeform:SELect:ENTRy?	Queries the contents of the waveform data of the specified number.	4-20
:OUTPut<x>:WAVeform:SELect:LOAD	Executes the loading of the waveform data of the specified number.	4-20
:OUTPut<x>:WAVeform:SELect:NUMBer	Sets the waveform data number in the waveform data selection list or queries the current setting.	4-20
:OUTPut<x>:WAVeform:SELect:SELect	Selects the waveform data of the specified number for the output waveform.	4-20
:OUTPut<x>:WAVeform:SELect:SENTry?	Queries the contents of the currently selected waveform data.	4-20
:OUTPut<x>:WAVeform:STUP?	Queries all settings related the registration of waveform data.	4-20
:OUTPut<x>:WAVeform:STUP:DELete	Executes the deletion of the waveform data of the specified number.	4-21
:OUTPut<x>:WAVeform:STUP:ENTRy?	Queries the contents of the waveform data of the specified number.	4-21
:OUTPut<x>:WAVeform:STUP:NUMBer	Sets the waveform data number in the waveform data registration list or queries the current setting.	4-21
:OUTPut<x>:WAVeform:STUP:REGist?	Queries all settings related the waveform data to be registered.	4-21
:OUTPut<x>:WAVeform:STUP:REGist:BODY	Sets the body file of the waveform data to be registered or queries the current setting.	4-21
:OUTPut<x>:WAVeform:STUP:REGist:HEADer	Sets the header file of the waveform data to be registered or queries the current setting.	4-21
:OUTPut<x>:WAVeform:STUP:REGist:NAME	Sets the name of the waveform data to be registered or queries the current setting.	4-21
:OUTPut<x>:WAVeform:STUP:REGist:SET	Registers the name, header file, and body file at the specified number.	4-22
STARt Group		
:STARt	Starts waveform generation.	4-22
STATus Group		
:STATus?	Queries all settings related to the communication status function.	4-23
:STATus:CONDition?	Queries the contents of the condition register.	4-23
:STATus:EESE	Sets the extended event enable register or queries the current setting.	4-23
:STATus:EESR?	Queries the extended event register and clear the register.	4-23
:STATus:ERRor?	Queries the code and description of the error that occurred.	4-24
:STATus:FILTer<x>	Sets the transition filter or queries the current setting.	4-24
:STATus:QENable	Sets whether or not to store messages other than errors to the error queue or queries the current setting.	4-24
:STATus:QMESsage	Sets whether or not to attach information to the response to the [:STATus:ERRor?] query or queries the current setting.	4-24
:STATus:SPOLl?	Executes serial polling.	4-24
STOP Group		
:STOP	Stops waveform generation.	4-24
SYSTem Group		
:SYSTem?	Queries all settings related to the system.	4-25
:SYSTem:ATLoad	Executes the loading of waveform data of all OUTPUTs to the waveform memory.	4-25
:SYSTem:LCD	Turns On/Off the display backlight or queries the current setting.	4-25
:SYSTem:OUTPut	Turns On/Off the waveform output or queries the current setting.	4-25
Common Command Group		
*CAL?	Performs calibration and queries the result.	4-26
*CLS	Clears the standard event register, extended event register, and error queue.	4-26
*ESE	Sets the standard event enable register or queries the current setting.	4-26
*ESR?	Queries the standard event register and clears the register.	4-27
*IDN?	Queries the instrument model.	4-27

Command	Functions	Page
*OPC	Sets an OPC event after the completion of the specified overlap command.	4-27
*OPC?	Creates a response after the completion of the specified overlap command.	4-27
*PSC	Sets whether or not to clear the registers at power up or queries the current setting.	4-27
*RST	Initializes settings.	4-27
*SRE	Sets the service request enable register or queries the current setting.	4-28
*STB?	Queries the status byte register.	4-28
*TST?	Performs a self-test and queries the result.	4-28
*WAI	Holds the subsequent command until the completion of the specified overlap operation.	4-28

4.2 CLOCk Group

The commands in this group deal with the clock and trigger.

You can make the same settings and inquiries as when the CLOCK key on the front panel is used.



:CLOCk?

Function Queries all settings related to the clock and trigger.

```
Syntax  :CLOCk?
Example :CLOCK?→:CLOCK:FREQUENCY 10.000000E+06;
        REFERENCE INTERNAL;INPUT INTERNAL;
        TRIGGER INTERNAL
```

:CLOCk:FREQuency

Function Sets the clock frequency or queries the current setting.

```
Syntax  :CLOCk:FREQuency {<Frequency>}
        :CLOCk:FREQuency?
Example :CLOCK:FREQUENCY 10MHz
        :CLOCK:FREQUENCY?→:CLOCK:
        FREQUENCY 10.000000E+06
```

:CLOCk:INPut

Function Selects internal or external clock input or queries the current setting.

```
Syntax  :CLOCk:INPut {INTernal|EXTernal}
        :CLOCk:INPut?
Example :CLOCK:INPUT INTERNAL
        :CLOCK:INPUT?→:CLOCK:INPUT INTERNAL
```

:CLOCk:REFerence

Function Selects internal or external 10-MHz reference signal or queries the current setting.

```
Syntax  :CLOCk:REFerence {INTernal|EXTernal}
        :CLOCk:REFerence?
Example :CLOCK:REFERENCE INTERNAL
        :CLOCK:REFERENCE?→:CLOCK:
        REFERENCE INTERNAL
```

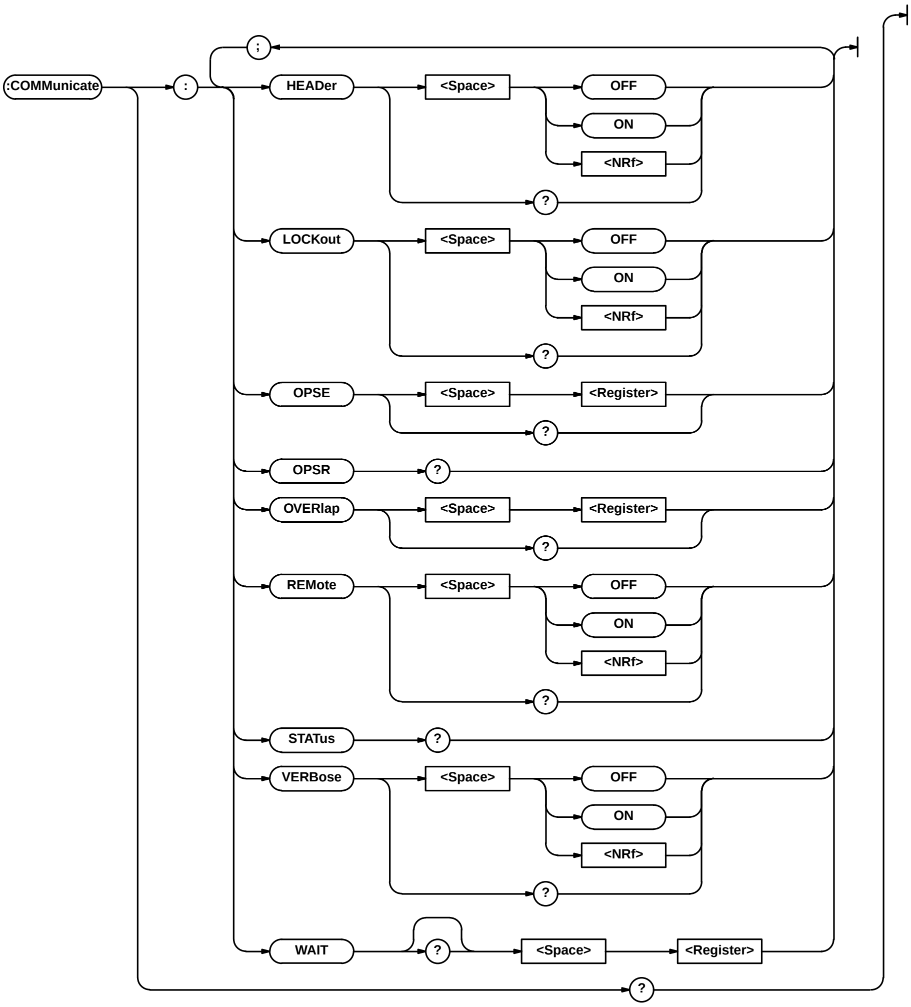
:CLOCk:TRIGger

Function Selects internal or external start trigger or queries the current setting.

```
Syntax  :CLOCk:TRIGger {INTernal|EXTernal}
        :CLOCk:TRIGger?
Example :CLOCK:TRIGGER INTERNAL
        :CLOCK:TRIGGER?→:CLOCK:TRIGGER INTERNAL
```

4.3 COMMunicate Group

The commands in this group deal with communications. There are no front panel keys that correspond to the commands in this group.



:COMMunicate?

Function Queries all settings related to communications.

Syntax `:COMMunicate?`

Example `:COMMUNICATE?→:COMMUNICATE:HEADER 1;`
`VERBOSE 1`

:COMMunicate:HEADer

Function Sets whether to add a header to the response to a query (example: `OUTPUT1:ATT -10.0`) or not add the header (example: `-10.0`).

Syntax `:COMMunicate:HEADer {<Boolean>}`
`:COMMunicate:HEADer?`

Example `:COMMUNICATE:HEADER ON`
`:COMMUNICATE:HEADER?→:COMMUNICATE:`
`HEADER 1`

:COMMunicate:LOCKout

Function Sets or clears local lockout.

Syntax `:COMMunicate:LOCKout {<Boolean>}`
`:COMMunicate:LOCKout?`

Example `:COMMUNICATE:LOCKOUT ON`
`:COMMUNICATE:LOCKOUT?→:COMMUNICATE:`
`LOCKOUT 1`

Description This is a command specific to the serial (RS-232) interface.

:COMMunicate:OPSE (Operation Pending Status Enable register)

Function Sets the overlap command that is to used by the `*OPC`, `*OPC?`, and `*WAI` commands or queries the current setting.

Syntax
```
:COMMunicate:OPSE {<Register>}
:COMMunicate:OPSE?
<Register>=0 to 65535
```

Example
```
:COMMUNICATE:OPSE 65535
:COMMUNICATE:OPSE?→:COMMUNICATE:OPSE 0
```

Description Since there are no overlap commands on the instrument, 0 is always returned.

:COMMunicate:OPSR? (Operation Pending Status Register)

Function Queries the value of the operation pending status register.

Syntax
```
:COMMunicate:OPSR?
```

Example
```
:COMMUNICATE:OPSR→0
```

Description There are no overlap commands on the VB8000.

:COMMunicate:OVERlap

Function Sets the commands to operate as overlap commands or queries the current setting.

Syntax
```
:COMMunicate:OVERlap {<Register>}
:COMMunicate:OVERlap?
<Register>=0 to 65535
```

Example
```
:COMMUNICATE:OVERLAP 65535
:COMMUNICATE:OVERLAP?→:COMMUNICATE:
OVERLAP 0
```

Description Since there are no overlap commands on the instrument, 0 is always returned.

:COMMunicate:REMote

Function Sets remote or local. ON is remote mode.

Syntax
```
:COMMunicate:REMote {<Boolean>}
:COMMunicate:REMote?
```

Example
```
:COMMUNICATE:REMOTE ON
:COMMUNICATE:REMOTE?→:COMMUNICATE:
REMOTE 1
```

Description This is a command specific to the serial (RS-232) interface.

:COMMunicate:STATus?

Function Queries line-specific status.

Syntax
```
:COMMunicate:STATus?
```

Example
```
:COMMUNICATE:STATUS?→0
```

Description The meaning of each status bit is as follows:

Bit	GP-IB	Serial
0	Unrecoverable transmission error	Parity error
1	Always 0	Framing error
2	Always 0	Break character detected
Other	Always 0	Always 0

The status bit is set when the corresponding cause occurs and cleared when it is read.

:COMMunicate:VERBose

Function Sets whether to return the response to a query using full spelling (example: `OUTPUT1:DFORMAT TCOMPLEMENTS`) or using abbreviation (example: `OUT1:DFOR TCOM`).

Syntax
```
:COMMunicate:VERBose {<Boolean>}
:COMMunicate:VERBose?
```

Example
```
:COMMUNICATE:VERBOSE ON
:COMMUNICATE:VERBOSE?→:COMMUNICATE:
VERBOSE 1
```

:COMMunicate:WAIT

Function Waits for one of the specified extended events to occur.

Syntax
```
:COMMunicate:WAIT {<Register>}
<Register>=0 to 65535 (extended event
register, see page 5-4)
```

Example
```
:COMMUNICATE:WAIT 65535
```

:COMMunicate:WAIT?

Function Creates the response that is returned when the specified event occurs.

Syntax
```
:COMMunicate:WAIT? {<Register>}
<Register>=0 to 65535 (extended event
register, see page 5-4)
```

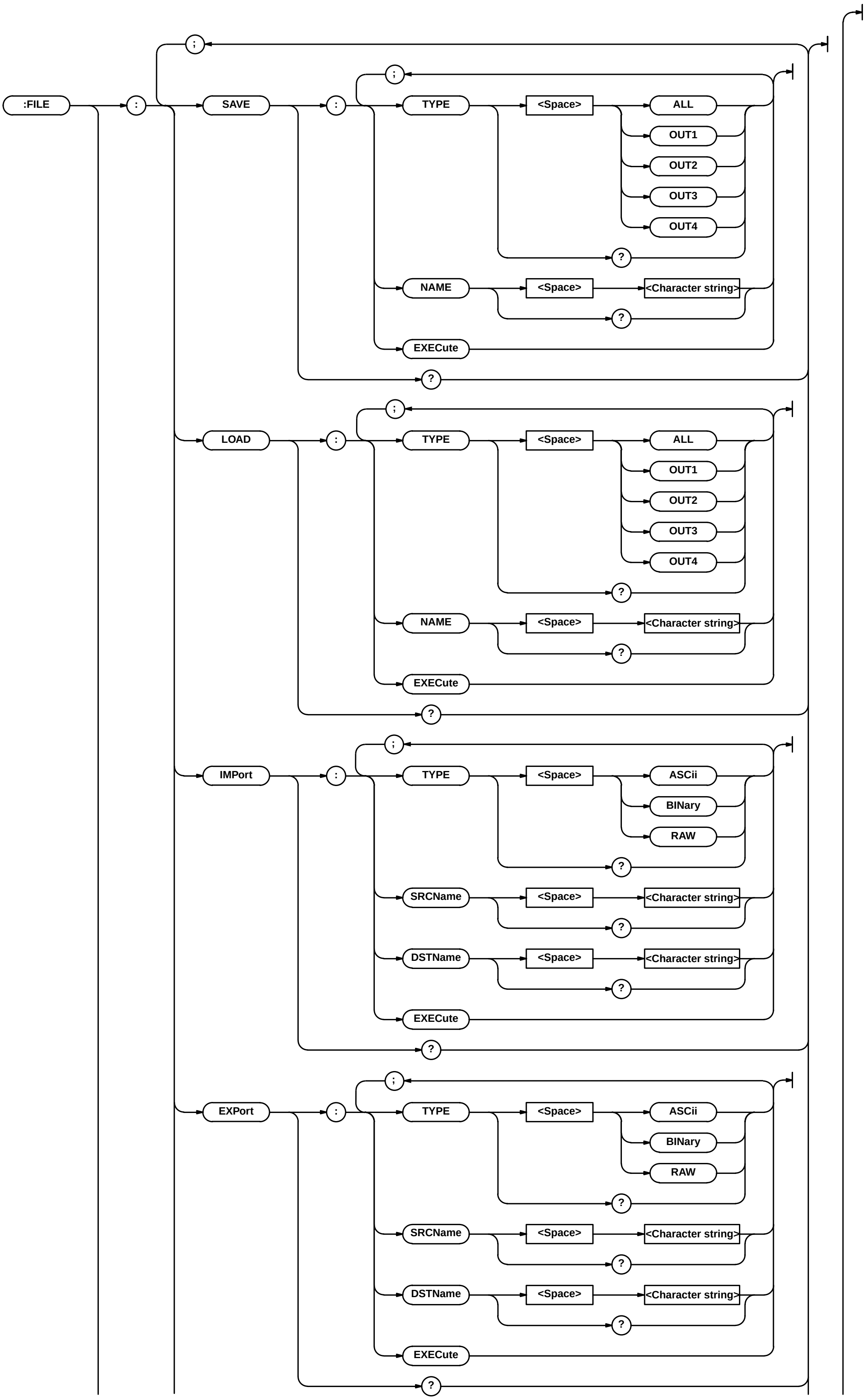
Example
```
:COMMUNICATE:WAIT? 65535→1
```

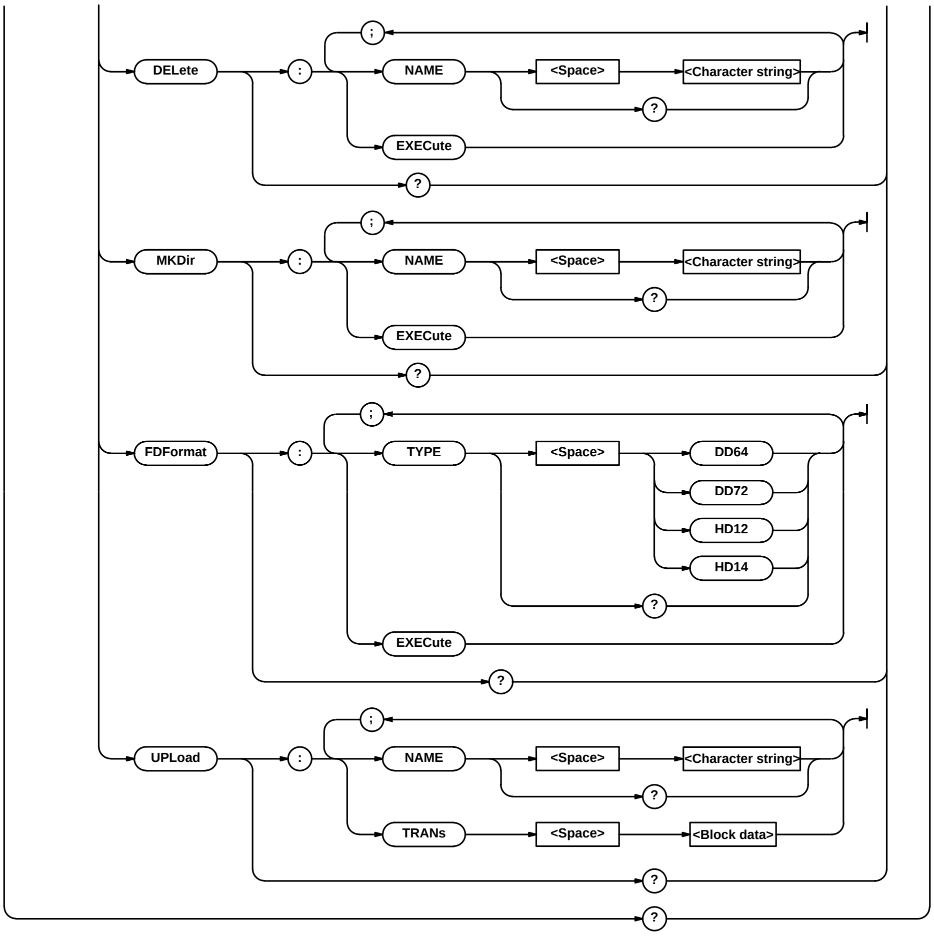
Operation pending status register/overlap enable register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.4 FILE Group

The commands in this group deal with file operations on the built-in hard disk and floppy disk. You can make the same settings and inquiries as when the FILE key on the front panel is used.





:FILE?

Function Queries all settings related to files.

Syntax :FILE?

Example FILE?→:FILE:SAVE:TYPE ALL;
NAME "/SC4-0/TST/TST.RAW";:FILE:LOAD:
TYPE ALL;NAME "/SC4-0/TST/TST.RAW";:
FILE:IMPORT:TYPE RAW;
SRCNAME "/SC4-0/TST/TST.RAW";
DSTNAME "/SC4-0/TST/TST.RAW";:FILE:
EXPORT:TYPE RAW;SRCNAME
"/SC4-0/TST/TST.RAW";
DSTNAME "/SC4-0/TST/TST.RAW";:FILE:
DELETE:NAME "/SC4-0/TST/TST.RAW";:FILE:
MKDIR:NAME "/SC4-0/TST/TST.RAW";:FILE:
FDFORMAT:TYPE HD14

:FILE:DELete?

Function Queries all settings related to the deletion of files.

Syntax :FILE:DELete?

Example :FILE:DELETE?→:FILE:DELETE:
NAME "/SC4-0/TST/TST.RAW"

:FILE:DELete:EXECute

Function Executes deleting of files.

Syntax :FILE:DELete:EXECute

Example :FILE:DELETE:EXECUTE

:FILE:DELete:NAME

Function Specifies the file to be deleted using a full path or queries the current setting.

Syntax :FILE:DELete:NAME?
:FILE:DELete:NAME {<Character string>}

Example :FILE:DELETE:NAME "/SC4-0/TST/TST.RAW"
:FILE:DELETE:NAME?→:FILE:DELETE:
NAME "/SC4-0/TST/TST.RAW"

Description Specify the file to be deleted using a full path and extension.

:FILE:EXPort?

Function Queries all settings related to exporting.

Syntax :FILE:EXPort?

Example :FILE:EXPORT?→:FILE:EXPORT:TYPE RAW;
SRCNAME "/SC4-0/RAW/SIN1K";
DSTNAME "/SC4-0/TST/TST"

:FILE:EXPort:DSTName

Function: Specifies the name of the export destination file using a full path or queries the current setting.

Syntax:
```
:FILE:EXPort:DSTName?
:FILE:EXPort:DSTName
{<Character string>}
```

Example:
```
:FILE:EXPORT:
DSTNAME "/SC4-0/TST/TST"
:FILE:EXPORT:DSTNAME?→:FILE:EXPORT:
DSTNAME "/SC4-0/TST/TST"
```

Description:
- Specify the file name using a full path. You can specify or omit the extension.
- The extension is automatically determined by the file type specified by the ":FILE:EXPort:TYPE" command. If you do not specify the extension, it is automatically added. If you specify a wrong extension, it is automatically corrected.

:FILE:EXPort:EXECute

Function: Executes the export operation.

Syntax:
```
:FILE:EXPort:EXECute
```

Example:
```
:FILE:EXPORT:EXECUTE
```

Description: Executes the export operation using the file type, the file to be exported, and the export destination that were specified before this command.

:FILE:EXPort:SRCName

Function: Specifies the name of the file to be exported using a full path or queries the current setting.

Syntax:
```
:FILE:EXPort:SRCName?
:FILE:EXPort:SRCName
{<Character string>}
```

Example:
```
:FILE:EXPORT:
SRCNAME "/SC4-0/RAW/SIN1K"
:FILE:EXPORT:SRCNAME?→:FILE:EXPORT:
SRCNAME "/SC4-0/RAW/SIN1K"
```

Description:
- Specify the file name using a full path. Do not specify the extension.
- The extension is automatically determined by the file type specified by the ":FILE:EXPort:TYPE" command. If you specify a extension, an execution error occurs.

:FILE:EXPort:TYPE

Function: Sets the type of file to be exported or queries the current setting.

Syntax:
```
:FILE:EXPort:TYPE?
:FILE:EXPort:TYPE {ASCii|BINary|RAW}
```

Example:
```
:FILE:EXPORT:TYPE RAW
:FILE:EXPORT:TYPE?→:FILE:EXPORT:
TYPE RAW
```

:FILE:FDFormat?

Function: Queries all settings related to the format of floppy disks.

Syntax:
```
:FILE:FDFormat?
```

Example:
```
:FILE:FDFORMAT?→:FILE:FDFORMAT:TYPE HD14
```

:FILE:FDFormat:EXECute

Function: Executes the floppy disk format.

Syntax:
```
:FILE:FDFormat:EXECute
```

Example:
```
:FILE:FDFORMAT:EXECUTE
```

Description: The floppy disk is formatted to the format type that was specified by the "FILE:FDFormat:TYPE" issued before this command.

:FILE:FDFormat:TYPE

Function: Sets the format type of the floppy disk or queries the current setting.

Syntax:
```
:FILE:FDFormat:TYPE?
:FILE:FDFormat:TYPE {DD64|DD72|HD12|
HD14}
```

Example:
```
:FILE:FDFORMAT:TYPE HD14
:FILE:FDFORMAT:TYPE?→:FILE:FDFORMAT:
TYPE HD14
```

:FILE:IMPort?

Function: Queries all settings related to importing.

Syntax:
```
:FILE:IMPort?
```

Example:
```
:FILE:IMPORT?→:FILE:IMPORT:TYPE RAW;
SRCNAME "/SC4-0/TST/TST";
DSTNAME "/SC4-0/TEST/TST"
```

:FILE:IMPort:DSTName

Function: Sets the import destination file name or queries the current setting.

Syntax:
```
:FILE:IMPort:DSTName?
:FILE:IMPort:DSTName
{<Character string>}
```

Example:
```
:FILE:IMPORT:
DSTNAME "/SC4-0/TEST/TST"
:FILE:IMPORT:DSTNAME?→:FILE:IMPORT:
DSTNAME "/SC4-0/TEST/TST"
```

Description:
- Specify the file name using a full path. Do not specify the extension.
- The extension is automatically determined by the file type specified by the ":FILE:IMPort:TYPE" command. If you specify a extension, an execution error occurs.

:FILE:IMPort:EXECute

Function Executes the import operation.

Syntax `:FILE:IMPort:EXECute`

Example `:FILE:IMPORT:EXECUTE`

Description Executes the import operation using the file type, the file to be imported, and the import destination that were specified before this command.

:FILE:IMPort:SRCName

Function Specifies the name of the file to be imported using a full path or queries the current setting.

Syntax
```
:FILE:IMPort:SRCName?
:FILE:IMPort:SRCName
{<Character string>}
```

Example
```
:FILE:IMPORT:
SRCNAME "/SC4-0/TST/TST"
:FILE:IMPORT:SRCNAME?→:FILE:IMPORT:
SRCNAME "/SC4-0/TST/TST"
```

Description
- Specify the file name using a full path. Do not specify the extension.
- The extension is automatically determined by the file type specified by the ":FILE:IMPort:TYPE" command. If you specify a extension, an execution error occurs.

:FILE:IMPort:TYPE

Function Sets the type of file to be imported or queries the current setting.

Syntax
```
:FILE:IMPort:TYPE?
:FILE:IMPort:TYPE {ASCii|BINary|RAW}
```

Example
```
:FILE:IMPORT:TYPE RAW
:FILE:IMPORT:TYPE?→:FILE:IMPORT:TYPE RAW
```

:FILE:LOAD?

Function Queries all settings related to the loading of settings of each OUTPUT or all OUTPUTs.

Syntax `:FILE:LOAD?`

Example
```
:FILE:LOAD?→:FILE:LOAD:TYPE ALL;
NAME "/SC4-0/TEST.ALL"
```

:FILE:LOAD:EXECute

Function Executes the loading of settings of each OUTPUT or all OUTPUTs.

Syntax `:FILE:LOAD:EXECute`

Example `:FILE:LOAD:EXECUTE`

Description Executes the load operation using the load destination and the file to be loaded that were specified before this command.

:FILE:LOAD:NAME

Function Specifies the name of the file to be loaded using a full path or queries the current setting.

Syntax
```
:FILE:LOAD:NAME?
:FILE:LOAD:NAME {<Character string>}
```

Example
```
:FILE:LOAD:NAME "/SC4-0/TEST.ALL"
:FILE:LOAD:NAME?→:FILE:LOAD:
NAME "/SC4-0/TEST.ALL"
```

Description
- Specify the file name using a full path. You can specify or omit the extension.
- The extension is automatically determined by the load destination that was specified by the ":FILE:LOAD:TYPE" command. If you do not specify the extension, the file with the extension matching the type is automatically selected. If you specify a wrong extension, an execution error occurs.

:FILE:LOAD:TYPE

Function Sets the load destination OUTPUT or queries the current setting.

Syntax
```
:FILE:LOAD:TYPE?
:FILE:LOAD:TYPE {ALL|OUT1|OUT2|OUT3|
OUT4}
```

Example
```
:FILE:LOAD:TYPE ALL
:FILE:LOAD:TYPE?→:FILE:LOAD:TYPE ALL
```

Description
- For ALL, the extension of the file to be loaded is ".ALL." Otherwise, it is ".CH."
- If you select ALL, all OUTPUTs become the load destination.

:FILE:MKDir?

Function Queries all settings related to directory creation.

Syntax `:FILE:MKDir?`

Example
```
:FILE:MKDIR?→:FILE:MKDIR:
NAME "/SC4-0/TTT"
```

:FILE:MKDir:EXECute

Function Creates the directory.

Syntax `:FILE:MKDir:EXECute`

Example `:FILE:MKDIR:EXECUTE`

Description Creates a directory using the directory name that was specified before this command.

:FILE:MKDir:NAME

Function Sets the name of the directory to be created or queries the current setting.

Syntax
```
:FILE:MKDir:NAME?
:FILE:MKDir:NAME {<Character string>}
```

Example
```
:FILE:MKDIR:NAME "/SC4-0/TTT"
:FILE:MKDIR:NAME?→:FILE:MKDIR:
NAME "/SC4-0/TTT"
```

Description
- Specify the directory using a full path.
- Only the last directory is created. Therefore, all upper-level directories must exist beforehand.

:FILE:SAVE?

Function Queries all settings related to the saving of settings of each OUTPUT or all OUTPUTs.

Syntax `:FILE:SAVE?`

Example `:FILE:SAVE?→:FILE:SAVE:TYPE ALL;NAME "/SC4-0/TEST.ALL"`

:FILE:SAVE:EXECute

Function Executes the saving of settings of each OUTPUT or all OUTPUTs.

Syntax `:FILE:SAVE:EXECute`

Example `:FILE:SAVE:EXECUTE`

Description Settings are saved according to the items to be saved and the save destination file that were specified before this command.

:FILE:SAVE:NAME

Function Specifies the name of the save destination file using a full path or queries the current setting.

Syntax `:FILE:SAVE:NAME?`
`:FILE:SAVE:NAME {<Character string>}`

Example `:FILE:SAVE:NAME "/SC4-0/TEST.ALL"`
`:FILE:SAVE:NAME?→:FILE:SAVE:NAME "/SC4-0/TEST.ALL"`

Description
- Specify the file name using a full path. You can specify or omit the extension.
- The extension is automatically determined by the save destination that was specified by the ":FILE:SAVE:TYPE" command. If you do not specify the extension, the file with the extension matching the type is selected. If you specify a wrong extension, an execution error occurs.

:FILE:SAVE:TYPE

Function Sets the OUTPUT or all OUTPUTs to be saved or queries the current setting.

Syntax `:FILE:SAVE:TYPE?`
`:FILE:SAVE:TYPE {ALL|OUT1|OUT2|OUT3|OUT4}`

Example `:FILE:SAVE:TYPE ALL`
`:FILE:SAVE:TYPE?→:FILE:SAVE:TYPE ALL`

Description For ALL, the extension of the file to be saved is ".ALL." Otherwise, it is ".CH."

:FILE:UPLoad:NAME

Function Sets the name of the file of the data transfer destination or queries the current setting.

Syntax `:FILE:UPLoad:NAME?`
`:FILE:UPLoad:NAME {<Character string>}`

Example `:FILE:UPLOAD:NAME "/SC4-0/TST/MTST.RAW"`

Description Specify the file name using a full path. Specify the extension also.

:FILE:UPLoad:TRANs

Function Executes data transfer via communications.

Syntax `:FILE:UPLoad:TRANs {<Block data>}`

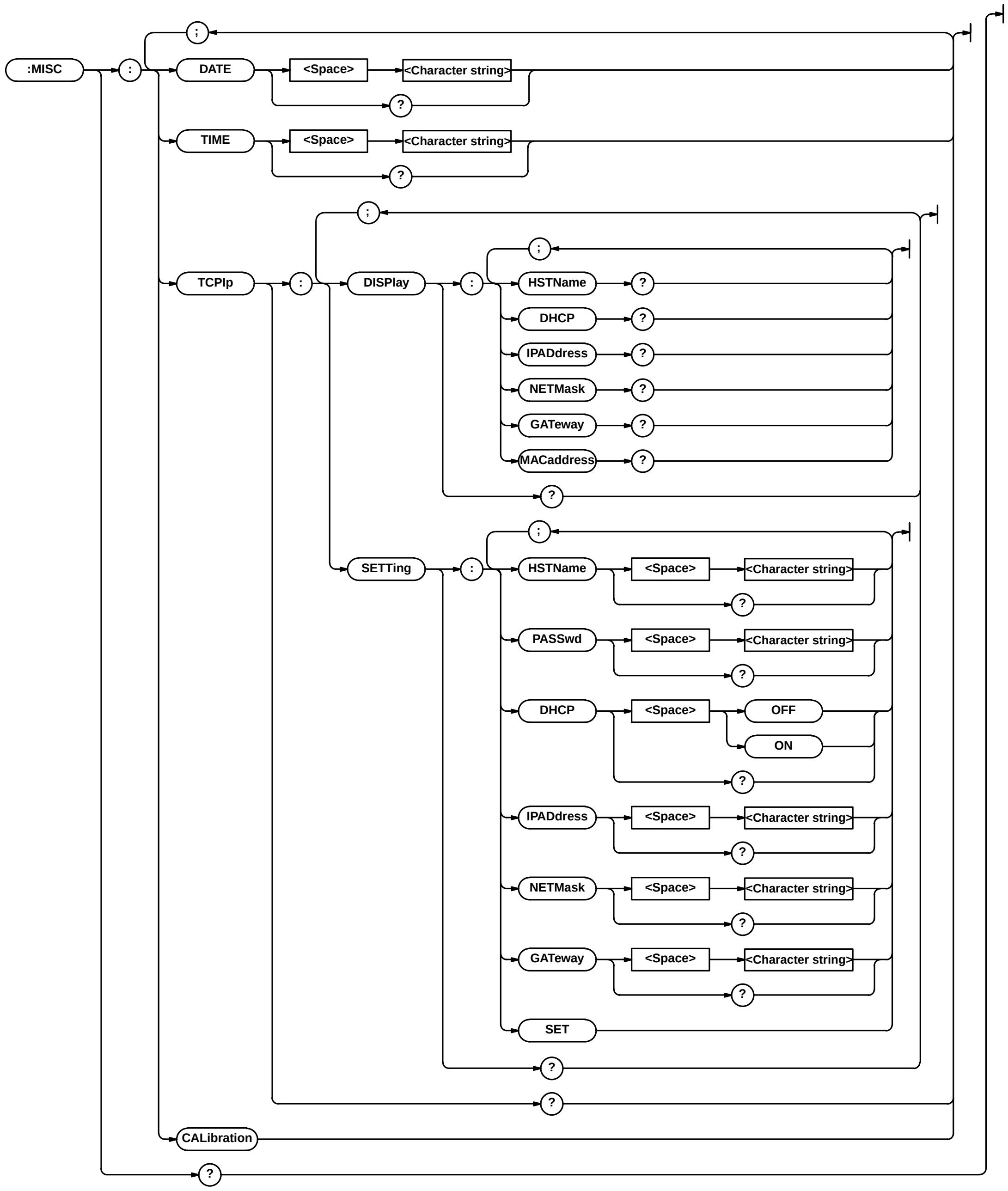
Example `:FILE:UPLOAD:TRANS #800004104.....`

Description The name of the save destination file must be specified using the "FILE:UPLOAD:NAME" command before this command.

4.5 MISC Group

The commands in this group deal with date, time, TCP/IP, and other settings.

You can make the same settings and inquiries as when the MISC key on the front panel is used.



:MISC?

Function Queries all settings related to the MISC group.

Syntax :MISC?

Example :MISC?→:MISC:TCPIP:DISPLAY:
HSTNAME "localhost.localdomain";
DHCP OFF;IPADDRESS "127.0.0.1";
NETMASK "255.0.0.0";GATEWAY "0.0.0.0";
MACADDRESS "00:00:64:82:10:03";:MISC:
TCPIP:SETTING:
HSTNAME"localhost.localdomain";
PASSWD "";DHCP OFF;IPADDRESS"127.0.0.1";
NETMASK "255.0.0.0";GATEWAY "0.0.0.0"

:MISC:CALibration

Function Executes calibration.

Syntax :MISC:CALibration

Example :MISC:CALIBRATION

:MISC:DATE

Function Sets the date or queries the current setting.

Syntax
```
:MISC:DATE?
:MISC:DATE {<String>}
<String>="YYYY/MM/DD"
```

Example
```
:MISC:DATE "20000/09/07"
:MISC:DATE?→:MISC:DATE "2000/09/07"
```

Description
- A query returns the current date.
- Upper level query does not return this information.

:MISC:TCPIp?

Function Queries all settings related to TCP/IP.

Syntax
```
:MISC:TCPIp?
```

Example
```
:MISC:TCPIP?→:MISC:TCPIP:DISPLAY:
HSTNAME "localhost.localdomain";
DHCP OFF;IPADDRESS "127.0.0.1";
NETMASK "255.0.0.0";GATEWAY "0.0.0.0";
MACADDRESS "00:00:64:82:10:03";:MISC:
TCPIP:SETTING:
HSTNAME "localhost.localdomain";
PASSWD "";DHCP OFF;
IPADDRESS "127.0.0.1";
NETMASK "255.0.0.0";GATEWAY "0.0.0.0"
```

:MISC:TCPIp:DISPlay?

Function Queries all the TCP/IP settings that the instrument is using.

Syntax
```
:MISC:TCPIp:DISPlay?
```

Example
```
:MISC:TCPIP:DISPLAY?→:MISC:TCPIP:
DISPLAY:HSTNAME "localhost.localdomain";
DHCP OFF;IPADDRESS "127.0.0.1";
NETMASK "255.0.0.0";GATEWAY "0.0.0.0";
MACADDRESS "00:00:64:82:10:03"
```

Description
- This command queries the current parameter information.
- It queries the information that appears when the [TCP/IP Info] soft key under the MISC key is pressed on the front panel.

:MISC:TCPIp:DISPlay:DHCP?

Function Queries whether or not a DHCP server is being used.

Syntax
```
:MISC:TCPIp:DISPlay:DHCP?
```

Example
```
:MISC:TCPIP:DISPLAY:DHCP?→:MISC:TCPIP:
DISPLAY:DHCP OFF
```

:MISC:TCPIp:DISPlay:GATeway?

Function Queries the current gateway.

Syntax
```
:MISC:TCPIp:DISPlay:GATeway?
```

Example
```
:MISC:TCPIP:DISPLAY:GATEWAY?→:MISC:
TCPIP:DISPLAY:GATEWAY "0.0.0.0"
```

:MISC:TCPIp:DISPlay:HSTName?

FunctionQueries the current host name.

Syntax
```
:MISC:TCPIp:DISPlay:HSTName?
```

Example
```
:MISC:TCPIP:DISPLAY:HSTNAME?→:MISC:
TCPIP:DISPLAY:
HSTNAME "localhost.localdomain"
```

:MISC:TCPIp:DISPlay:IPADdress?

Function Queries the current IP address.

Syntax
```
:MISC:TCPIp:DISPlay:IPADdress?
```

Example
```
:MISC:TCPIP:DISPLAY:IPADDRESS?→:MISC:
TCPIP:DISPLAY:IPADDRESS "127.0.0.1"
```

:MISC:TCPIp:DISPlay:MACaddress?

Function Queries the current MAC address.

Syntax
```
:MISC:TCPIp:DISPlay:MACaddress?
```

Example
```
:MISC:TCPIP:DISPLAY:MACADDRESS?→:MISC:
TCPIP:DISPLAY:
MACADDRESS "00:00:64:82:10:03"
```

:MISC:TCPIp:DISPlay:NETMask?

Function Queries the current netmask value.

Syntax
```
:MISC:TCPIp:DISPlay:NETMask?
```

Example
```
:MISC:TCPIP:DISPLAY:NETMASK?→:MISC:
TCPIP:DISPLAY:NETMASK "255.0.0.0"
```

:MISC:TCPIp:SETTing?

Function Queries all TCP/IP settings that are activated when the “MISC:TCPIp:SETTing:SET” command is issued.

Syntax
```
:MISC:TCPIp:SETTing?
```

Example
```
:MISC:TCPIP:SETTING?→:MISC:TCPIP:
SETTING:HSTNAME "localhost.localdomain";
PASSWD "";DHCP OFF;IPADDRESS
"127.0.0.1";NETMASK "255.0.0.0";
GATEWAY "0.0.0.0"
```

Description This is the same information that is displayed for each item when the [TCP/IP Setting] soft key of the MISC key on the front panel is pressed.

:MISC:TCPIp:SETTing:DHCP

Function Sets whether or not to use a DHCP server or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:DHCP?
:MISC:TCPIp:SETTing:DHCP {OFF|ON}
```

Example
```
:MISC:TCPIP:SETTING:DHCP OFF
:MISC:TCPIP:SETTING:DHCP?→:MISC:TCPIP:
SETTING:DHCP OFF
```

Description
- Set to ON when using a DHCP server, and OFF when not.
- The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:GATeway

Function Sets the gateway or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:GATeway?
:MISC:TCPIp:SETTing:GATeway {<String>}
```

Example
```
:MISC:TCPIP:SETTING:GATEWAY "0.0.0.0"
:MISC:TCPIP:SETTING:GATEWAY?→:MISC:
TCPIP:SETTING:GATEWAY "0.0.0.0"
```

Description The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:HSTName

Function Sets the host name or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:HSTName?
:MISC:TCPIp:SETTing:HSTName {<String>}
```

Example
```
:MISC:TCPIP:SETTING:
HSTNAME "localhost.localdomain"
:MISC:TCPIP:SETTING:HSTNAME?→:MISC:
TCPIP:SETTING:
HSTNAME "localhost.localdomain"
```

Description The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:IPADdress

Function Sets the IP address or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:IPADdress?
:MISC:TCPIp:SETTing:IPADdress {<String>}
```

Example
```
:MISC:TCPIP:SETTING:
IPADDRESS "127.0.0.1"
:MISC:TCPIP:SETTING:IPADDRESS?→:MISC:
TCPIP:SETTING:IPADDRESS "127.0.0.1"
```

Description The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:NETMask

Function Sets the netmask or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:NETMask?
:MISC:TCPIp:SETTing:NETMask {<String>}
```

Example
```
:MISC:TCPIP:SETTING:NETMASK "255.0.0.0"
:MISC:TCPIP:SETTING:NETMASK?→:MISC:
TCPIP:SETTING:NETMASK "255.0.0.0"
```

Description The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:PASSwd

Function Sets the password or queries the current setting.

Syntax
```
:MISC:TCPIp:SETTing:PASSwd?
:MISC:TCPIp:SETTing:PASSwd {<String>}
```

Example
```
:MISC:TCPIP:SETTING:PASSWD ""
:MISC:TCPIP:SETTING:PASSWD?→:MISC:
TCPIP:SETTING:PASSWD ""
```

Description The ":MISC:TCPIp:SETTing:SET" command must be executed for the settings to be activated on the VB8000.

:MISC:TCPIp:SETTing:SET

Function Executes the setting of TCP/IP.

Syntax
```
:MISC:TCPIp:SETTing:SET
```

Example
```
:MISC:TCPIP:SETTING:SET
```

Description Applies the settings that were specified using ":MISC:TCPIp:SETTing:***" command to the VB8000.

:MISC:TIME

Function Sets the time or queries the current setting.

Syntax
```
:MISC:TIME?
:MISC:TIME {<String>}
<String>="HH:MM:SS"
```

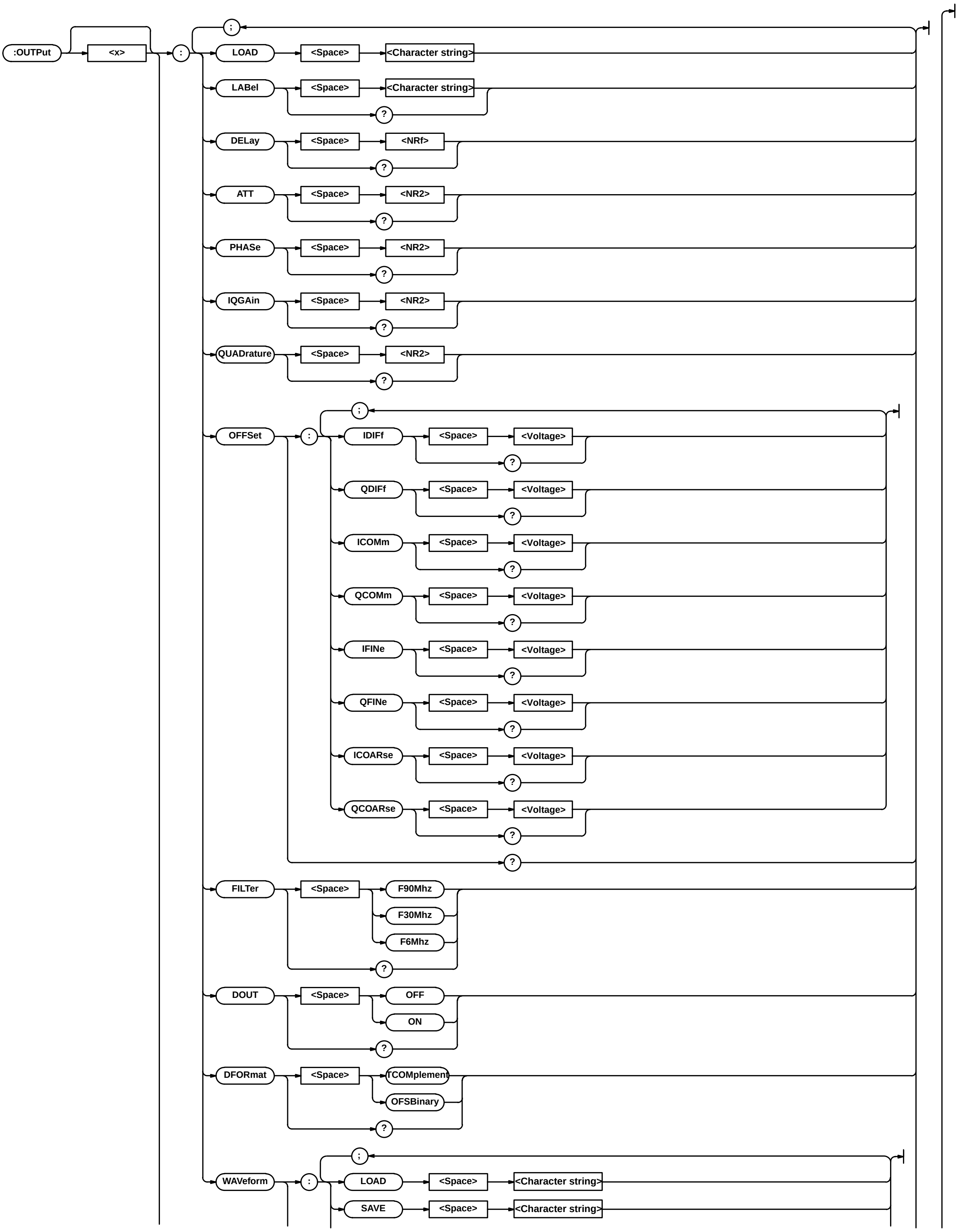
Example
```
:MISC:TIME "14:00:20"
:MISC:TIME?→:MISC:TIME "14:00:20"
```

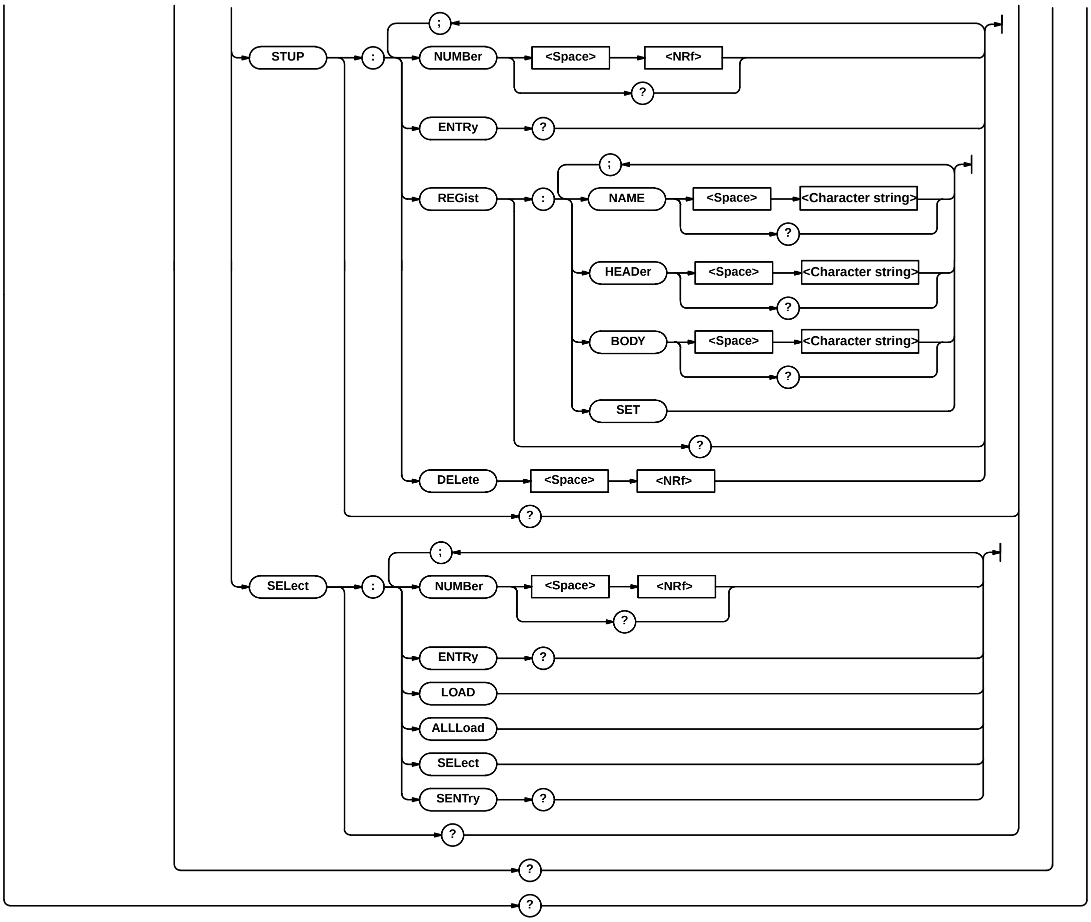
Description
- A query returns the current time.
- Upper level query does not return this information.

4.6 OUTPut Group

The commands in this group deal with each OUTPUT.

You can make the same settings, execution, and inquiries as when the OUTPUT1 to OUTPUT4 menu of the SETUP key on the front panel is used.





:OUTPut<x>?

Function Queries all settings related to the OUTPUT.

Syntax
```
:OUTPut<x>?
<x>=1 to 4
```

Example
```
:OUTPUT1?→:OUTPUT1:LABEL "WAVE 1";
DELAY 100;ATT -10.0;PHASE 20.0;
IQGAIN -10.0;QUADRATURE -20.0;OFFSET:
IDIFF -100.000E-03;QDIFF 100.000E-03;
ICOMM -500.000E-03;QCOMM 1.500000E+00;
IFINE -100.000E-03;QFINE 100.000E-03;
ICOARSE -500.000E-03;
QCOARSE 1.500000E+00;:OUTPUT1:
FILTER F30MHZ;DOUT ON;
DFORMAT TCOMPLEMENTS;WAVEFORM:STUP:
NUMBER 1;:OUTPUT1:WAVEFORM:SELECT:
NUMBER 1;SENTRY "1,L,SAMPLE,
/SC4-0/RAW/TRI1K.RAW,/SC4-0/RAW/
SIN1K.RAW"
```

:OUTPut<x>:ATT

Function Sets the attenuator or queries the current setting.

Syntax
```
:OUTPut<x>:ATT?
:OUTPut<x>:ATT {<NR2>}
<x>=1 to 4
```

Example
```
:OUTPUT1:ATT -10.0
:OUTPUT1:ATT?→:OUTPUT1:ATT -10.0
```

:OUTPut<x>:DELay

Function Sets the delay when generating waveforms or queries the current setting.

Syntax
```
:OUTPut<x>:DELay?
:OUTPut<x>:DELay {<NRf>}
<x>=1 to 4
```

Example
```
:OUTPUT1:DELAY 100
:OUTPUT1:DELAY?→:OUTPUT1:DELAY 100
```

:OUTPut<x>:DFORmat

Function Sets the digital out format or queries the current setting.

Syntax
```
:OUTPut<x>:DFORmat?
:OUTPut<x>:DFORmat
{TCOMplement|OFSBinary}
<x>=1 to 4
```

Example
```
:OUTPUT1:DFORMAT TCOMPLEMENT
:OUTPUT1:DFORMAT?→:OUTPUT1:
DFORMAT TCOMPLEMENTS
```

:OUTPut<x>:DOUT

Function Turns On/Off digital out or queries the current setting.

Syntax
```
:OUTPut<x>:DOUT?
:OUTPut<x>:DOUT {OFF|ON}
<x>=1 to 4
```

Example
```
:OUTPUT1:DOUT ON
:OUTPUT1:DOUT?→:OUTPUT1:DOUT ON
```

:OUTPut<x>:FILTer

Function Sets the type of low-pass filter or queries the current setting.

Syntax
```
:OUTPut<x>:FILTer?
:OUTPut<x>:FILTer {F90Mhz|F30Mhz|F6MHz}
<x>=1 to 4
```

Example
```
:OUTPUT1:FILTER F30MHZ
:OUTPUT1:FILTER?→:OUTPUT1:FILTER F30MHZ
```

:OUTPut<x>:LABel

Function Sets the label of the OUTPUT or queries the current setting.

Syntax
```
:OUTPut<x>:LABel?
:OUTPut<x>:LABel {<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:LABEL "WAVE 1"
:OUTPUT1:LABEL?→:OUTPUT1:LABEL "WAVE 1"
```

:OUTPut<x>:LOAD

Function Loads setup information to the OUTPUT.

Syntax
```
:OUTPut<x>:LOAD {<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:LOAD "/SC4-0/TEST.CH"
```

Description
- Specify the file name using a full path.
- If you do not specify the extension, it is automatically set to ".CH." If you specify a wrong extension, an execution error occurs.

:OUTPut<x>:IQGAin

Function Sets the I/Q gain ratio or queries the current setting.

Syntax
```
:OUTPut<x>:IQGAin?
:OUTPut<x>:IQGAin {<NR2>}
<x>=1 to 4
```

Example
```
:OUTPUT1:IQGAIN -10.0
:OUTPUT1:IQGAIN?→:OUTPUT1:IQGAIN -10.0
```

:OUTPut<x>:OFFSet?

Function Queries all settings related to the offset.

Syntax
```
:OUTPut<x>:OFFSet?
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET?→:OUTPUT1:OFFSET:
IDIFF 0.0E+00;QDIFF 100.000E-03;
ICOMM 1.500000E+00;QCOMM 0.0E+00;
IFINE 0.0E+00;QFINE 100.000E-03;
ICOARSE 1.500000E+00;QCOARSE 0.0E+00
```

:OUTPut<x>:OFFSet:ICOARse

Function Sets the offset on the I side of the single-ended output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:ICOARse?
:OUTPut<x>:OFFSet:ICOARse {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:ICOARSE -300MV
:OUTPUT1:OFFSET:ICOARSE?→:OUTPUT1:
OFFSET:ICOARSE -300.000E-03
```

Description This command is valid only on single-ended output models.

:OUTPut<x>:OFFSet:ICOMm

Function Sets the common-mode offset on the I side of the differential output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:ICOMm?
:OUTPut<x>:OFFSet:ICOMm {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:ICOMM -200MV
:OUTPUT1:OFFSET:ICOMM?→:OUTPUT1:
OFFSET:ICOMM -200.000E-03
```

Description This command is valid only on differential output models.

:OUTPut<x>:OFFSet:IDIFf

Function Sets the differential offset on the I side of the differential output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:IDIFf?
:OUTPut<x>:OFFSet:IDIFf {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:IDIFF -50.0MV
:OUTPUT1:OFFSET:IDIFF?→:OUTPUT1:
OFFSET:IDIFF -50.000E-03
```

Description This command is valid only on differential output models.

:OUTPut<x>:OFFSet:IFINe

Function Sets the offset on the I side of the single-ended output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:IFINe?
:OUTPut<x>:OFFSet:IFINe {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:IFINE -40.0MV
:OUTPUT1:OFFSET:IFINE?→:OUTPUT1:
OFFSET:IFINE -40.000E-03
```

Description This command is valid only on single-ended output models.

:OUTPut<x>:OFFSet:QCOARse

Function Sets the offset on the Q side of the single-ended output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:QCOARse?
:OUTPut<x>:OFFSet:QCOARse {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:QCOARSE 1200MV
:OUTPUT1:OFFSET:QCOARSE?→:OUTPUT1:
OFFSET:QCOARSE 1.200000E+00
```

Description This command is valid only on single-ended output models.

:OUTPut<x>:OFFSet:QCOMm

Function Sets the common-mode offset on the Q side of the differential output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:QCOMm?
:OUTPut<x>:OFFSet:QCOMm {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:QCOMM 1300MV
:OUTPUT1:OFFSET:QCOMM?→:OUTPUT1:OFFSET:
QCOMM 1.300000E+00
```

Description This command is valid only on differential output models.

:OUTPut<x>:OFFSet:QDIFf

Function Sets the differential offset on the Q side of the differential output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:QDIFf?
:OUTPut<x>:OFFSet:QDIFf {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:QDIFF 20.4MV
:OUTPUT1:OFFSET:QDIFF?→:OUTPUT1:OFFSET:
QDIFF 20.400E-03
```

Description This command is valid only on differential output models.

:OUTPut<x>:OFFSet:QFINe

Function Sets the offset on the Q side of the single-ended output model or queries the current setting.

Syntax
```
:OUTPut<x>:OFFSet:QFINe?
:OUTPut<x>:OFFSet:QFINe {<Voltage>}
<x>=1 to 4
```

Example
```
:OUTPUT1:OFFSET:QFINE 60.0MV
:OUTPUT1:OFFSET:QFINE?→:OUTPUT1:OFFSET:
QFINE 60.000E-03
```

Description This command is valid only on single-ended output models.

:OUTPut<x>:PHASe

Function Sets the phase or queries the current setting.

Syntax
```
:OUTPut<x>:PHASe?
:OUTPut<x>:PHASe {<NR2>}
<x>=1 to 4
```

Example
```
:OUTPUT1:PHASE 20.0
:OUTPUT1:PHASE?→:OUTPUT1:PHASE 20.0
```

:OUTPut<x>:QUADrature

Function Sets the quadrature offset or queries the current setting.

Syntax
```
:OUTPut<x>:QUADrature?
:OUTPut<x>:QUADrature {<NR2>}
<x>=1 to 4
```

Example
```
:OUTPUT1:QUADRATURE -20.0
:OUTPUT1:QUADRATURE?→:OUTPUT1:
QUADRATURE -20.0
```

:OUTPut<x>:WAVeform?

Function Queries all settings related waveform data.

Syntax
```
:OUTPut<x>:WAVeform?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM?→:OUTPUT1:WAVEFORM:
STUP:NUMBER 0;
ENTRY "0,U,AAA,/SC4-0/RAW/SIN1K.RAW,
/SC4-0/RAW/TRI1K.RAW";REGIST:NAME "";
HEADER "";BODY "";:OUTPUT1:WAVEFORM:
SELECT:NUMBER 0;ENTRY "0,U,AAA,
/SC4-0/RAW/SIN1K.RAW,
/SC4-0/RAW/TRI1K.RAW";SENTRY ""
```

:OUTPut<x>:WAVeform:LOAD

Function Executes the loading of the waveform group list.

Syntax
```
:OUTPut<x>:WAVeform:LOAD
{<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:LOAD "/SC4-0/TTT.IDX"
```

Description
- Specify the file name using a full path.
- If you omit the extension, [.IDX] is automatically determined. If you specify a wrong extension, an execution error occurs.

:OUTPut<x>:WAVeform:SAVE

Function Executes the saving of the waveform group list.

Syntax
```
:OUTPut<x>:WAVeform:SAVE
{<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SAVE "/SC4-0/TTT.IDX"
```

Description
- Specify the file name using a full path.
- If you omit the extension, [.IDX] is automatically determined. If you specify a wrong extension, an execution error occurs.

:OUTPut<x>:WAVeform:SELect?

Function Queries all settings related the selection of waveform data.

Syntax
```
:OUTPut<x>:WAVeform:SELect?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT?→
:OUTPUT1:WAVEFORM:SELECT:NUMBER 0;
ENTRY "1,U,SAMPLE,/SC4-0/RAW/TRI1K.RAW,
/SC4-0/RAW/SIN1K.RAW";SENTRY ""
```

Description Returns the entry number and the waveform data information of the currently selected waveform data.

:OUTPut<x>:WAVeform:SELect:ALLLoad

Function Executes the loading of all waveform data that are registered in the specified OUTPUT.

Syntax
```
:OUTPut<x>:WAVeform:SELect:ALLLoad
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:ALLLOAD
```

Description This command may take awhile, because all waveform data that are registered in the specified OUTPUT are loaded.

:OUTPut<x>:WAVeform:SELect:ENTRy?

Function Queries the waveform data information corresponding to the number that was specified by the ":OUTPut<x>:WAVeform:SELect:NUMBer" command.

Syntax
```
:OUTPut<x>:WAVeform:SELect:ENTRy?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:ENTRY?→
:OUTPUT1:WAVEFORM:SELECT:
ENTRY "1,U,SAMPLE,/SC4-0/RAW/TRI1K.RAW,
/SC4-0/RAW/SIN1K.RAW"
```

Description
- The format of the waveform data information is "<Number>,<Status (U/L/E)>,<Waveform data name>,<Header file name>,<Body file name>."
- Check the information using this command before executing the ":OUTPut<x>:WAVeform:SELect:LOAD" command.

:OUTPut<x>:WAVeform:SELect:LOAD

Function Loads the waveform data corresponding to the number that was specified by the ":OUTPut<x>:WAVeform:SELect:NUMBer" command.

Syntax
```
:OUTPut<x>:WAVeform:SELect:LOAD
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:LOAD
```

:OUTPut<x>:WAVeform:SELect:NUMBer

Function Sets the waveform data number in the waveform data selection list or queries the current setting.

Syntax
```
:OUTPut<x>:WAVeform:SELect:NUMBer?
:OUTPut<x>:WAVeform:SELect:NUMBer
{<NRf>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:NUMBER 1
:OUTPUT1:WAVEFORM:SELECT:NUMBER?→
:OUTPUT1:WAVEFORM:SELECT:NUMBER 1
```

Description This command by itself does not affect the system. However, the number that is specified by this command is used when commands such as ":OUTPut<x>:WAVeform:SELect:LOAD" and ":OUTPUT<x>:WAVeform:SELect:SELect" are executed.

:OUTPut<x>:WAVeform:SELect:SELect

Function Sets output waveform to the waveform data corresponding to the number that was specified by the ":OUTPut<x>:WAVeform:SELect:NUMBer" command.

Syntax
```
:OUTPut<x>:WAVeform:SELect:SELect
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:SELECT
```

Description The "OUTPut<x>:WAVeform:SELect:NUMBer" command must be issued before this command to specify the number.

:OUTPut<x>:WAVeform:SELect:SENTry?

Function Queries the contents of the currently selected waveform data.

Syntax
```
:OUTPut<x>:WAVeform:SELect:SENTry?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:SELECT:SENTRY?→
:OUTPUT1:WAVEFORM:SELECT:
SENTRY "1,L,SAMPLE,/SC4-0/RAW/TRI1K.RAW,
/SC4-0/RAW/SIN1K.RAW"
```

Description The format of the waveform data information is "<Number>,<Status (U/L/E)>,<Waveform data name>,<Header file name>,<Body file name>."

:OUTPut<x>:WAVeform:STUP?

Function Queries all settings related the registration of waveform data.

Syntax
```
:OUTPut<x>:WAVeform:STUP?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP?→:OUTPUT1:
WAVEFORM:STUP:NUMBER 0;
ENTRY "0,U,AAA,/SC4-0/RAW/SIN1K.RAW,
/SC4-0/RAW/TRI1K.RAW";REGIST:NAME "";
HEADER "";BODY ""
```

:OUTPut<x>:WAVeform:STUP:DELete

Function Deletes the waveform data corresponding to the number that was specified by the ":OUTPut<x>:WAVeform:STUP:NUMBer" command.

Syntax
```
:OUTPut<x>:WAVeform:STUP:DELete {<NRf>}
<x>=1 to 4
<NRf>=0 to 255
```

Example
```
:OUTPUT1:WAVEFORM:STUP:DELETE
```

:OUTPut<x>:WAVeform:STUP:ENTRy?

Function Queries the waveform data information corresponding to the number that was specified by the ":OUTPut<x>:WAVeform:STUP:NUMBer" command.

Syntax
```
:OUTPut<x>:WAVeform:STUP:ENTRy?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP:ENTRY?→
:OUTPUT1:WAVEFORM:STUP:ENTRY
"1,U,SAMPLE,/SC4-0/RAW/TRI1K.RAW,
/SC4-0/RAW/SIN1K.RAW"
```

Description
- The format of the waveform data information is "<Number>,<Status (U/L/E)>,<Waveform data name>,<Header file name>,<Body file name>."
- Check the waveform data information using this command before executing the registration command.

:OUTPut<x>:WAVeform:STUP:NUMBer

Function Sets the waveform data number in the waveform data registration list or queries the current setting.

Syntax
```
:OUTPut<x>:WAVeform:STUP:NUMBer?
:OUTPut<x>:WAVeform:STUP:NUMBer {<NRf>}
<x>=1 to 4
<NRf>=0 to 255
```

Example
```
:OUTPUT1:WAVEFORM:STUP:NUMBER 1
:OUTPUT1:WAVEFORM:STUP:NUMBER?→
:OUTPUT1:WAVEFORM:STUP:NUMBER 1
```

Description This command by itself does not affect the system. However, the number that is specified by this command is used when the ":OUTPut<x>:WAVeform:STUP:REGist:SET" command is executed.

:OUTPut<x>:WAVeform:STUP:REGist?

Function Queries all settings related the waveform data to be registered.

Syntax
```
:OUTPut<x>:WAVeform:STUP:REGist?
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP:REGIST?→
:OUTPUT1:WAVEFORM:STUP:REGIST:
NAME "SAMPLE";HEADER "TRI1K.RAW";
BODY "SIN1K.RAW"
```

:OUTPut<x>:WAVeform:STUP:REGist:BODY

Function Sets the body file of the waveform data to be registered or queries the current setting.

Syntax
```
:OUTPut<x>:WAVeform:STUP:REGist:BODY?
:OUTPut<x>:WAVeform:STUP:REGist:BODY
{<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP:REGIST:BODY?→
:OUTPUT1:WAVEFORM:STUP:REGIST:
BODY "/SC4-0/RAW/SIN1K.RAW"
```

Description
- Be sure to specify an extension.
- This command does not register the body file to the VB8000. It is registered when the "OUTPut<x>:WAVeform:STUP:REGist:SET" command is executed.

:OUTPut<x>:WAVeform:STUP:REGist:HEADer

Function Sets the header file of the waveform data to be registered or queries the current setting.

Syntax
```
:OUTPut<x>:WAVeform:STUP:REGist:HEADer?
:OUTPut<x>:WAVeform:STUP:REGist:HEADer
{<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP:REGIST:HEADER?→
:OUTPUT1:WAVEFORM:STUP:REGIST:
HEADER "/SC4-0/RAW/TRI1K.RAW"
```

Description
- Be sure to specify an extension.
- This command does not register the header file to the VB8000. It is registered when the "OUTPut<x>:WAVeform:STUP:REGist:SET" command is executed.

:OUTPut<x>:WAVeform:STUP:REGist:NAME

Function Sets the name of the waveform data to be registered or queries the current setting.

Syntax
```
:OUTPut<x>:WAVeform:STUP:REGist:NAME?
:OUTPut<x>:WAVeform:STUP:REGist:NAME
{<Character string>}
<x>=1 to 4
```

Example
```
:OUTPUT1:WAVEFORM:STUP:REGIST:
NAME "SAMPLE"
```

Description This command does not register the name to the VB8000. It is registered when the "OUTPut<x>:WAVeform:STUP:REGist:SET" command is executed.

:OUTPut<x>:WAVeform:STUP:REGist:SET

Function Registers the name, header file, and body file to the number that was specified by the ":OUTPut<x>:WAVeform:STUP:NUMBer" command.

Syntax `:OUTPut<x>:WAVeform:STUP:REGist:SET`
`<x>=1 to 4`

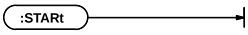
Example `:OUTPUT1:WAVEFORM:STUP:REGIST:SET`

Description Applies the values that were specified by "OUTPut<x>:WAVeform:STUP:REGist:***" commands to the VB8000.

4.7 STARt Group

The command in the STARt group is used to start waveform generation.

You can execute the same operation as the START/STOP key on the front panel.



:STARt

Function Starts waveform generation.

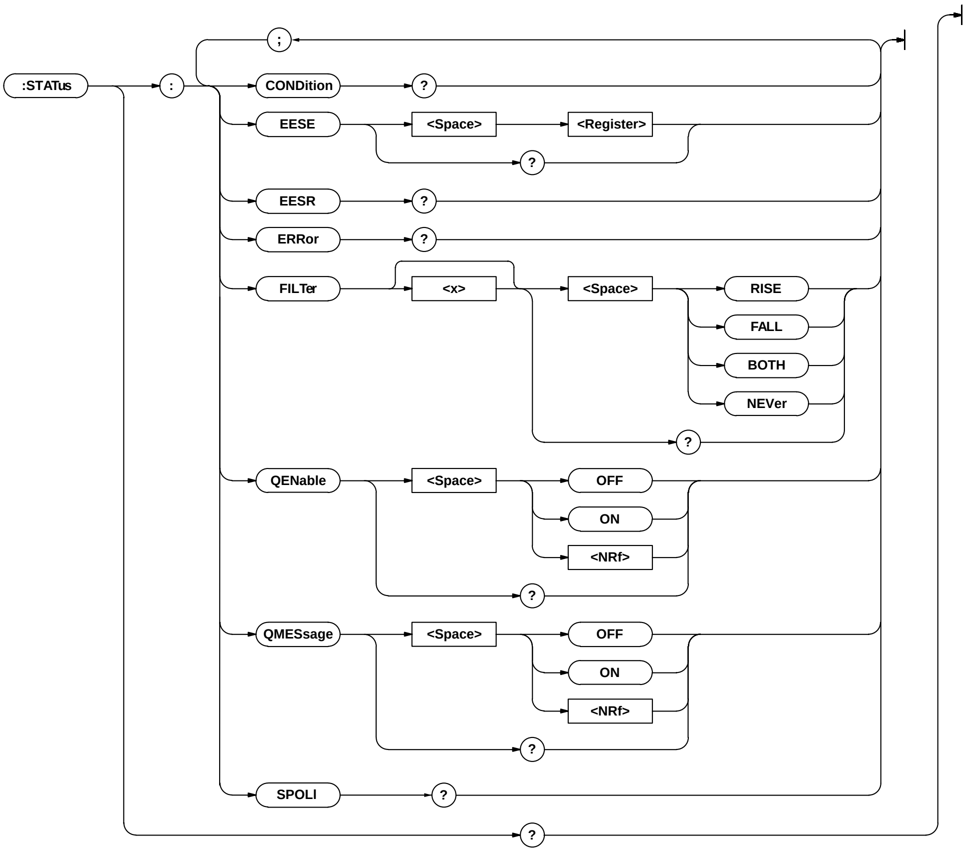
Syntax `:STARt`

Example `:START`

Description The "STOP" command is used to stop waveform generation.

4.8 STATus Group

The commands in the STATus group are used to make settings and inquiries related to the status report. There are no front panel keys that correspond to the commands in this group. For details on the status report, see chapter 5.



:STATus?

Function Queries all settings related to the communication status function.

Syntax :STATus?

Example :STATUS?→:STATUS:EESE 0;FILTER1 NEVER; FILTER2 NEVER;FILTER3 NEVER; FILTER4 NEVER;FILTER5 NEVER; FILTER6 NEVER;FILTER7 NEVER; FILTER8 NEVER;FILTER9 NEVER; FILTER10 NEVER;FILTER11 NEVER; FILTER12 NEVER;FILTER13 NEVER; FILTER14 NEVER;FILTER15 NEVER; FILTER16 NEVER;QENABLE 0;QMESSAGE 1

:STATus:CONDition?

Function Queries the contents of the condition register.

Syntax :STATus:CONDition?

Example :STATUS:CONDITION→16

Description For the description regarding how to synchronize the program using :STATus:CONDition, see page 3-8.

:STATus:EESE (Extended Event Status Enable register)

Function Sets the extended event enable register or queries the current setting.

Syntax :STATus:EESE {<Register>}
:STATus:EESE?
<Register>=0 to 65535

Example :STATUS:EESE #B00000000
:STATUS:EESE?→:STATUS:EESE 0

:STATus:EESR? (Extended Event Status Register)

Function Queries the extended event register and clear the register.

Syntax :STATus:EESR?

Example :STATUS:EESR?→0

:STATus:ERRor?

Function Queries the error code and message information (top of the error queue).

Syntax `:STATus:ERRor?`

Example `:STATUS:ERROR?→901,"Backup failure"`

Description
- When there is no error, "0, "No error"" is returned.
- The message cannot be returned in Japanese.
- You can specify whether or not to add the message using the "`STATus:QMESsage`" command.

:STATus:FILTer<x>

Function Sets the transition filter or queries the current setting.

Syntax `:STATus:FILTer<x> {RISE|FALL|BOTH|NEVer}`
`:STATus:FILTer<x>?`
`<x>=1 to 16`

Example `:STATUS:FILTER2 RISE`
`:STATUS:FILTER2?→:STATUS:FILTER2 RISE`

Description Specify how each bit of the condition register is to change to set the event. If "`RISE`" is specified, the event is set when the bit changes from "`0`" to "`1`."

:STATus:QENable

Function Sets whether or not to store messages other than errors to the error queue (ON/OFF) or queries the current setting.

Syntax `:STATus:QENable {<Boolean>}`
`:STATus:QENable?`

Example `:STATUS:QENABLE ON`
`:STATUS:QENABLE?→:STATUS:QENABLE 1`

:STATus:QMESsage

Function Sets whether or not to attach information to the response to the [`:STATus:ERRor?`] query (ON/OFF) or queries the current setting.

Syntax `:STATus:QMESsage {<Boolean>}`
`:STATus:QMESsage?`

Example `:STATUS:QMESSAGE ON`
`:STATUS:QMESSAGE?→:STATUS:QMESSAGE 1`

:STATus:SPOLl? (Serial Poll)

Function Executes serial polling.

Syntax `:STATus:SPOLl?`

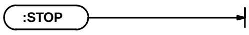
Example `:STATUS:SPOLL?→:STATUS:SPOLL 0`

Description This is a command specific to the serial (RS-232) interface.

4.9 STOP Group

The command in the STOP group is used to stop waveform generation.

You can execute the same operation as the START/STOP key on the front panel.



:STOP

Function Stops waveform generation.

Syntax `:STOP`

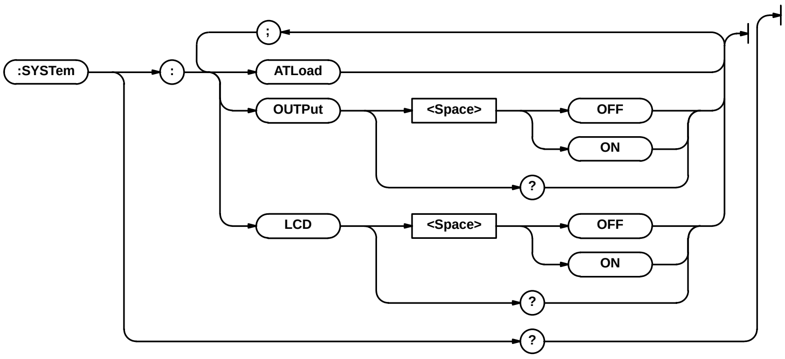
Example `:STOP`

Description The "`STARt`" command is used to start waveform generation.

4.10 SYSTem Group

The commands in this group deal with the system.

You can make the same settings, execution, and inquiries as when the [Auto Load] soft key of the SETUP key, the OUTPUT key, and the DISPLAY key on the front panel is used.



:SYSTem?

Function Queries all settings related to the system.

Syntax `:SYSTem?`

Example `:SYSTEM?→:SYSTEM:OUTPUT ON;LCD ON`

:SYSTem:ATLoad

Function Executes the loading of waveform data of all OUTPUTs to the waveform memory.

Syntax `:SYSTem:ATLoad`

Example `:SYSTEM:ATLOAD`

Description This command may take awhile, because all waveform data that are registered in all OUTPUTs are loaded.

:SYSTem:LCD

Function Turns On/Off the display backlight or queries the current setting.

Syntax `:SYSTem:LCD?`
`:SYSTem:LCD {OFF|ON}`

Example `:SYSTEM:LCD ON`
`:SYSTEM:LCD?→:SYSTEM:LCD ON`

:SYSTem:OUTPut

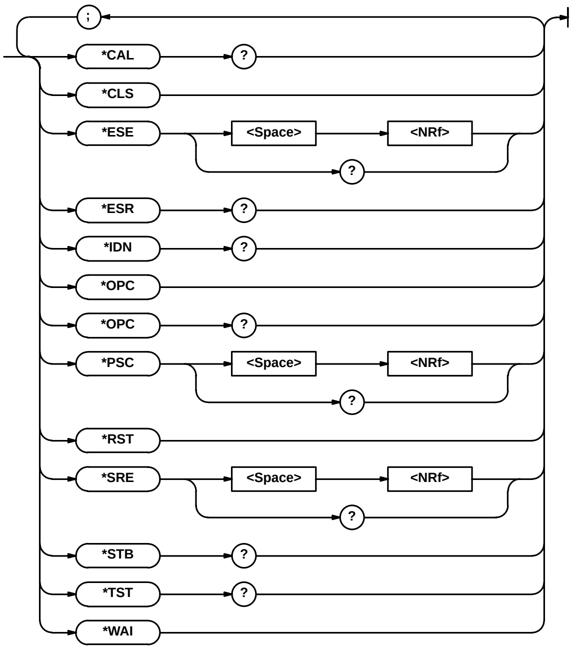
Function Turns On/Off the waveform output or queries the current setting.

Syntax `:SYSTem:OUTPut?`
`:SYSTem:OUTPut {OFF|ON}`

Example `:SYSTEM:OUTPUT ON`
`:SYSTEM:OUTPUT?→:SYSTEM:OUTPUT ON`

4.11 Common Command Group

The commands in the common group are defined in the IEEE 488.2-1992 and are independent of the instrument's functions. There are no front panel keys that correspond to the commands in this group.



*CAL? (CALibrate)

Function Performs calibration and queries the result.

Syntax `*CAL?`

Example `*CAL?→0`

Description If the calibration terminates normally, "0" is returned. If abnormality is detected, "1" is returned.

*CLS (CLear Status)

Function Clears the standard event register, extended event register, and error queue.

Syntax `*CLS`

Example `*CLS`

Description
- If the *CLS command is located immediately after the program message terminator, the output queue is also cleared.
- For details on the register and queue, see chapter 5.

*ESE (standard Event Status Enable register)

Function Sets the standard event enable register or queries the current setting.

Syntax `*ESE {<NRf>}`
`*ESE?`
`<NRf>=0 to 255`

Example `*ESE 253`
`*ESE?→253`

Description
- Specify the value as a sum of each bit in decimal notation.
- For example, specifying "*ESE 253" will cause the standard enable register to be set to "11111101." In this case, bit 2 of the standard event register is disabled which means that bit 5 (ESB) of the status byte register is not set to "1," even if a "query error" occurs.
- The default value is "*ESE 0" (all bits disabled).
- A query using *ESE? will not clear the contents of the standard event enable register.
- For details on the standard event enable register, see page 5-3.

*ESR?(standard Event Status Register)

Function Queries the standard event register and clears the register.

Syntax `*ESR?`

Example `*ESR?→32`

Description
- The sum of the bits is returned as a decimal value.
- You can check what type of events occurred when an SRQ is generated.
- For example, if a value of "32" is returned, this indicates that the standard event register is set to "00100000." In this case, you can see that the SRQ occurred due to a "command syntax error."
- A query using *ESR? will clear the contents of the standard event register.
- For details on the standard event register, see page 5-3.

*IDN? (IDeNtify)

Function Queries the instrument model.

Syntax `*IDN?`

Example `*IDN?→YOKOGAWA,703150-{16,64}{2,4,6,8},0,F1.01`

Description The information is returned in the following form: <Manufacturer>,<Model>,<Serial No.>,<Firmware version>In actuality, <Serial No.> is not returned (always 0).

*OPC (OPeration Complete)

Function Sets a "1" to bit 0 (OPC bit) of the standard event register bit upon the completion of the specified overlap command.

Syntax `*OPC`

Example `*OPC`

Description
- For the description regarding how to synchronize the program using *OPC, see page 3-7.
- The "COMMunicate:OPSE" command is used to specify the overlap command.
- If *OPC is not the last command of the message, the operation is not guaranteed.

*OPC? (OPeration Complete)

Function If *OPC? is transmitted and the specified overlap command is completed, ASCII code "1" is returned.

Syntax `*OPC?`

Example `*OPC?→1`

Description
- For the description regarding how to synchronize the program using *OPC, see page 3-7.
- The "COMMunicate:OPSE" command is used to specify the overlap command.
- If *OPC? is not the last command of the message, the operation is not guaranteed.

*PSC (Power-on Status Clear)

Function Sets whether or not to clear the registers below at power up or queries the current setting. The register is cleared when the value rounded to an integer is a non-zero value.
- Standard event enable register
- Extended event enable register
- Transition filter

Syntax
```
*PSC {<NRf>}
*PSC?
<NRf>=0 (don't clear), non-zero (clear)
```

Example
```
*PSC 1
*PSC?→1
```

Description For details on the registers, see chapter 5.

*RST (ReSeT)

Function Initializes settings.

Syntax `*RST`

Example `*RST`

Description Also clears *OPC and *OPC? commands that have been sent earlier.

*SRE (Service Request Enable register)

Function Sets the service request enable register or queries the current setting.

Syntax
```
*SRE {<NRf>}
*SRE?
<NRf>=0 to 255
```

Example
```
*SRE 239
*SRE?→239
```

Description
- Specify the value as a sum of each bit in decimal notation.
- For example, specifying "*SRE 239" will cause the service request enable register to be set to "11101111." In this case, bit 4 of the service request enable register is disabled which means that bit 5 (ESB) of the status byte register is not set to "1," even if "the output queue is not empty."
- Bit 6 (MSS) of the status byte register is the MSS bit itself, and therefore, it is ignored.
- The default value is "*SRE 0" (all bits disabled).
- A query using *SRE? will not clear the contents of the service request enable register.
- For details on the service request enable register, see page 5-1.

*STB? (STatus Byte)

Function Queries the status byte register.

Syntax
```
*STB?
```

Example
```
*STB?→4
```

Description
- The sum of the bits is returned as a decimal value.
- Since the register is read without executing serial polling, bit 6 is an MSS bit not RQS.
- For example, if a value of "4" is returned, this indicates that the status byte register is set to "00000100." In this case, you can see that "the error queue is not empty" (an error occurred).
- A query using *STB? will not clear the contents of the status byte register.
- For details on the status byte register, see page 5-2.

*TST?

Function Performs a self-test and queries the result. The self test involves internal memory tests.

Syntax
```
*TST?
```

Example
```
*TST?→0
```

Description
- "0" is returned if the self test is successful, "1" if it is not.
- This command executes the same Memory test as the Self Test menu of the MISC key.

*WAI (WAIt)

Function Holds the subsequent command until the completion of the specified overlap operation.

Syntax
```
*WAI
```

Example
```
*WAI
```

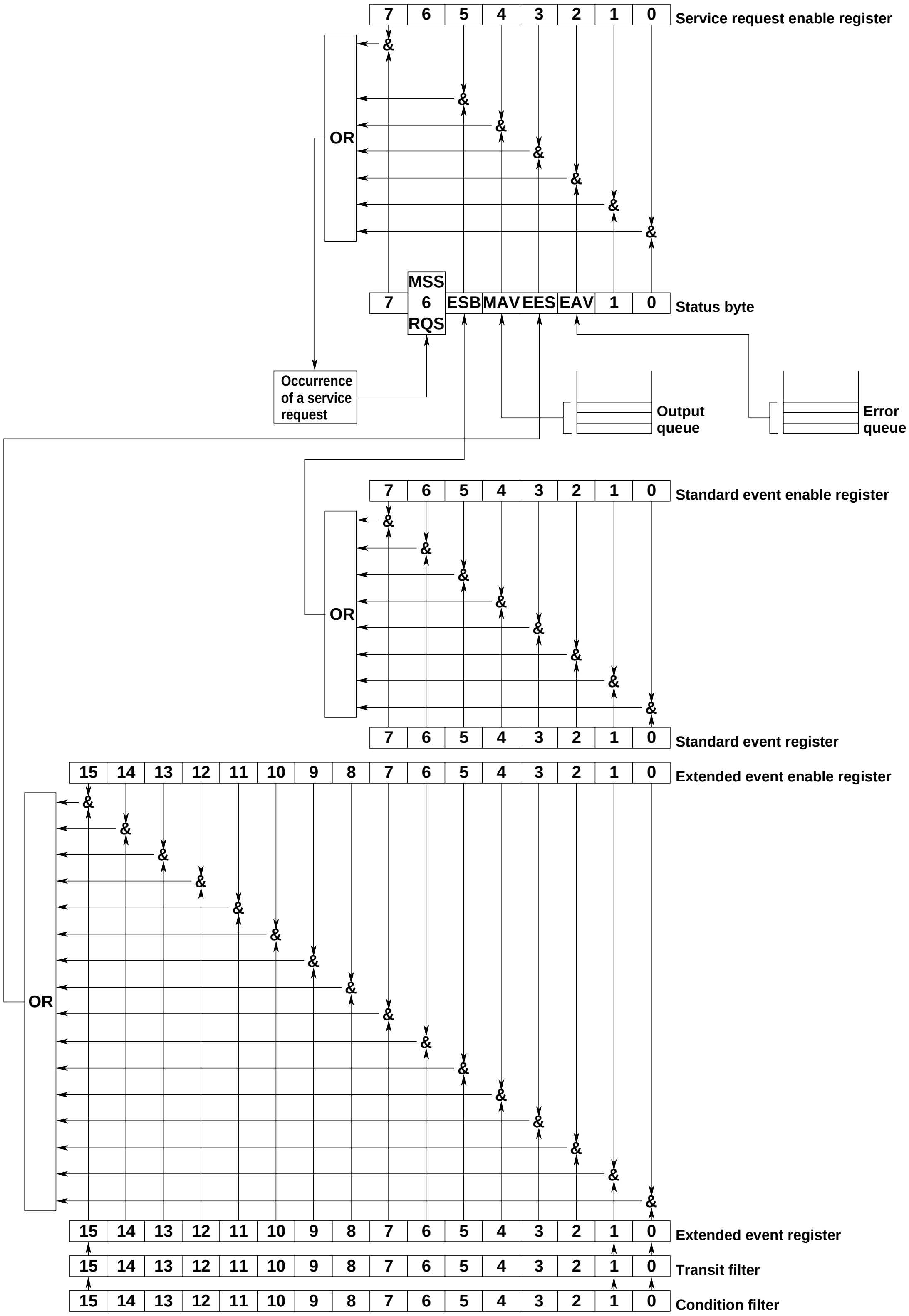
Description There are no overlap commands on the VB8000.

Chapter 5 Status Report

5.1 Status Reports

Status Reports

The figure below shows the status report that is read by serial polling. This status report is an extended version of the status report defined in IEEE 488.2-1992.



Summary of the Registers and Queues

Name	Functions	Write	Read
Status byte		–	Serial polling (RQS), *STB?(MSS)
Service request enable register	Status byte mask	*SRE	*SRE?
Standard event register	Changes in device status	–	*ESR?
Standard event enable register	Standard event register mask	*ESE	*ESE?
Extended event register	Changes in device status	–	STATus:EESR?
Extended event enable register	Extended event register mask	STATus:EESE	STATus:EESE?
Condition register	Current device status	–	STATus:CONDition?
Transition filter	Conditions that change the extended event register	STATus:FILTer<x>	STATus:FILTer<x>?
Output queue	Stores a response message to a query		All query commands
Error queue	Stores the error No. and message	–	STATus:ERRor?

Registers and Queues That Affect the Status Byte

Registers that affect the bits of the status byte are shown below.

Standard event register:	Sets bit 5 (ESB) of the status byte to "1" or "0."
Output queue:	Sets bit 4 (MAV) of the status byte to "1" or "0."
Extended event register:	Sets bit 3 (EES) of the status byte to "1" or "0."
Error queue:	Sets bit 2 (EAV) of the status byte to "1" or "0."

Enable Registers

Registers that are used to mask a bit so that the bit will not affect the status byte, even if it is set to 1, are shown below.

Status byte:	Mask the bits using the service request enable register.
Standard event register:	Mask the bits using the standard event enable register.
Extended event register:	Mask the bits using the extended event enable register.

Reading and Writing to the Registers

For example, the *ESE command is used to set the bits in the standard event enable register to 1's or 0's. The *ESE? command is used to query whether the bits in the standard event enable register are 1's or 0's. For details regarding these commands, see chapter 4.

5.2 Status Byte

Status Byte

	RQS						
7	6	ESB	MAV	EES	EAV	1	0
	MSS						

- **Bits 0, 1, and 7**
  Not used (always 0)
- **Bit 2 EAV (Error Available)**
  Set to "1" when the error queue is not empty. In other words, this bit is set to "1" when an error occurs. See the page 5-5.
- **Bit 3 EES (Extend Event Summary Bit)**
  Set to "0" when the logical product of the extended event register and the corresponding enable register is "1." In other words, this bit is set to "1" when an event occurs inside the instrument. See the page 5-4.
- **Bit 4 MAV (Message Available)**
  Set to "1" when the output queue is not empty. In other words, this bit is set to "1" when there are data to be transmitted. See the page 5-5.
- **Bit 5 ESB (Event Summary Bit)**
  Set to "0" when the logical product of the standard event register and the corresponding enable register is "1." In other words, this bit is set to "1" when an event occurs inside the instrument. See the page 5-3.

Bit 6 RQS (Request Service)/MSS (Master Status Summary)

Set to "1" when the logical AND of the status byte excluding Bit 6 and the service request enable register is not "0." In other words, this bit is set to "1" when the instrument is requesting service from the controller.

RQS is set to "1" when the MSS bit changes from "0" to "1," and cleared when serial polling is carried out or when the MSS bit changes to "0."

Bit Masking

If you wish to mask a certain bit of the status byte so that it does not cause an SRQ, set the corresponding bit of the service request enable register to "0."

For example, to mask bit 2 (EAV) so that service is not requested when an error occurs, set bit 2 of the service request enable register to "0." This is done using the *SRE command. The *SRE? request command can be used to query the service request enable register to check whether each bit is set to "1" or "0." For details on the *SRE command, see chapter 4.

Status Byte Operation

A service request is issued when bit 6 of the status byte becomes a "1." Bit 6 is set to "1" when any of the other bits becomes a "1" (when the corresponding bit of the service request enable register is also set to "1").

For example, if an event occurs and any of the bits of the logical AND of the standard event register and the corresponding enable register becomes a "1", then bit 5 (ESB) is set to "1." At this point, if bit 5 of the service request enable register is "1," then bit 6 (MSS) is set to "1" causing the instrument to request service from the controller.

In addition, you can also check what type of event occurred by reading the contents of the status byte.

Reading the Status Byte

The following two methods are available in reading the contents of the status byte:

- **Query using the *STB? command**
  A *STB? query causes bit 6 to be an MSS bit. Therefore, the MSS bit is read. No bits in the status byte are cleared after reading the status byte.
- **Serial polling**
  Serial polling causes bit 6 to be a RQS bit. Therefore, the RQS bit is read. After reading the status byte, only the RQS bit is cleared. You cannot read the MSS bit when serial polling is used.

Clearing the Status Byte

There are no methods available that can forcibly clear all the bits of the status byte. The bits that are cleared for each operation are shown below.

- **When a query is made using the *STB? command**
  None of the bits are cleared.
- **When serial polling is executed**
  Only the RQS bit is cleared.
- **When a *CLS command is received.**
  Receiving the *CLS command will not clear the status byte itself, but the contents of the standard event register that affect the status byte. As a result, the corresponding bit of the status byte is cleared. Since the *CLS command does not clear the output queue, bit 4 (MAV) of the status byte is unaffected. However, if the *CLS command is received immediately after the program message terminator, the output queue is also cleared.

5.3 Standard Event Register

Standard Event Register

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

- **Bit 7 PON (Power ON)**
  Set to "1" when the instrument is turned ON.
- **Bit 6 URQ (User Request)**
  Not used (always 0)
- **Bit 5 CME (Command Error)**
  Set to "1" when there is an error in the command syntax.
  Example Received a command name with a spelling error or character data not in the selection.
- **Bit 4 EXE (Execution Error)**
  Set to "1" when the command syntax is correct, but the command cannot be executed in the current state of the instrument.
  Example Received a command with a parameter outside the range or a command dealing with an unsupported option.
- **Bit 3 DDE (Device Dependent Error)**
  Set to "1" when a command cannot be executed for internal reasons other than a command syntax error and command execution error.
- **Bit 2 QYE (Query Error)**
  Set to "1" when a query command is transmitted, but the error queue is empty or the data are lost.
  Example No response data, output queue overflowed and data were lost.
- **Bit 1 RQC (Request Control)**
  Not used (always 0)
- **Bit 0 OPC (Operation Complete)**
  Set to "1" when the operation specified by the *OPC command (see chapter 4) has been completed.

Bit Masking

If you wish to mask a certain bit of the standard event register so that it does not cause bit 5 (ESB) of the status byte to change, set the corresponding bit of the standard event enable register to "0."

For example, to mask bit 2 (QYE) so that the ESB bit is not set to "1" when a query error occurs, set bit 2 of the standard event enable register to "0." This is done using the *ESE command. The *ESE? request command can be used to query the standard event enable register to check whether each bit is set to "1" or "0." For details on the *ESE command, see chapter 4.

Standard Event Register Operation

Standard event register is a register for the eight types of events that occur inside the instrument. When any of the bits becomes a "1," bit 5 (ESB) of the status byte is set to "1" (when the corresponding bit of the standard event enable register is also set to "1").

Example

1. A query error occurs.
2. Bit 2 (QYE) is set to "1."
3. If bit 2 of the standard event enable register is a "1", then bit 5 (ESB) of the status byte is set to "1."

In addition, you can also check what type of event occurred in the instrument by reading the contents of the standard event register.

Reading the Standard Event Register

ïThe *ESR? command can be used to read the contents of the standard event register. The register is cleared after it is read.

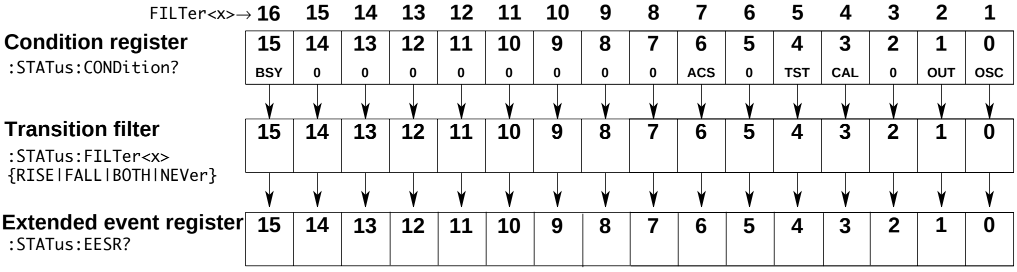
Clearing the Standard Event Register

The standard event register is cleared in the following three cases:

- When the contents of the standard event register are read using the *ESR command.
- When a *CLS command is received.
- When the instrument is power cycled.

5.4 Extended Event Register

The transition filter detects the changes in the condition register that indicate the internal condition of the instrument and writes the result to the extended event register.



FILTer<x>→	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Condition register :STATus:CONDition?	15 BSY	14 0	13 0	12 0	11 0	10 0	9 0	8 0	7 0	6 ACS	5 0	4 TST	3 CAL	2 0	1 OUT	0 OSC
Transition filter :STATus:FILTer<x> {RISE|FALL|BOTH|NEVer}	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Extended event register :STATus:EESR?	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The meaning of each bit of the condition register is as follows:

Bit 0	OSC (Oscillating) Set to 1 during oscillation (START).
Bit 1	OUT (Output) Set to 1 while outputting signals to the BNC connector on the front panel.
Bit 3	CAL (Calibration) Set to 1 while calibration is being executed.
Bit 4	TST (Testing) Set to 1 while self-test is being executed.
Bit 6	ACS (Accessing) Set to 1 while the floppy disk or built-in hard disk is being accessed.
Bit 15	BSY (Busy) Set to 1 while the instrument is being configured.

The transition filter parameters detect changes in the specified bit (numerical suffix, 1 to 16) of the condition register in the following manner and overwrite the extended event register.

RISE	Sets the specified bit of the extended event register to "1", on a 0-to-1 change.
FALL	Sets the specified bit of the extended event register to "1", on a 1-to-0 change.
BOTH	Sets the specified bit of the extended event register to "1", on both 0-to-1 and 1-to-0 change.
NEVer	Always 0.

5.5 Output Queue and Error Queue

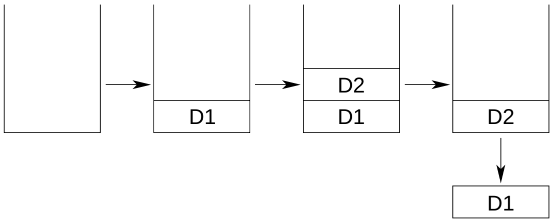
Output Queue

The output queue stores response messages for the queries.

As shown below, data are stored in order and read from the oldest ones first. The output queue is also cleared for the following cases:

- When a new message is received from the controller.
- When a deadlock occurs (see page 2-2)
- When a device clear command (DCL or SDC) is received.
- When the system is power cycled.

The *CLS command cannot be used to clear the output queue. Bit 4 (MAV) of the status byte can be used to check whether or not the output queue is empty.



Error Queue

The error queue stores the error number and message when an error occurs. For example, if the controller sends an incorrect program message, the error number “113” and the message “Undefined header” are stored in the error queue when the error is displayed.

The STATus:ERRor? query can be used to read the contents of the error queue. As with the output queue, the messages are read from the oldest ones first. When the error queue overflows, the last message is replaced by the message “350, Queue overflow.”

The error queue is also cleared for the following cases:

- When a *CLS command is received.
- When the system is power cycled.

Bit 2 (EAV) of the status byte can be used to check whether or not the error queue is empty.

6.1 Before Programming

Environment

Model:	MS-DOS computer equipped with AT-GPIB/TNT IEEE-488.2 board from National Instruments.
Language:	Quick Basic

Setting Up the VB8000

Initialization

None of the sample programs given in this chapter include initialization of the VB8000.

Address 1

All the sample programs given in this chapter use address 1 for the VB8000, so be sure to assign the instrument to address 1 as described on page 1-5.

6.2 Setting Waveform Generation Conditions for Each Output

```
'**********************************************************************
'*                                                                    *
'*      VB8000 Sample Program for GP-IB interface                     *
'*                    Microsoft QuickBASIC 4.0/4.5 Version            *
'*                                                                    *
'**********************************************************************
'
REM $INCLUDE: 'qbdecl4.bas'
'
DEVICE$ = "DEV1": CALL IBFIND(DEVICE$, DEV%)
CALL IBSIC(DEV%)
BORD$ = "GPIB0": CALL IBFIND(BORD$, BD%)
CALL IBSIC(BD%)
V% = 1: IBSRE(BD%, V%)
CALL IBCLR(DEV%)
'
CMD$ = ":COMMUNICATE:HEADER OFF"                  'Don't add headers to the query
responses.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":CLOCK:FREQUENCY 100MHz"                  'Set the clock frequency to 100
MHz.
CALL IBWRT(DEV%, CMD$)
'
'Settings for Output Port 1
CMD$ = ":OUTPUT1:DELAY 0"                         'Set the delay to 0 clock cycles.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:ATT -10"                         'Set the attenuator to -10 dB.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:PHASE 45"                        'Set the phase to 45 degrees.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:IQGAIN -10"                      'Set the I/Q gain to 10%.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:QUADRATURE -20"                  'Set the quadrature offset to -20
degrees.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:FILTER F30MHZ"                   'Set the filter to 30 MHz.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SETUP:NUMBER 1"         'Select number 1 in the waveform
data registration list.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SETUP:REGIST:NAME "+CHR$(34)+"SAMPLE"+CHR$(34)
                                                  'Set the name of the waveform data
to be registered to "SAMPLE."
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SETUP:REGIST:HEADER "+CHR$(34)+"/SC4-0/RAW/
TESTHEAD.RAW"+CHR$(34)
                                                  'Register a header file.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SETUP:REGIST:BODY "+CHR$(34)+"/SC4-0/RAW/
TESTBODY.RAW"+CHR$(34)
                                                  'Register a body file.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SETUP:REGIST:SET"       'Register to number 1 in the
waveform data registration list.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SELECT:NUMBER 1"        'Set the waveform data number to
1.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT1:WAVEFORM:SELECT:LOAD"            'Load waveform data number 1.
CALL IBWRT(DEV%, CMD$)
'
'Settings for Output Port 2
CMD$ = ":OUTPUT2:DELAY 0"                         'Set the delay to 0 clock cycles.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT2:ATT -10"                         'Set the attenuator to -10 dB.
CALL IBWRT(DEV%, CMD$)
```

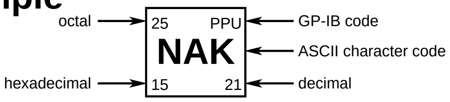
```
'
CMD$ = ":OUTPUT2:PHASE 45"                    'Set the phase to 45 degrees.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT2:LVERROR -10"                 'Set the I/Q gain to 10%.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT2:PSERROR -20"                 'Set the quadrature offset to -20
degrees.
CALL IBWRT(DEV%, CMD$)
'
CMD$ = ":OUTPUT2:FILTER F30MHZ"               'Set the filter to 30 MHz.
CALL IBWRT(DEV%, CMD$)
'
'
CMD$ = ":START"                               'Start waveform output.
CALL IBWRT(DEV%, CMD$)
'
```

Appendix 1 ASCII Character Codes

The following table shows the ASCII character codes.

	0	1	2	3	4	5	6	7
0	0 NUL 0 0	20 DEL 10 16	40 0 SP 20 32	60 16 0 30 48	100 0 @ 40 64	120 16 P 50 80	140 0 ‘ 60 96	160 16 p 70 112
1	1 GTL SOH 1 1	21 LLO DC1 11 17	41 1 ! 21 33	61 17 1 31 49	101 1 A 41 65	121 17 Q 51 81	141 1 a 61 97	161 17 q 71 113
2	2 STX 2 2	22 DC2 12 18	42 2 ” 22 34	62 18 2 32 50	102 2 B 42 66	122 18 R 52 82	142 2 b 62 98	162 18 r 72 114
3	3 ETX 3 3	23 DC3 13 19	43 3 # 23 35	63 19 3 33 51	103 3 C 43 67	123 19 S 53 83	143 3 c 63 99	163 19 s 73 115
4	4 SDC EOT 4 4	24 DCL DC4 14 20	44 4 $ 24 36	64 20 4 34 52	104 4 D 44 68	124 20 T 54 84	144 4 d 64 100	164 20 t 74 116
5	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 5 % 25 37	65 21 5 35 53	105 5 E 45 69	125 21 U 55 85	145 5 e 65 101	165 21 u 75 117
6	6 ACK 6 6	26 SYN 16 22	46 6 & 26 38	66 22 6 36 54	106 6 F 46 70	126 22 V 56 86	146 6 f 66 102	166 22 v 76 118
7	7 BEL 7 7	27 ETB 17 23	47 7 ’ 27 39	67 23 7 37 55	107 7 G 47 71	127 23 W 57 87	147 7 g 67 103	167 23 w 77 119
8	10 GET BS 8 8	30 SPE CAN 18 24	50 8 (28 40	70 24 8 38 56	110 8 H 48 72	130 24 X 58 88	150 8 h 68 104	170 24 x 78 120
9	11 TCT HT 9 9	31 SPD EM 19 25	51 9) 29 41	71 25 9 39 57	111 9 I 49 73	131 25 Y 59 89	151 9 i 69 105	171 25 y 79 121
A	12 LF A 10	32 SUB 1A 26	52 10 * 2A 42	72 26 : 3A 58	112 10 J 4A 74	132 26 Z 5A 90	152 10 j 6A 106	172 26 z 7A 122
B	13 VT B 11	33 ESC 1B 27	53 11 + 2B 43	73 27 ; 3B 59	113 11 K 4B 75	133 27 [5B 91	153 11 k 6B 107	173 27 { 7B 123
C	14 FF C 12	34 FS 1C 28	54 12 , 2C 44	74 28 < 3C 60	114 12 L 4C 76	134 28 \ 5C 92	154 12 l 6C 108	174 28 | 7C 124
D	15 CR D 13	35 GS 1D 29	55 13 - 2D 45	75 29 = 3D 61	115 13 M 4D 77	135 29] 5D 93	155 13 m 6D 109	175 29 } 7D 125
E	16 SO E 14	36 RS 1E 30	56 14 . 2E 46	76 30 > 3E 62	116 14 N 4E 78	136 30 ^ 5E 94	156 14 n 6E 110	176 30 ~ 7E 126
F	17 SI F 15	37 US 1F 31	57 15 / 2F 47	77 UNL ? 3F 63	117 15 O 4F 79	137 UNT _ 5F 95	157 15 o 6F 111	177 DEL (RUBOUT) 7F 127
	Address Command	**Universal Command**	**Listener Address**		**Talker Address**		**Secondary Command**	

Example



Appendix 2 Error Messages

This section describes the error messages related to communications.

- All error messages are displayed in English.
- If servicing is required, please contact your nearest YOKOGAWA dealer.
- Only error messages related to communications are listed here. For other error messages, see User's Manual IM 703150-01E.
- You can use the [STATus:ERRor?] command to query the error that occurred.

Error in communication command

Code	Messages	Description and Corrective Action	Page
102	Syntax error	There is a syntax error other than the codes listed below.	Chapter 3, Chapter 4
103	Invalid separator	<DATA SEPARATOR> is missing. Use a comma to separate the data.	3-2
104	Data type error	The <DATA> type is not correct. Write using the correct data form.	3-5, 3-6
105	GET not allowed	Device trigger function cannot be used. GET is not supported for responses to interface messages.	
108	Parameter not allowed	There are too many <DATA>. Check the number of data points.	3-5, Chapter 4
109	Missing parameter	Required <DATA> is missing. Write the required data.	3-5, Chapter 4
111	Header separator error	<HEADER SEPARATOR> is missing. Use a space to separate the header and data.	3-2
112	Program mnemonic too long	<mnemonic> is too long. Check the mnemonic (alphanumerical character string).	Chapter 4
113	Undefined header	No such command. Check the header.	Chapter 4
114	Header suffix out of range	The value of <HEADER> is not correct. Check the header.	Chapter 4
120	Numeric data error	The mantissa of the value is missing. A mantissa is required before the exponent in the <NRf> form.	3-5
123	Exponent too large	The exponent is too large. Make the exponent after "E" smaller in the <NR3> form.	3-5, Chapter 4
124	Too many digits	There are too many significant digits. The value must be less than equal to 255 digits.	3-5, Chapter 4
128	Numeric data not allowed	Numerical data cannot be used. Write in a data form other than the <NRf> form.	3-5, Chapter 4
131	Invalid suffix	The unit is not correct. Check the unit of the <Voltage> and <Time>.	3-5
134	Suffix too long	The spelling of the unit is too long. Check the unit of the <Voltage> and <Time>.	3-5
138	Suffix not allowed	Units cannot be used. Units other than those for <Voltage> and <Time> cannot be used.	3-5
141	Invalid character data	No such selection available. Select character data from the selections available in {...|...|...}.	Chapter 4
144	Character data too long	The spelling of <CHARACTER DATA> is too long. Check the spelling of the character strings in {...|...|...}.	Chapter 4

Code	Messages	Description and Corrective Action	Page
148	Character data not allowed	<CHARACTER DATA> cannot be used. Write in a data form other than {...|...|...}.	Chapter 4
150	String data error	There is no delimiter to the right of <STRING DATA>. Enclose <String data> in double quotation or single quotation marks.	3-6
151	Invalid string data	The contents of <STRING DATA> are inappropriate. <String data> is too long or invalid character is present.	Chapter 4
158	String data not allowed	<STRING DATA> cannot be used. Write in a data form other than <String data> form.	Chapter 4
161	Invalid block data	The data length of <BLOCK DATA> does not match. <Block data> cannot be used.	3-6, Chapter 4
168	Block data not allowed	<BLOCK DATA> cannot be used. <Block data> cannot be used.	Chapter 4
171	Invalid expression	There is an invalid character in the <EXPRESSION DATA>. Equations cannot be used.	Chapter 4
178	Expression data not allowed	<EXPRESSION DATA> cannot be used. Equations cannot be used.	Chapter 4
181	Invalid outside macro definition	The placeholder is outside the macro. Macro functions defined in IEEE488.2 are not supported.	–

Error in communication execution

Code	Messages	Description and Corrective Action	Page
221	Setting conflict	There is a conflict in the setup information. Check the relevant setting values.	Chapter 4
222	Data out of range	The value of <DATA> is outside the range. Check the range.	Chapter 4
223	Too much data	The length of <DATA> is too long. Check the length of the data.	Chapter 4
224	Illegal parameter value	The value of <DATA> is inappropriate. Check the range.	Chapter 4
241	Hardware missing	The hardware is not implemented. Check the existence of options.	–
260	Expression error	<EXPRESSION DATA> is not correct. Equations cannot be used.	–
270	Macro error	Macro nesting is too deep. Macro functions defined in IEEE488.2 are not supported.	–
272	Macro execution error	Macros cannot be used. Macro functions defined in IEEE488.2 are not supported.	–
273	Illegal macro label	The macro label is inappropriate. Macro functions defined in IEEE488.2 are not supported.	–
275	Macro definition too long	The macro is too long. Macro functions defined in IEEE488.2 are not supported.	–
276	Macro recursion error	Macro was recursively called. Macro functions defined in IEEE488.2 are not supported.	–
277	Macro redefinition not allowed	Macros cannot be redefined. Macro functions defined in IEEE488.2 are not supported.	–
278	Macro header not found	Such macro is not defined. Macro functions defined in IEEE488.2 are not supported.	–

Error in communication query

Code	Messages	Description and Corrective Action	Page
410	Query INTERRUPTED	Query transmission was aborted. Check the order of transmission and reception.	3-2
420	Query UNTERMINATED	There is no response that can be transmitted. Check the order of transmission and reception.	3-2
430	Query DEADLOCKED	Deadlock occurred. Aborting transmission. Set the length of a program message including the <PMT> to less than or equal to 1024 bytes.	3-2
440	Query UNTERMINATED after indefinite response	The order to request the response is not correct. Do not specify a query after the *IDN? or *OPT? command.	–

Error in system operation

Code	Messages	Description and Corrective Action	Page
912	Fatal error in Communication -driver	Communication driver error. Servicing required.	–

Warning

Code	Messages	Description and Corrective Action	Page
5	*OPC/? exists in message	*OPC/? is in the middle of the message. Place the *OPC or *OPC? command at the end of the program message.	–

Miscellaneous

Code	Messages	Description and Corrective Action	Page
350	Queue overflow	Read the error queue. Occurs when there are 16 or more messages in the error buffer.	5-5
390	Over run error (serial interface only)	Lower the baud rate.	

Note

Code “350” occurs when the error queue overflows. This error is output only during a STATus:ERRor? query and does not appear on the screen.

Appendix 3 About the IEEE.488.2-1992 Standard

The GP-IB interface of the instrument conforms to the IEEE 488.2-1992 Standard. This standard specifies that the following 23 points be stated in the document. This section will describe these points.

(1) Of the IEEE 488.1 interface functions, the subsets that are supported

See section 1.4, “GP-IB Interface Specifications.”

(2) The operation of the device when it is assigned an address outside the 0 to 30 range

The address of this instrument cannot be set to an address outside the 0 to 30 range.

(3) Reaction of the device when the user changes the address

The address change occurs when the address is specified using the MISC key menu. The new address is valid until the next time it is changed.

(4) Device settings at power-up. The commands that can be used at power-up.

Basically, the previous settings are used (settings that existed when the power was turned OFF).

All commands can be used at power-up.

(5) Message exchange options

(a) Input buffer size
1024 bytes

(b) Queries that return multiple response units
See the example of the commands given in chapter 4.

(c) Queries that create response data when the command syntax is being analyzed
All queries create response data when the command syntax is analyzed.

(d) Queries that create response data during reception
There are no queries of which the response data are created upon receiving a send request from the controller.

(e) Commands that have parameters the restrict one another
See the example of the commands given in chapter 4.

(6) Items that are included in the functional or composite header elements constituting a command

See chapter 3 and chapter 4.

(7) Buffer sizes that affect block data transmission

During block data transmission, the output queue is expanded according to the size.

(8) A list of program data elements that can be used in equations and their nesting limitations

Equations cannot be used.

(9) Syntax of the responses to queries

See the example of the commands given in chapter 4.

(10) Communication between devices that do not follow the response syntax

None

(11) Size of the response data block

1 to 16000004(4000001Å~4) bytes

(12) A list of supported common commands

See section 4.11, “Common Command Group.”

(13) Device condition after a successful calibration

Same condition as before the execution of calibration.

(14) The maximum length of block data that can be used for the *DDT trigger macro definition

Not supported.

(15) The maximum length of the macro label for defining macros, the maximum length of block data that can be used for the macro definition, and the process when recursion is used in macro definitions

Macro functions are not supported.

(16) Reply to the IDN? query

See section 4.11, “Common Command Group.”

(17) The size of the storage area for protected user data for *PUD and *PUD? commands

*PUD and *PUD? are not supported.

(18) The length of the *RDT and *RDT? resource names

*RDT and *RDT? are not supported.

(19) The change in the status due to *RST, *LRN?, *RCL, and *SAV

*RST

See section 4.11, “Common Command Group.”

*LRN?, *RCL, and *SAV

These common commands are not supported.

(20) The extent of the self-test using the *TST? command

Executes all the MEMORY tests (each internal memory) of the Self Test menu of the MISC key.

(21) The structure of the extended return status

See chapter 5.

(22) Whether each command is processed in an overlap fashion or sequentially

See section 3.5, “Synchronization with the Controller” and chapter 4.

(23) The description of the execution of each command

See the functions of each command in chapter 4 and User’s Manual IM703150-01E.

Index

Symbols

10-MHz reference signal 4-5

A

ASCII character code App-1
address 1-5
address command 1-7
all load 4-21
attenuator 4-17
auto load 4-25

B

backlight 4-25
baud rate 2-8
block data 3-6
body file 4-21
boolean 3-6

C

calibration 4-13, 4-26
character data 3-6
character string data 3-6
clock 4-5
clock frequency 4-5
clock input 4-5
condition register 4-23
connector 2-3

D

DCL 1-6
data 3-3, 3-5
data format 2-7
data length 2-8
data transfer 4-13
delay 4-17
delete file 4-9
digital out 4-17
directory creation 4-11

E

error 4-24
error message App-2
error queue 5-5
export 4-9
extended event enable register 4-23
extended event register 5-4

F

file 4-9
format 4-10
functional specification 1-4

G

GET 1-6
GP-IB cable 1-2
GP-IB connector 1-1
GTL 1-6

H

handshake 2-5, 2-8
header 4-6
header file 4-21
host name 4-15

I

I/Q gain ratio 4-19
IFC 1-6
IP address 4-14
import 4-10
interface message 1-6

L

LLO 1-6
LOCAL key 1-1, 2-1
label 4-19
list of command 4-1
listener function 1-3
load 4-11, 4-19
local 4-7
local lockout 4-6
low-pass filter 4-19

M

MAC address 4-14
MISC key 1-1, 2-1
message 3-1
model 4-27
multi-line message 1-6, 1-7

O

OUTPUT 4-17
offset 4-19
operation pending status register 4-7
output queue 5-5

P

parity bit 2-8
phase 4-19
program message 3-1

Q

quadrature offset 4-19
query 3-5

R

REMOTE indicator 1-1, 2-1
REN 1-6
RS-232 2-1
reception function 2-2
remote 4-7
response message 3-1

S

SDC 1-6
SPD 1-6
SPE 1-6
save 4-13
self-test 4-28
serial interface connector 2-1
serial poll 5-1
signal name 2-3
specification 1-4, 2-2
standard event register 5-3
start trigger 4-5
status byte 5-2
status byte register 4-28
status report 5-1
stop bit 2-8

T

TCP/IP 4-15
talker function 1-3
transition filter 4-24
transmission function 2-2
trigger 4-5

U

uni-line message 1-6
universal command 1-7
upload 4-13

W

waveform data 4-19
waveform generation start 4-22
waveform generation stop 4-24
waveform group list 4-19
waveform output ON/OFF 4-25
wiring example 2-4