MT9810A Optical Test Set Remote Control Operation Manual

Third Edition

To ensure that the MT9810A Optical Test Set is used safely, read the safety information in the MS9710A Optical Test Set Manual first. Keep this manual with the Optical Test Set.

ANRITSU CORPORATION

Document No.: M-W1483AE-3.0

MT9810A　Optical Test Set Remote Control
Operation Manual

14	July	1998	(First Edition)
20	February	2003	(Third Edition)

Trademark

Introduction

This manual describes the remote control of the MT9810A Optical Test Set. This product can control the MT9810A and incorporate the measurement result through the GPIB/RS-232C interface.

Table of Contents

Section 1 Overview

This section outlines the remote control functions of the MT9810A Optical Test Set.

1.1 Overview

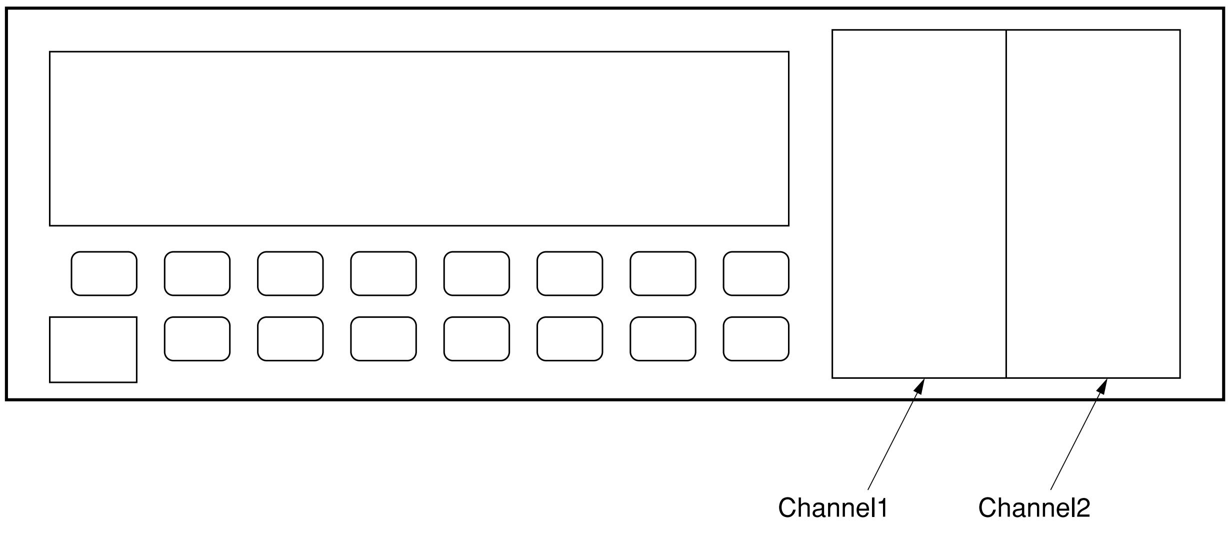
The MT9810A Optical Test Set can perform almost all operations remotely using a computer. This product comes standardized with a GPIB interface port (IEEE Std 488.2-1987) and an RS-232C interface port.

1.2 Selecting the Interface Port

The interface port is selected from the front panel of the MT9810A main unit. The two ports cannot be used at the same time. Refer to the Section 2 "How to Connect" for more details.

1.3 Channel Numbers of the Unit

Up to two units can be mounted on the MT9810A. There are commands that specify the channel number to which the unit is mounted. The left channel is Channel 1 and the right channel is Channel 2 as seen from the front.



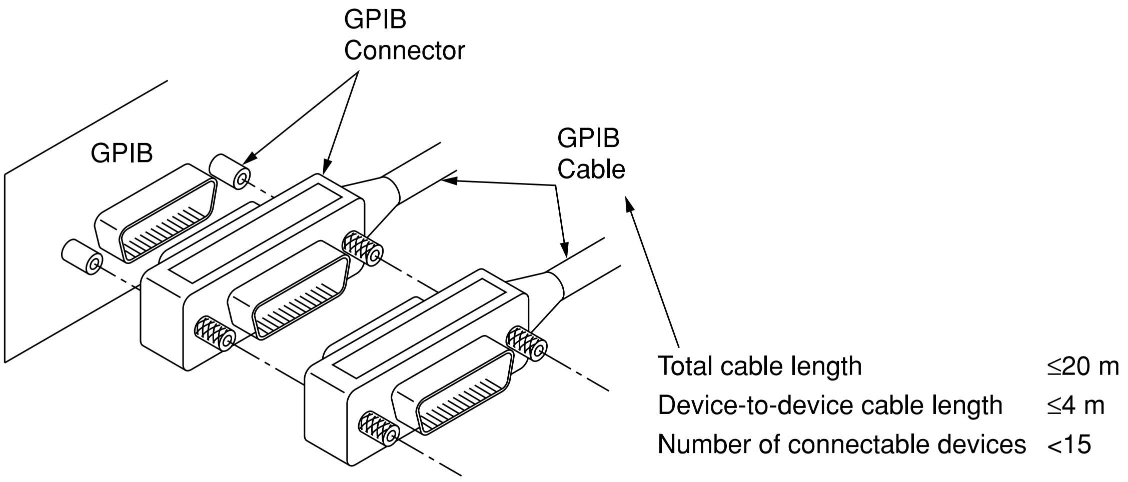
Section 2 How to Connect

This section explains how to connect GPIB and RS-232C cables between the MT9810A Optical Test Set and external devices such as a host computer, personal computer, and printer. This section also explains how to set the interfaces of the MT9810A.

2.1 Connecting Device Using a GPIB Cable

The MT9810A has a GPIB cable connector mounted on the back panel. Be sure to connect the GPIB cable before turning on the power.

A maximum of 15 devices, including a controller, can be connected to one system. Connect these device in accordance with the conditions shown in the following figure.



Total cable length	≤20 m
Device-to-device cable length	≤4 m
Number of connectable devices	<15

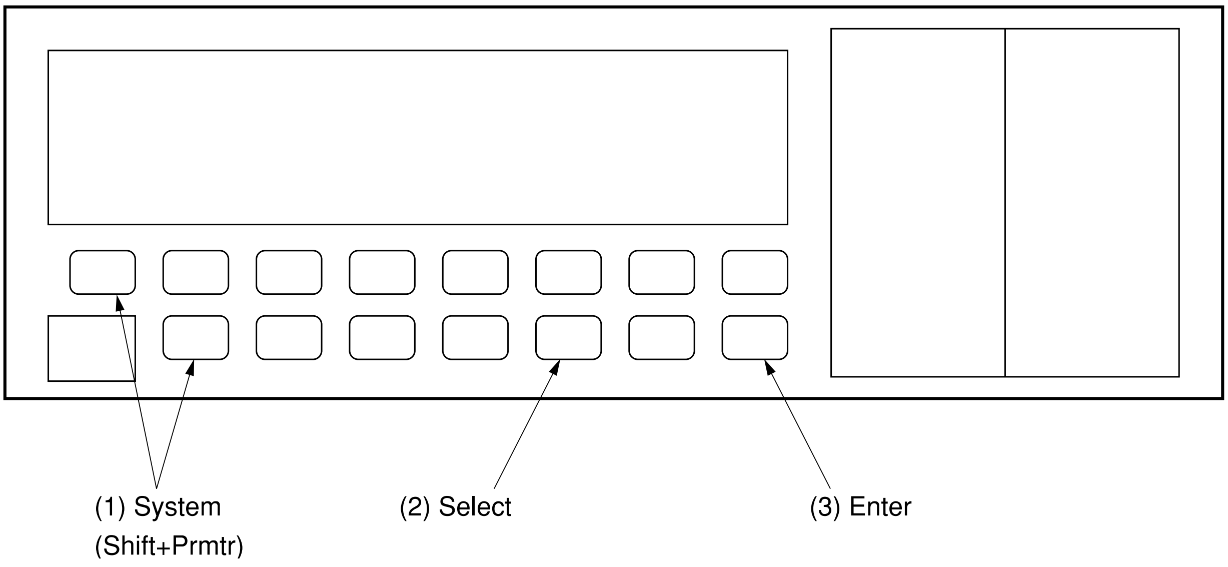
2.1.1 Setting the Interface for the Connection Port

Set the interface of the connection port to GPIB. The setting method is shown below.

(1) Select "Remote Interface" with the System key.

(2) Switch to "GPIB" with the Select key.

(3) Enter the setting by pressing the Enter key.



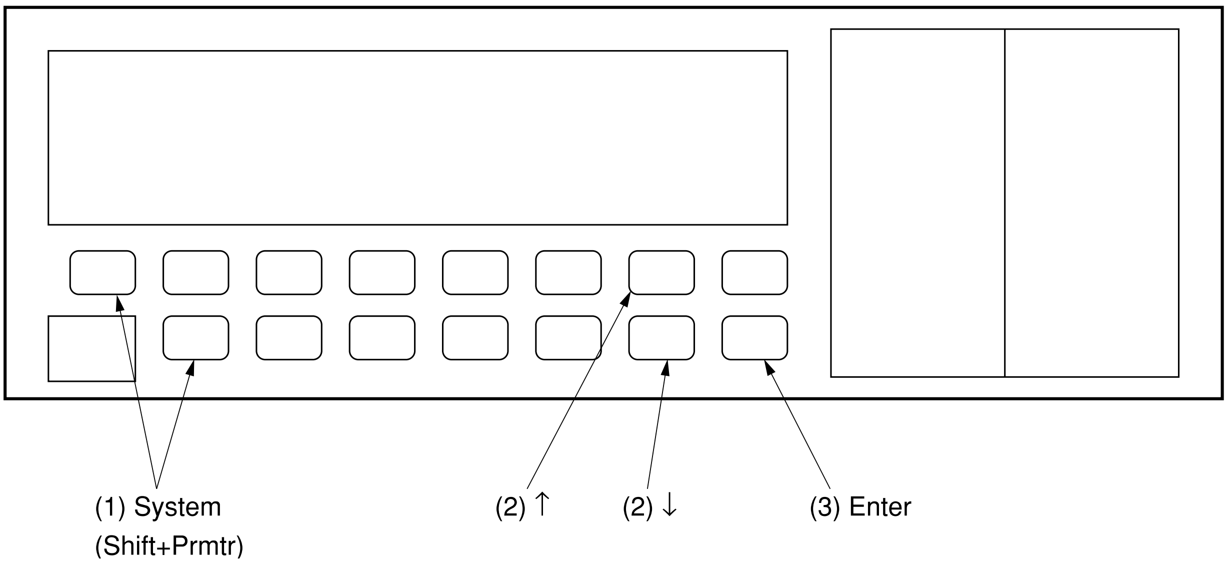
2.1.2 Confirming and Setting the Address

Be sure to set the GPIB address of the MT9810A after turning on the power. Set the address using the front panel with the MT9810A set to the local mode.

(1) Select "GPIB ADDRESS" with the System key.

(2) Specify the address with the ↑ and ↓ keys. (The input address range is from 0 to 30.)

(3) Enter the setting by pressing the Enter key.



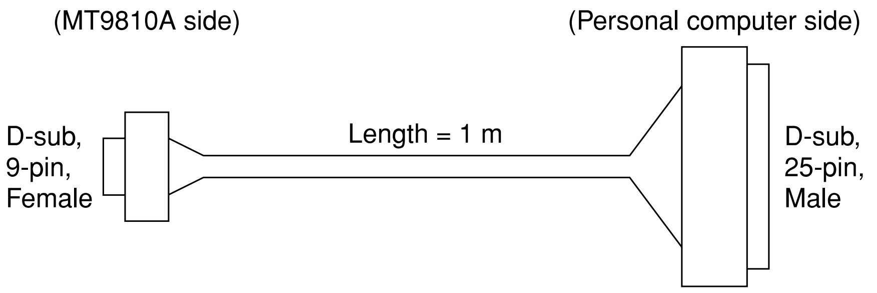
2.2 Connecting a Device Using an RS-232C Cable

The MT9810A has an RS-232C connector mounted on the back panel.

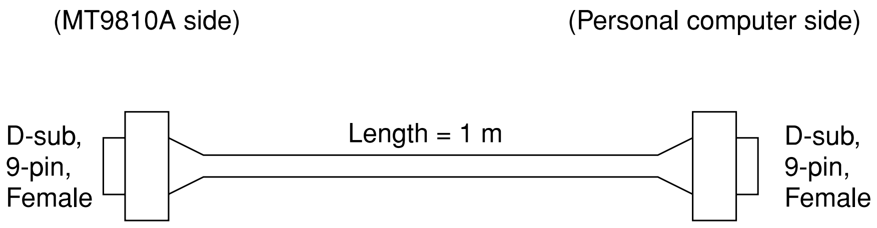
NOTE:

RS-232C connectors are available in 9-pin and 25-pin types. The 9-pin type is usually used for DOS/V personal computers, while the 25-pin type is usually used for the NEC PC9801/PC9821 Series. Before purchasing an RS-232C cable, check the type of the RS-232C connector on the external device. The following two types of RS-232C cables are available as application parts for this product.

• RS-232C cable (for 25-pin type personal computer)

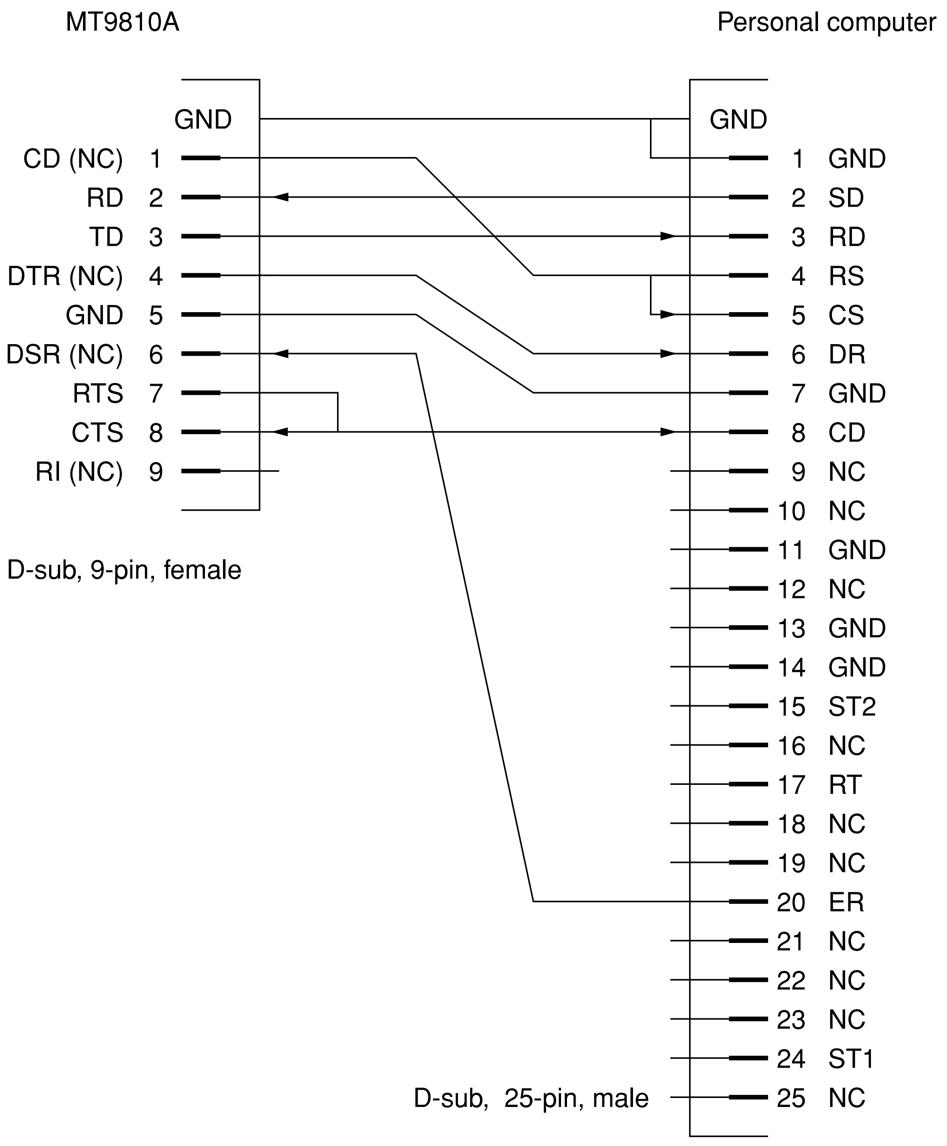


• RS-232C cable (for DOS/V personal computer)

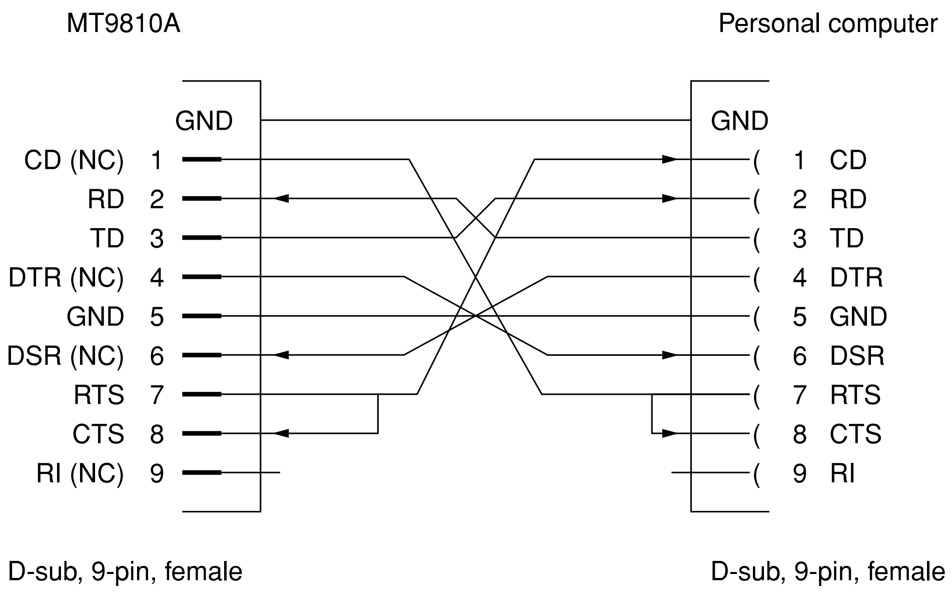


2.2.1 RS-232C Interface Signal Connection Diagrams

The following diagram shows the connection of RS-232C interface signals between the MT9810A and a personal computer.



Connection to the external computer with a D-sub 25-pin interface



Connection to the DOS/V personal computer

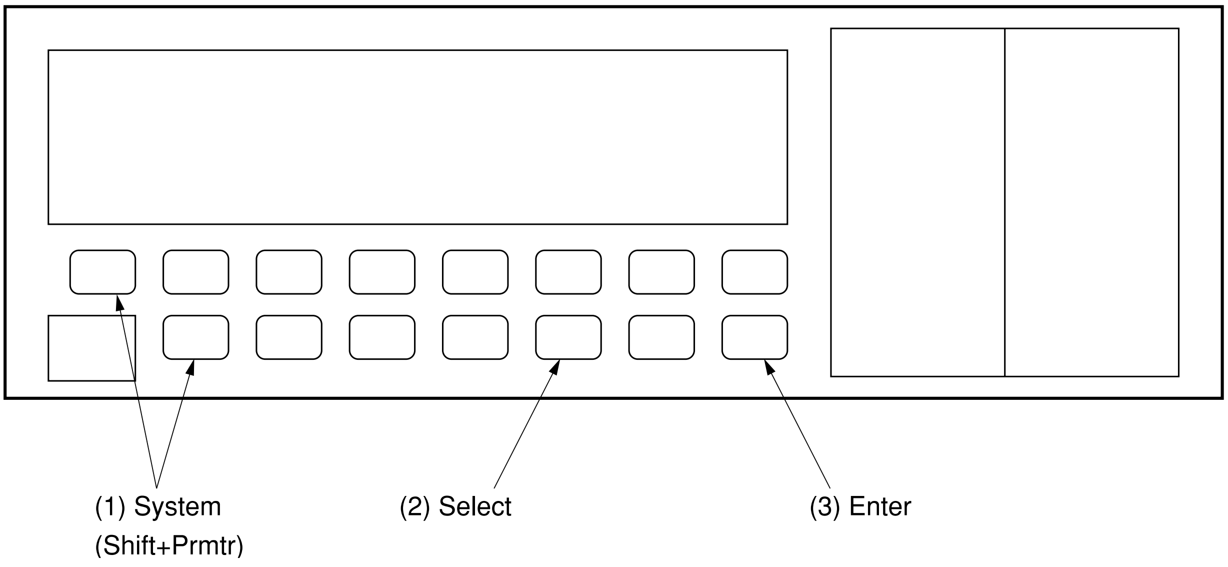
2.2.2 Setting the Interface of the Connection Port

Set the interface of the connection port to RS-232C. The setting method is shown below.

(1) Select "Remote Interface" with the System key.

(2) Switch the interface to "RS-232C" with the Select key.

(3) Enter the setting by pressing the Enter key.



2.2.3 Setting RS-232C Interface Conditions

Set the interface conditions for the RS-232C port of MT9810A to match the interface conditions of the connected external device. The setting method is shown below.

(1) Select the setting items with the System key.

(2) Specify the setting values with the Select key.

(3) Enter the setting by pressing the Enter key.

The setting items are shown in the Table 2-1.

Table 2-1

Item	System key	Setting value
Baud rate	RS-232C Baudrate	1200/2400/4800/9600/14400/19200 bps
Stop bit	RS-232C StopBit	1/2 bit
Parity bit	RS-232C ParityBit	ODD/EVEN/NONE
Character length	RS-232C Character	7/8 bit

2.3 Default Value

The factory-set values are shown in the Table 2-2.

Table 2-2

Setting item	Default value
Remote interface	GPIB
GPIB address	15
RS-232C baud rate	9600 bps
RS-232C stop bit	1 bit
RS-232C parity bit	Even
RS-232C character length	8 bits

Section 3 Specifications

This section explains the GPIB standard, RS-232C standard, and device message list of the MT9810A Optical Test Set.

3.1 GPIB Specifications

The GPIB Specifications of the MT9810A is summarized in the Table 3-1.

Table 3-1

Item	Specifications value and description
Function	Conforms to IEEE 488.2. MT9810A can be controlled from an external controller.
Interface functions	SH1: All of source handshake functions are supported. Data send timing is controlled. AH1: All of acceptor handshake functions are supported. Data receive timing is controlled. T6: Basic talker functions are supported. A serial port function is supported. A talk-only function is not supported. The function of releasing the talker with MLA is supported. L4: Basic listener functions are supported. A listen-only function is not supported. The function of releasing the listener by MTA is supported. SR1: All of service request/status byte functions are supported. RL1: All of remote/local functions are supported. A local lockout function is supported. PP0: A parallel poll function is not supported. DC1: All of device clear functions are supported. DT0: A disk trigger function is not supported. C0: A controller function is not supported. A controller function is performed during external plot output.

3.2 RS-232C Specifications

The RS-232C Specifications of the MT9810A is summarized in the Table 3-2.

Table 3-2

Item	Specifications
Function	Control from external controller
Communication method	Asynchronous (start-stop), half-duplex
Communication control method	No flow control
Baud rate	1200, 2400, 4800, 9600, 14400, 19200 bps
Data bits	7 bits, 8 bits
Parity	Odd parity (ODD), even parity (EVEN), non-parity (NON)
Start bits	1 bit
Stop bits	1 bit, 2 bits
Connector	D-sub 9-pin connector, male

3.3 Device Message List

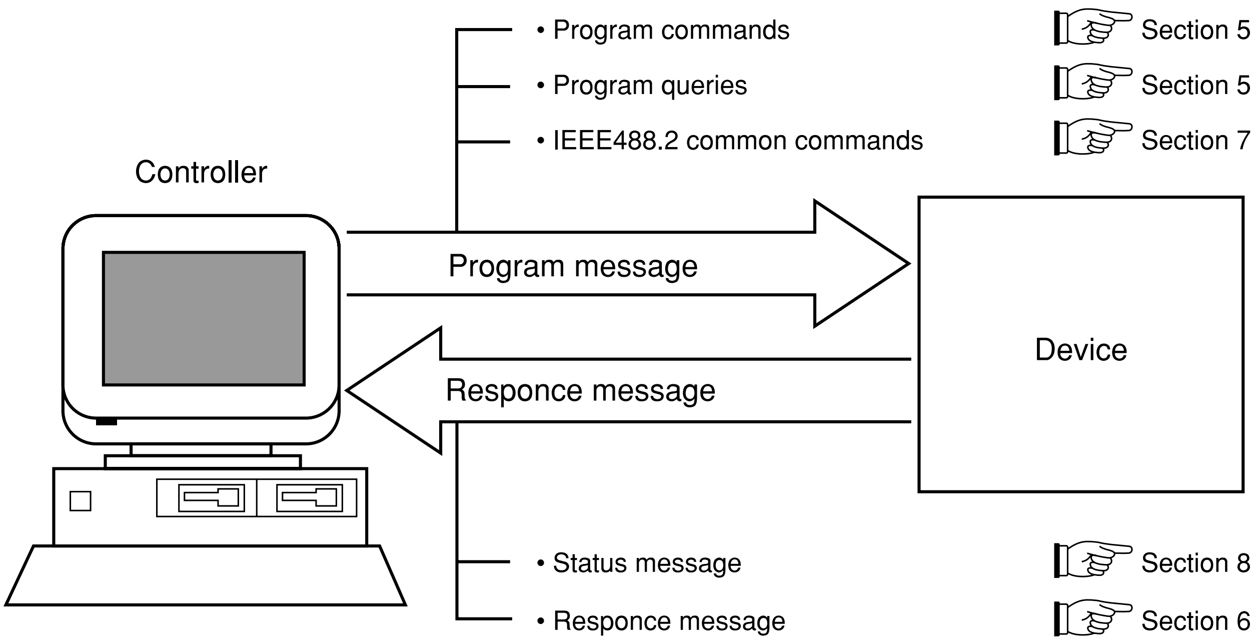
Device messages are data messages which are transferred between a controller and the devices. These messages are classified into program messages and response messages.

Program messages are ASCII messages transferred from a controller to the devices. Program messages are further classified into program commands and program queries. These two types of commands are explained later in this manual.

Program commands include device-dependent commands which are exclusively used for controlling the MT9810A and IEEE 488.2 common commands. IEEE 488.2 common commands are program commands which are commonly applicable to other IEEE 488.2-ready measuring instruments (including the MT9810A) on the GPIB interface bus.

Program queries are commands used to get response messages from devices. Program queries must be transferred from a controller to a device in advance so that the controller can receive response messages from the device later.

Response messages are ASCII data messages which are transferred from a device to a controller. Among response messages, status messages, and response messages corresponding to program queries are listed later in this manual.



In program and response messages, numeric data may end with a suffix (unit).

The above messages are transferred through the device input/output buffer. The output buffer is also called an output queue. A brief description of the output buffer is given below.

Input buffer

Input buffer is an FIFO (first in first out) type memory area, that stores DABs (program and query messages) temporarily before analysis of syntax and execution.
The input buffer size of the MT9810A is 256 bytes.

Output queue

Output queue is an FIFO-type queue memory area, that stores all DABs (response messages) output from a device to a controller until those messages are read by the controller.
The output queue size of the MT9810A is 256 bytes.

3.3.1 IEEE 488.2 common commands and the commands supported by the MT9810A

The 39 common commands specified by IEEE 488.2 standard is shown in the Table 3-3. Among these commands, the commands supported by the MT9810A are marked with the check marks (√).

Table 3-3

Mnemonic	Fully spelled out command name	Standardized by IEEE 488.2	Supported by MT9810A
∗ADD	Accept Address Command	Optional	
∗CAL	Calibration Query	Optional	
∗CLS	Clear Status Command	Required	√
∗DDT	Define Device Trigger Command	Optional	
∗DDT?	Define Device Trigger Query	Optional	
∗DLF	Disable Listener Function Command	Optional	
∗DMC	Define Macro Command	Optional	
∗EMC	Enable Macro Command	Optional	
∗EMC?	Enable Macro Query	Optional	
∗ESE	Standard Event Status Enable Command	Required	√
∗ESE?	Standard Event Status Enable Query	Required	√
∗ESR?	Standard Event Status Register Query	Required	√
∗GMC?	Get Macro contents Query	Optional	
∗IDN?	Identification Query	Required	√
∗IST?	Individual Status Query	Optional	
∗LMC?	Learn Macro Query	Optional	
∗LRN?	Learn Device Setup Query	Optional	
∗OPC	Operation Complete Command	Required	√
∗OPC?	Operation Complete Query	Required	√
∗OPT?	Option Identification Query	Optional	√
∗PCB	Pass Control Back Command	Other than C0: Required	
∗PMC	Purge Macro Command	Optional	
∗PRE	Parallel Poll Register Enable Command	Optional	
∗PRE?	Parallel Poll Register Enable Query	Optional	
∗PSC	Power On Status Clear Command	Optional	
∗PSC?	Power On Status Clear Query	Optional	
∗PUD	Protected User Data Command	Optional	
∗PUD?	Protected User Data Query	Optional	
∗RCL	Recall Command	Optional	
∗RDT	Resource Description Transfer Command	Optional	
∗RDT?	Resource Description Transfer Query	Optional	
∗RST	Reset Command	Required	√
∗SAV	Save Command	Optional	
∗SRE	Service Request Enable Command	Required	√
∗SRE?	Service Request Enable Query	Required	√
∗STB?	Read Status Byte Query	Required	√
∗TRG	Trigger Command	DT1: Required	
∗TST?	Self Test Query	Required	√
∗WAI	Wait to Continue Command	Required	√

NOTE:

IEEE 488.2 commands always begin with an asterick (∗). Refer to the Section 7 "Common Commands" for more details.

3.3.2 Device Message List

The device message list unique to the MT9810A is shown in the Table 3-4, 3-5 and 3-6. There are two types of commands: HP commands and SCPI-compliant Anritsu original commands. The types of commands are also shown in the table.

Table 3-4 Main frame

Function	Command	HP	SCPI	Reference
Brightness	DISPlay:BRIGhtness	√		Section 9.1.1
Display ON/OFF	DISPlay[:STATe]	√		Section 9.1.2
Calendar	SYSTem:DATE	√		Section 9.1.7
Time	SYSTem:TIME	√		Section 9.1.9
Buzzer	SYSTem:BEEPer:STATe		√	Section 9.1.3
Header	SYSTem:COMMunicate:GPIB:HEAD		√	Section 9.1.5
	SYSTem:COMMunicate:SERial:HEAD		√	Section 9.1.6
Inserted unit	SYSTem:CHANnel:STATe		√	Section 9.1.4
Error	SYSTem:ERRor		√	Section 9.1.8

Table 3-5 Optical sensor

Function	Command	HP	SCPI	Reference
Zero-set	SENSe[1|2]:CORRection:COLLect:ZERO	√		Section 9.2.6
Calibration factor	SENSe[1|2]:CORRection[:LOSS:[:INPut[:MAG	√		Section 9.2.7
Auto range	Nitude]]]	√		Section 9.2.17
Manual range	SENSe[1|2]:POWer:RANGe:AUTO	√		Section 9.2.18
Reference value	SENSe[1|2]:POWer:RANGe:[UPPer]	√		Section 9.2.19
Displays the reference value	SENSe[1|2]:POWer:REFerence	√		Section 9.2.20
Reference measurement	SENSe[1|2]:POWer:REFerence:DISPlay	√		Section 9.2.21
Reference selection	SENSe[1|2]:POWer:REFernce:STATe	√		Section 9.2.22
Unit	SENSe[1|2]:POWer:REFernce:STATe:RATio	√		Section 9.2.23
Wavelength	SENSe[1|2]:POWer:UNIT	√		Section 9.2.24
Unit of wavelength	SENSe[1|2]:POWer:WAVelength	√		Section 9.2.25
Measurement data	SENSe[1|2]:POWer:WAVelength:UNI	√		Section 9.2.2
The number of averaging	FETCh[1|2][:SCALar]:POWer[:DC]		√	Section 9.2.3
Auto bandwidth	SENSe[1|2]:AVERage:COUNt		√	Section 9.2.5
Bandwidth	SENSe[1|2]:BANDwidth:AUTO		√	Section 9.2.4
Modulation frequency	SENSe[1|2]:BANDwidth		√	Section 9.2.11
Measurement interval	SENSe[1|2]:FILTer:BPASs:FREQuency		√	Section 9.2.16
The number of measurement	SENSe[1|2]:POWer:INTerval		√	Section 9.2.26
Logging	SENSe[1|2]:TRIGger:COUNt		√	Section 9.2.12
Statistical measurement	SENSe[1|2]:INITiate[:IMMediate]		√	Section 9.2.27
Measurement stop	SENSe[1|2]:TRIGger[:SEQuence][:IMMediate]		√	Section 9.2.1
Logging data	ABORt[1|2]		√	Section 9.2.14
Logging data information	SENSe[1|2]:MEMory:DATa		√	Section 9.2.15
Maximum value	SENSe[1|2]:MEMory:DATa:INFO		√	Section 9.2.8
Minimum value	SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MAXimum		√	Section 9.2.9
Difference between maximum and minimum values	SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MINimum SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:PTPeak		√	Section 9.2.10
Measurement conditions	SENSe[1|2]:MEMory:COPY[:NAME]		√	Section 9.2.13
High-speed transfer mode start	READ[1|2]		√	Section 9.2.28
High-speed transfer mode stop	READ[1|2]:ABORt		√	Section 9.2.29

Table 3-6 Light source

Function	Command	HP	SCPI	Reference
Modulation frequency	SOURce[1|2]:AM[:INTerval]:FREQuency	√		Section 9.3.1
Attenuation	SOURce[1|2]:POWer:ATTenuation	√		Section 9.3.3
Optical output	SOURce[1|2]:POWer:STATe	√		Section 9.3.4
Wavelength	SOURce[1|2]:POWer:WAVelength	√		Section 9.3.5
Unit of wavelength	SOURce[1|2]:POWer:WAVelength:UNIT		√	Section 9.3.6
Measurement condition	SOURce[1|2]:MEMory[1|2]:COPY[:NAME]		√	Section 9.3.2

In the portion described as [1|2], enter the channel number into which the target unit is inserted (1 or 2). The brackets ([ ]) are not required.
When you send the LIGHT SOURCE COMMAND to OPTICAL SENSOR, the command error occurs.
At the opposite case (send the OPTICAL SENSOR COMMAND to LIGHT SOURCE), the command error occures too.

Section 4   Initial Setting

Initialization of the GPIB interface system is devided into three levels. At level 1, "bus initialization" is performed to place the system bus in the idle state. At level 2, "message exchange initialization" is performed to enable devices to receive program messages. At level 3, "device initialization" is performed to initialize device-dependent functions. At these three initialization levels, preparations are made for starting devices.

IEEE 488.2 specifies the initialization of the GPIB system as described in the Table 4-1.

Table 4-1

Level	Initialization type	Overview	Combination and priority of levels
1	Bus initialization	Interface functions of all devices connected to the bus are initialized by an IFC message from a controller.	This level may be combined with other levels. However, initialization at level 1 must be performed before initialization at other levels.
2	Message exchange initialization	Message exchange is initialized and the function of reporting completion of operation to the controller is disabled. This initialization can be ferformed either for all devices on the GPIB using GPIB bus command DCL, or only for the specified devices using a GPIB bus command SDC.	This level may be combined withother levels. However, initialization at level 2 must be performed before initialization at level 3.
3	Device initialization	Only the specified devices on the GPIB are initialized to the known states with an *RST command irrespective of the past use state.	This level may be combined with other levels. However, initialization at level 3 must be performed after initialization at levels 1 and 3.

When controlled from a controller via the RS-232C interface port, the MT9810A can use the "device initialization" function (level 3). However, it cannot use "bus initialization" (level 1) and "message exchange initialization" (level 2) functions.
When controlled from a controller via a GPIB interface bus, the MT9810A can use all the above initialization functions (levels 1 to 3).

4.1 Initialization of Bus by IFC Statement

(1) Format

IFC Δ select-code

(2) Explanation

This function can be used when the MT9810A is controlled from a controller via a GPIB interface bus.
On the GPIB corresponding to the specified select code, the IFC line is activated for about 100 μs (as electrically set at the low level). When IFC is executed, interface functions of all devices connected to the GPIB bus line corresponding to the specified select code are initialized. Only the system controller can send this command.

"Initialization of interface functions" refers to the processing in which controller-set device interface functions (talker, listener, etc.) are reset to their initial states. Functions marked with the check marks (√) in the following table are initialized. The function marked with a triangle (Δ) is initialized partially.

Table 4-2

No	Function	Symbol	Initialization by IFC
1	Source handshake	SH	√
2	Acceptor handshake	AH	√
3	Talker or extended talker	T or TE	√
4	Listener or extended listener	L or LT	√
5	Service request	SR	Δ
6	Remote/local	RL	
7	Parallel/poll	PP	
8	Device clear	DC	
9	Device trigger	DT	
10	Controller	C	√

If the IFC statement is True (the IFC line is set at the low level through execution of the IFC statement), initialization is not performed at levels 2 and 3. Therefore, device operating states are not affected.

The examples of device states set by the IFC statement are shown in the Table 4-3.

Table 4-3

Item	Device state
Talker/listener	All talkers and listeners are set in the idle state (TIDS, LIDS) within 100 µs.
Controller	If the controller is not active (SACS: System control ACtive State), it enters the idle state, or CIDS, (Controller IDle State) within 100 µs.
Return of control right	If the system controller (the first device on the GPIB which is used as a controller) has granted the control right to another device when IFC is executed, the control right is returned to the system controller. Generally, pressing the RESET key on the system controller allows an IFC message to be output from the system controller.
Devices issuing service request	The state in which an SRQ message is issued by a device (the SRQ line is set at the LOW level by the device) is not canceled, but the state in which all devices on the system bus are placed in the serial poll mode by the controller is canceled.
Devices in remote state	For the devices currently in the remote state, the remote state is not canceled by the IFC message.

4.2 Initialization of Message Exchange by DCL and SDC Bus Commands

(1) Format

DCL ∆ select-code [primary-address] [secondary-address]

(2) Explanation

This function can be used when the MT9810A is controlled by a controller via the GPIB interface bus.
This statement initializes message exchange for all device on the GPIB corresponding to the specified select code or only for the specified devices.

The purpose of message exchange is to allow the controller to send new commands when the controller cannot control message-exchange-related parts inside the devices due to execution of programs although it is not necessary to change the panel settings.

(3) When only a select code is specified

Message exchange is initialized for all the devices on the GPIB corresponding to the specified select code. DCL issues a DCL (Device Clear) bus command to the GPIB.

(4) When an address is also specified

Message exchange is initialized only for the specified device. Listeners on the GPIB corresponding to the specified select code are canceled, only the specified device is set as a listener, and an SDC (Selected Device Clear) bus command is issued.

(5) Items subject to initialization of message exchange

Table 4-4

Item	Device state
Input buffer and output queue	The settings are cleared.
Syntax analysis, execution control, and response generation parts	The functions are reset.
Device commands including *RST	All commands interfering with execution of these commands are cleared.
Paired parameter/program message	All commands and queries of which execution has been suspended due to paired parameters are discarded.
*OPC command processing	The specified device is set in the OCIS (Operating Complete Command Idle State). The operation complete bit cannot be set in the standard event status register. Section 7
*OPC? query processing	The specified device is set in the OQIS (Operating Complete Query Idle State). The operation complete bit 1 cannot be set in the output queue. The MAV (Message Available) bit is cleared. Section 7
Automatic system configuration	*ADD and *DLF common commands are invalidated. (The MT9810A does not support these commands.)
Device function	All parts related to message exchange are set in the idle state. The device waits for a message from the controller.

The following operations using DCL are prohibited.

(a) Changing the current device settings and stored data
(b) Interrupting front panel I/O
(c) Changing status bits other than the MAV bit when clearing the output queue
(d) Affecting or interrupting the device operation currently being performed

(6) Orders of issuing GPIB bus commands using DCL statements

Orders of issuing GPIB bus commands using DCL, SDC statements are summarized in the Table 4-5.

Table 4-5

Statement	Bus command issue order (ATN line: Low level)	Data (ATN line: High level)
DCLselect-code	UNL, DCL	————
DCLdevice-number	UNL, LISTEN address, [secondary-address], SDC	————

4.3 Initialization of Devices by ∗RST Command

(1) Format

∗RST

(2) Explanation

The ∗RST (Reset) command is one of the IEEE 488.2 common command, which is used to reset a specified device at level 3.

Generally, devices are set in various states using device-dependent commands (device messages). Among these commands, the ∗RST command is used to reproduce a known state of a device. Completion of device operation is invalidated like level 2.

(3) Specification of device number in WRITE statement

The device at the specified address is initialized at level 3.

(4) Items subject to device initialization

Table 4-6

Item	Device state
Device-dependent functions and states	The specified device is set in a known state irrespective of its history. (Refer to the lists on the following pages.)
∗OPC command processing	The specified device is set in the OCIS (Operating Complete Command Idle State). The operation complete bit cannot be set in the standard event status register. ☞ **Section 7**
∗OPC? query processing	The specified device is set in the OQIS (Operating Complete Query Idle State). The operation complete bit 1 cannot be set in the output queue. The MAV (Message Available) bit is cleared. ☞ **Section 7**
Macro command	Macro operation is disabled, and sets the state in which macro commands cannot be accepted. The returnes the macro definitions to the designer's state.

NOTES:

∗RST command does not affect the following items:

1. IEEE 488.1 interface state
2. Device address
3. Output queue
4. Service request enable register
5. Standard event status enable register
6. Power-on-status-clear flag setting
7. Calibration data affecting device standard
8. RS-232C interface condition

4.4 Device States at Power-ON

When the power is turned on:

(1) The MT9810A is restored to the last Power-OFF state.

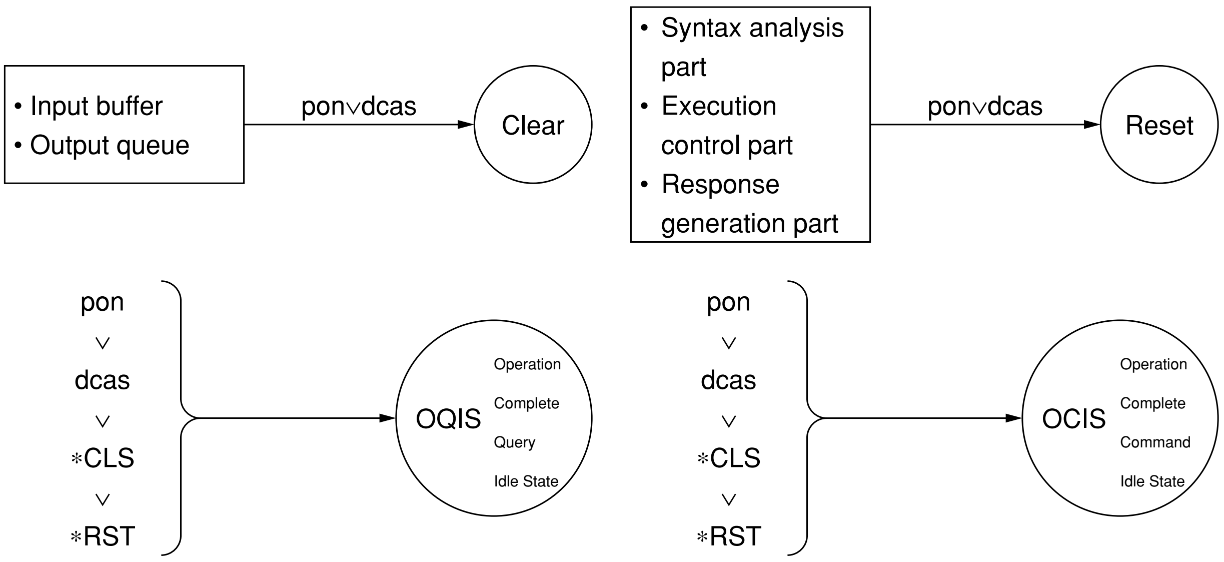
(2) The input buffer and output queue are cleared.

(3) Syntax analysis, execution control, and response generation parts are reset.

(4) The device is set in the OCIS.

(5) The device is set in the OQIS.

(6) The MT9810A does not support a *PSC command. Therefore, the standard event status register and standard event status enable register are cleared.
Events are recorded after being cleared.
States (2) to (5) are set except when the power is turned on. The following diagram describes these states.



4.4.1 Items not changes at Power-ON

(1) Address

(2) Associating calibration data

(3) Data and states are changed by the responses to the following common query commands.

∗IDN? .... Refer to the Section 7 "Common Commands"
∗OPT? .... Refer to the Section 7 "Common Commands"
∗PSC? .... Not supported by the MT9810A
∗PUD? .... Not supported by the MT9810A
∗RDT? .... Not supported by the MT9810A

4.4.2 Items related to PSC flag

When the PSC (Power-ON status clear) flag is False, the service request enable register, standard event status enable register, and parallel poll enable register are not affected. Refer to the Section 8.3 "Enabling the SRQ" for the service request enable register, and refer to the Section 8.4 "Standard Event Status Register" for the standard event status enable register
When the PSC flag is Low level (True) or the ∗PSC command has not been executed, the above registers are cleared.

NOTE:

The PSC command is not supported by the MT9810A.

4.4.3 Items that change at Power-ON

(1) Current device function state

(2) Status information

(3) ∗SAV/∗RCL register (Not supported by the MT9810A)

(4) Macro definition made with a ∗DDT command (Not supported by the MT9810A)

(5) Macro definition made with a ∗DMC command (Not supported by the MT9810A)

(6) Macro enabled with an ∗EMC command (Not supported by the MT9810A)

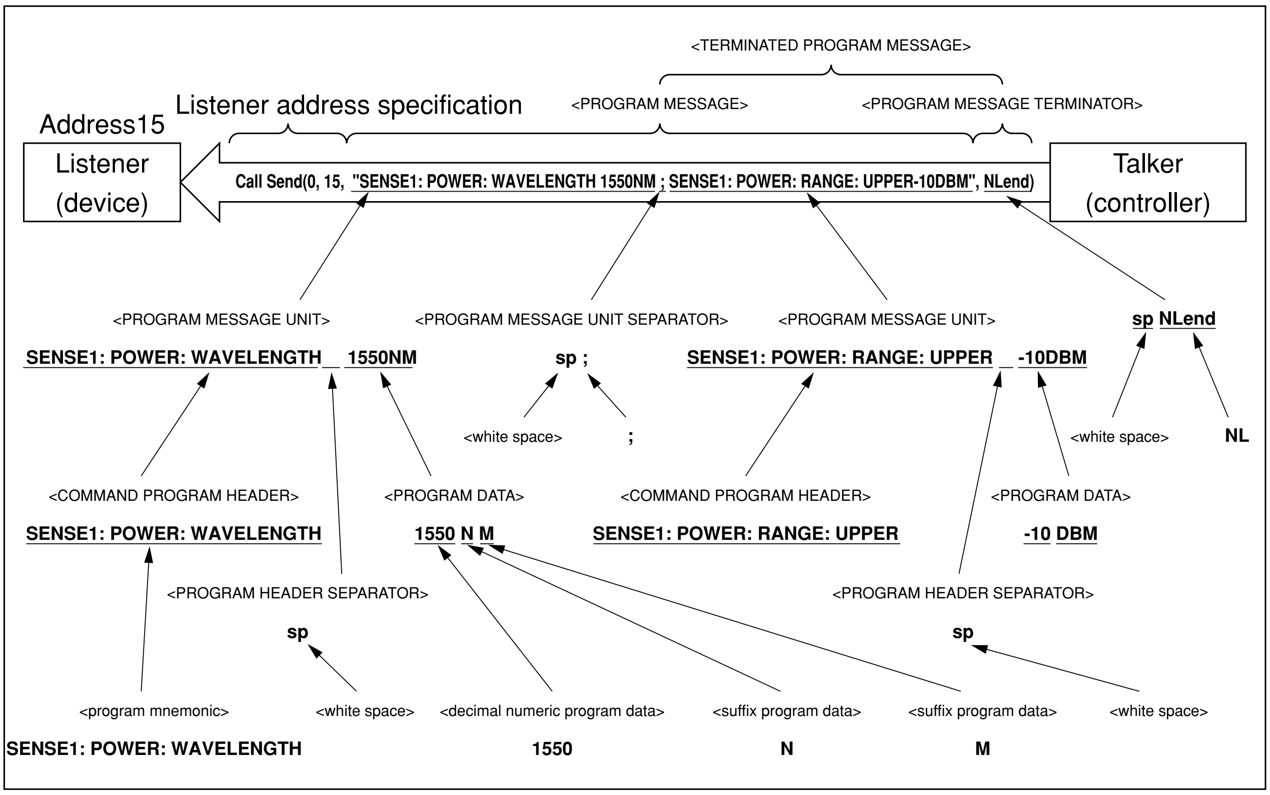
(7) Address received with a ∗PCB command (Not supported by the MT9810A)

Section 5 Listener Input Formats

Device messages are data messages transferred between the controller and devices, which can be classified into program messages and response messages. This section explains the formats of the program messages received by listeners.

A program message is a sequence of program message units. Each unit is a program command or query.

The following diagram shows how to set the wavelength and measurement range of the power meter unit inserted into Channel 1 to 1550 nm and –10 dBm. As it explained in the diagram, two program message units SENSE1:POWER:WAVELENGTH 1550NM and SENSE1:POWER:RANGE:UPPER –10DBM are connected with the program message unit separator and sent to the device from the controller as one program message.


<TERMINATED PROGRAM MESSAGE>
Listener address specification
<PROGRAM MESSAGE>
<PROGRAM MESSAGE TERMINATOR>
Address15
Listener (device)
Call Send(0, 15, "SENSE1: POWER: WAVELENGTH 1550NM ; SENSE1: POWER: RANGE: UPPER-10DBM", NLend)
Talker (controller)
<PROGRAM MESSAGE UNIT>
<PROGRAM MESSAGE UNIT SEPARATOR>
<PROGRAM MESSAGE UNIT>
sp NLend
SENSE1: POWER: WAVELENGTH 1550NM
sp ;
SENSE1: POWER: RANGE: UPPER -10DBM
<white space>
;
<white space>
NL
<COMMAND PROGRAM HEADER>
<PROGRAM DATA>
<COMMAND PROGRAM HEADER>
<PROGRAM DATA>
SENSE1: POWER: WAVELENGTH
1550 N M
SENSE1: POWER: RANGE: UPPER
-10 DBM
<PROGRAM HEADER SEPARATOR>
<PROGRAM HEADER SEPARATOR>
sp
sp
<program mnemonic>
<white space>
<decimal numeric program data>
<suffix program data>
<suffix program data>
<white space>
SENSE1: POWER: WAVELENGTH
1550
N
M

A program message is a sequence of functional elements, the minimum units that can represent functions. In the above figure, functional elements are indicated by capital characters with them enclosed in < >. Functional elements are further classified into coding elements which are indicated by lowercase characters with them enclosed in < >.

The chart indicating the route of selection of functional elements is called a functional syntactical chart. The chart indicating the route of selection of coding elements is called a coding syntactical chart. Refer to the Section 5.1 "Summary of Listener Input Program Message Syntactical Notation" for the program message formats using these functional and coding syntactical charts.

Coding elements indicate coding of the actual bus which is required to send functional element data byte to a device. Upon receipt of a functional element data byte, the listener checks whether individual elements follow the coding syntax rules. If these elements do not follow the rules, the listener causes a command error without regarding the elements as functional elements.

5.1 Summary of Listener Input Program Message Syntactical Notation

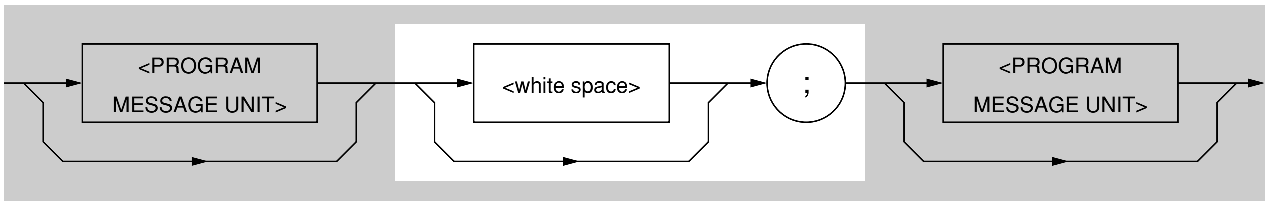
This section gives a general description of program message functional units and program data formats. Refer to the Section 5.2 "Program Message Functional Elements" for program message functional units and the Section 5.3 "Program Data Format" for data formats. (Compound commands and common commands are excluded.)

5.1.1 Separator, Terminator, and Space Before Header

(1) <PROGRAM MESSAGE UNIT SEPARATOR>

Link two or more <PROGRAM MESSAGE UNIT> elements using zero or more spaces and a semicolon.

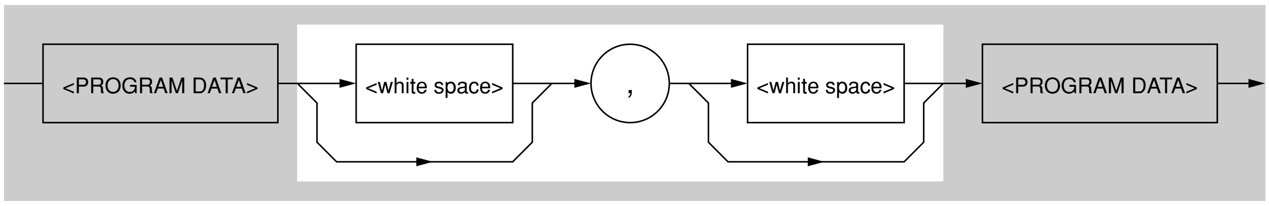
<Example> The general format for linking two <PROGRAM MESSAGE UNIT> elements



(2) <PROGRAM DATA SEPARATOR>

Separate two or more contiguous pieces of <PROGRAM DATA> using a comma in between zero or more spaces.

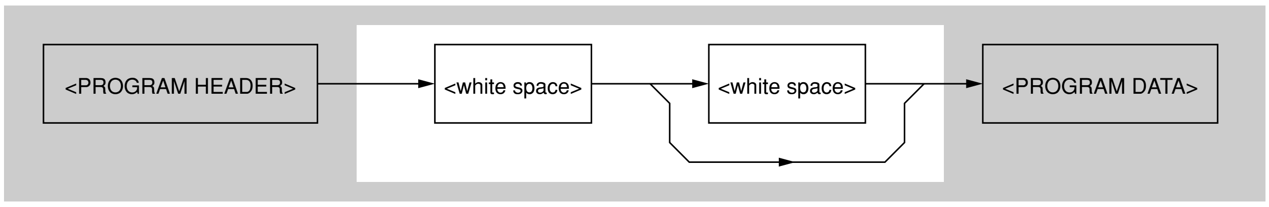
<Example> The general format for separating two pieces of <PROGRAM DATA>



(3) <PROGRAM HEADER SEPARATOR>

Separate <PROGRAM HEADER> and <PROGRAM DATA> using one space and zero or more spaces.

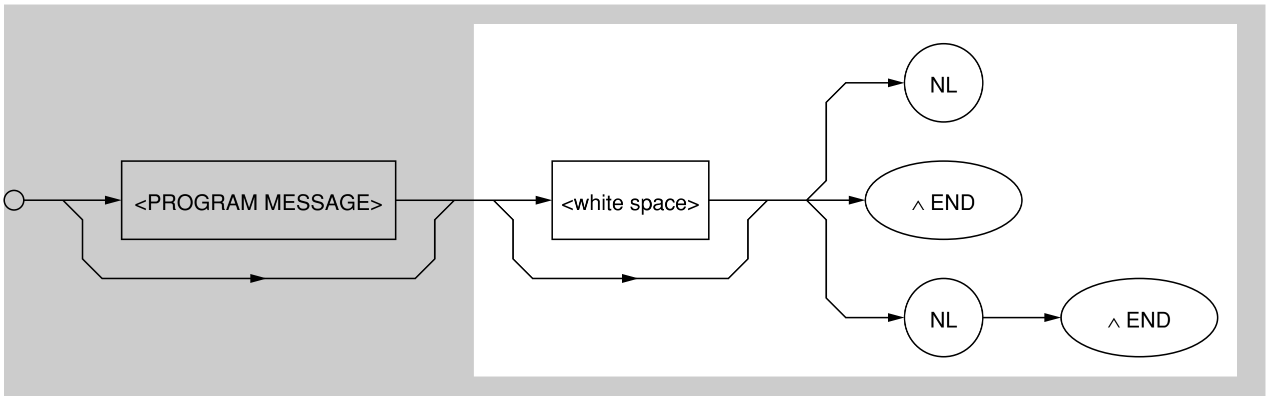
<Example> The general format of single command <PROGRAM HEADER>



(4) <PROGRAM MESSAGE TERMINATOR>

Add zero or more spaces and one of NL, EOI, and a combination of NL and EOI at the end of a <PROGRAM MESSAGE>.

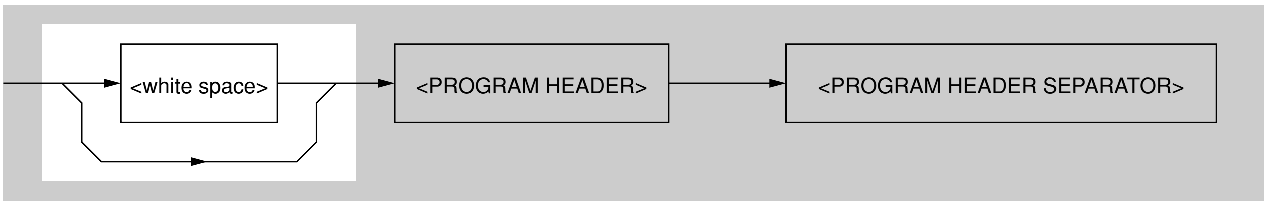
<General format>



(5) Space before header

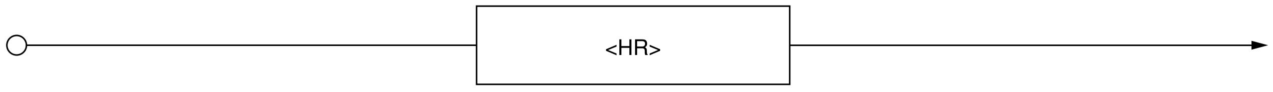
Zero or more spaces can precede a <PROGRAM HEADER>.

<General format>



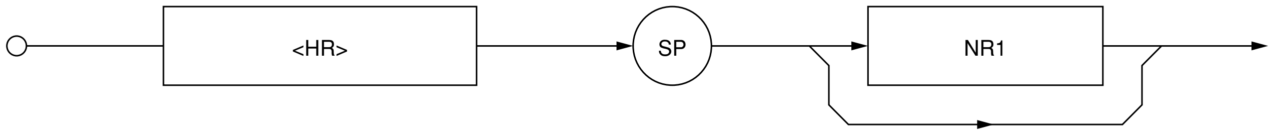
5.1.2 General Format of Program Command Message

(1) Message without data specification



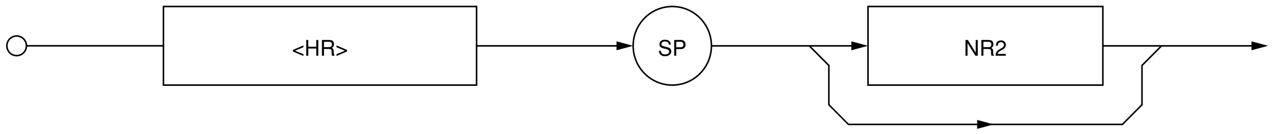
HR: COMMAND PROGRAM HEADER

(2) Message with integer data



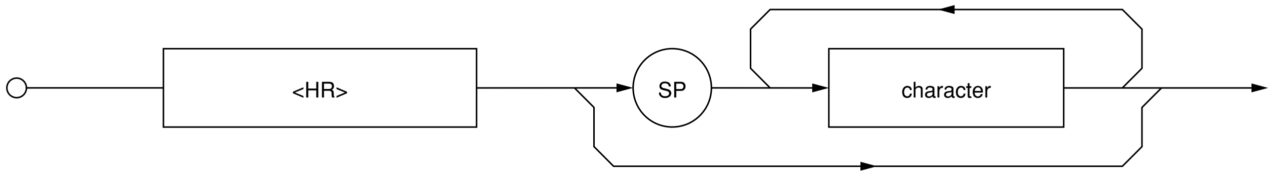
NR1: Integer

(3) Message with real number

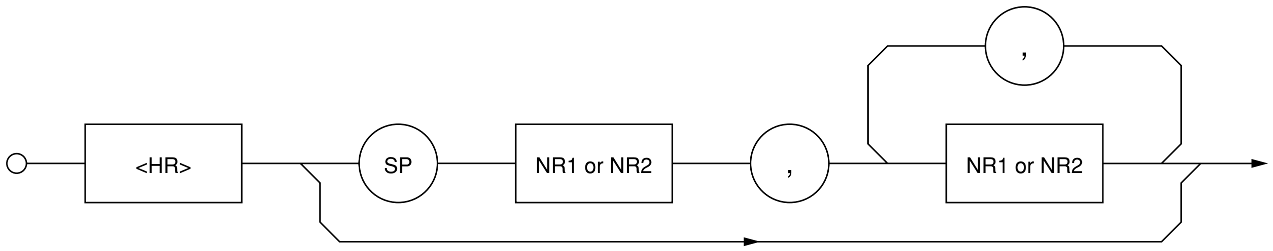


NR2: Real number

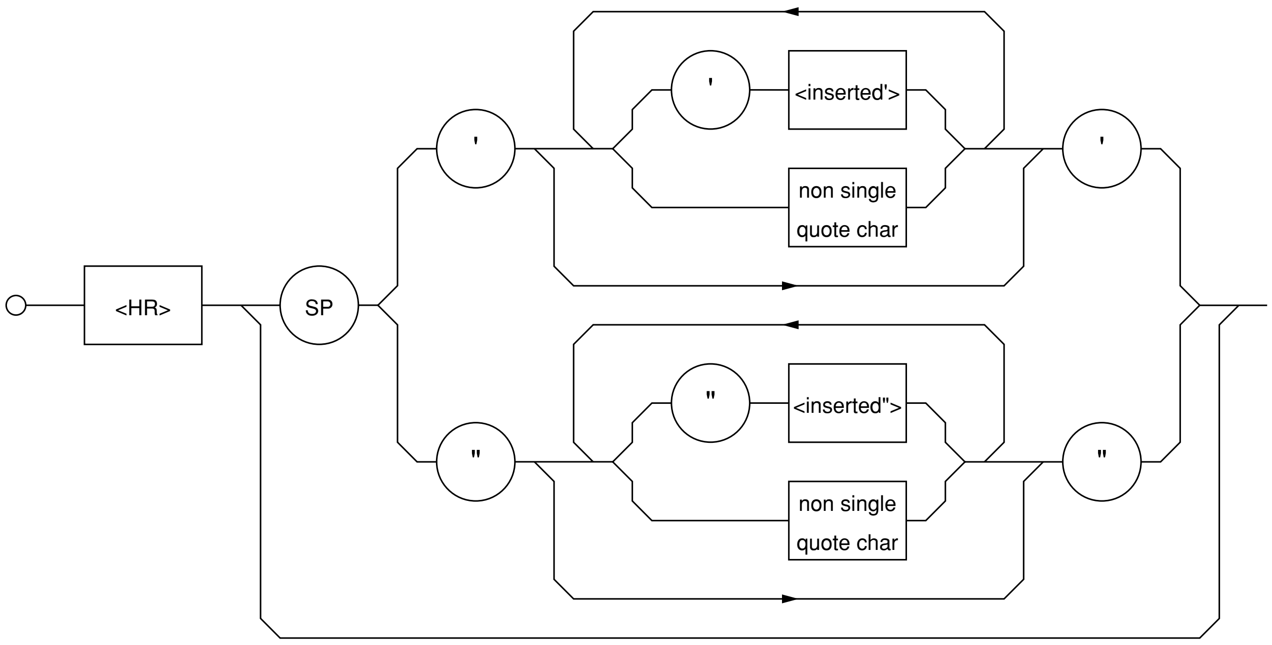
(4) Message with fixed or arbitrary character string data (data length ≤ 12 characters)



(5) Message with multiple pieces of program data (starts with NR1)



(6) Character-only message that can use all ASCII 7-bits

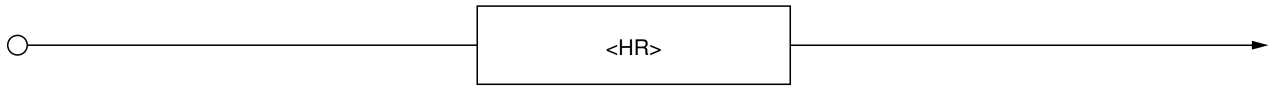


<inserted '>:	Single ASCII code representing a value 27
non-single quote char:	Single ASCII code representing a value other than 27
<inserted ">:	Single ASCII code representing a value 22
non-single quote char:	Single ASCII code representing a value other than 22

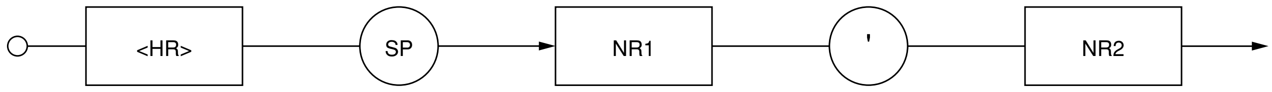
5.1.3 General Format of Query Message

Add a question mark (?) at the end of command <PROGRAM HEADER> for a query <PROGRAM HEADER>.

(1) Message without query data specification



(2) Message with query data specification

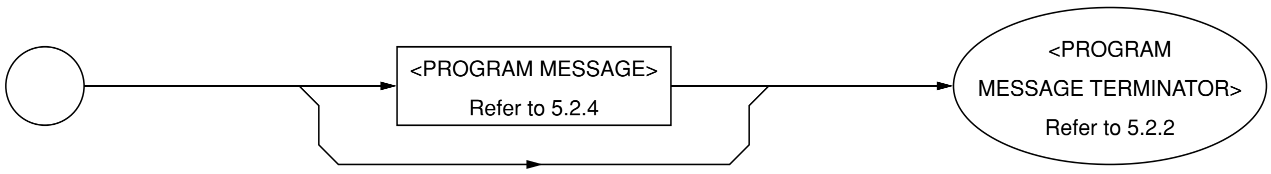


5.2 Program Message Functional Elements

A device accepts a program message by detecting the terminator added at the end of the program message. The following pages describe the functional elements of the program message.

5.2.1 <TERMINATED PROGRAM MESSAGE>

<TERMINATED PROGRAM MESSAGE> is defined as follows:

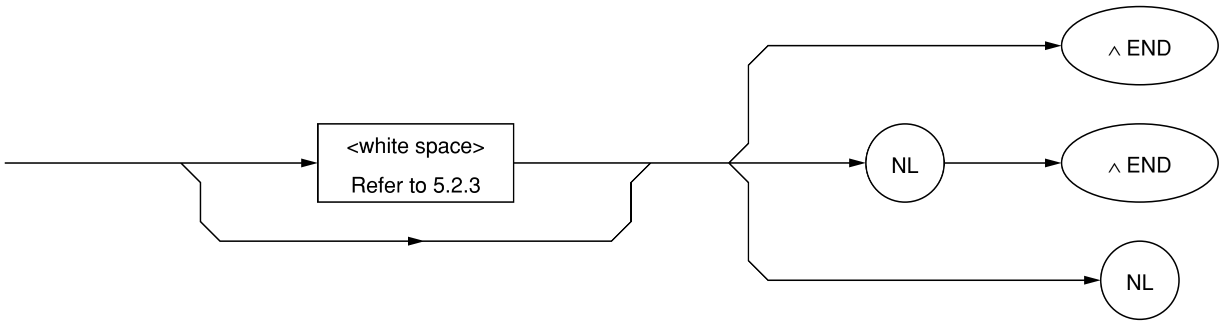


<TERMINATED PROGRAM MESSAGE> is a data message containing all the necessary functional elements to be sent from a controller to a device.

To complete the transfer of <PROGRAM MESSAGE>, <PROGRAM MESSAGE TERMINATOR> is added at the end of <PROGRAM MESSAGE>.

5.2.2 <PROGRAM MESSAGE TERMINATOR>

<PROGRAM MESSAGE TERMINATOR> is defined as follows:

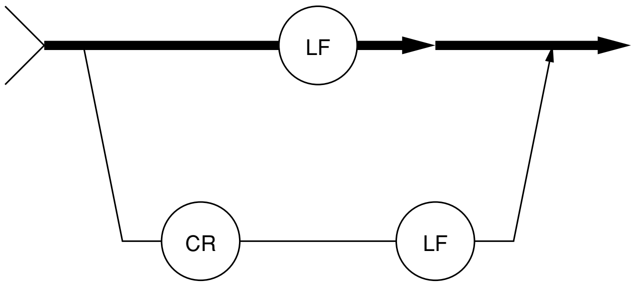


<PROGRAM MESSAGE TERMINATOR> terminates a sequence of one or more fixed-length <PROGRAM MESSAGE UNIT> elements.

NL	Defined as a single ASCII code byte 0A (decimal 10), which is an ASCII control character LF (Line Feed) that moves the printing position down one line. Because the printing starts at a new line, it is also called NL (New Line).
END	Sets the EOI line, one of GPIB control buses, at the LOW level (True), generating an EOI signal.

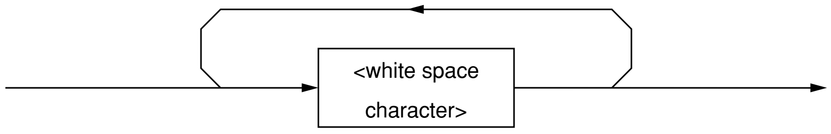
NOTE:

The CR code is used to return the printing position to the first character position on the same line; however, most listeners ignore this cord. Some products available on the market uses CR-LF code, so most controllers are so designed that CR and LF codes are issued in succession.



5.2.3 <white space>

<white space> is defined as follows:

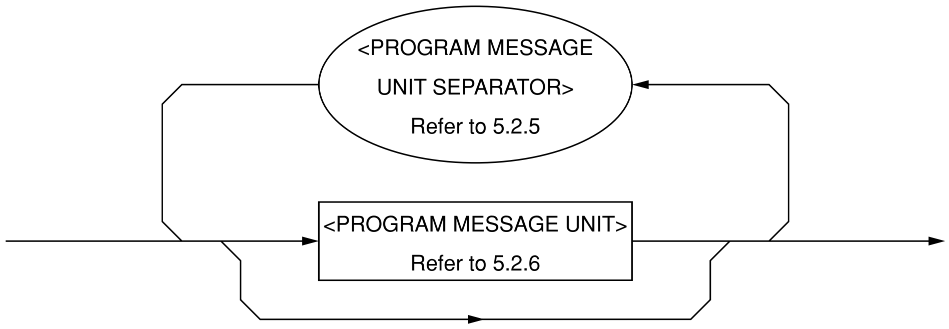


<white space character> is one of single ASCII code bytes 00 to 09 and 0B to 20 (decimal values 0 to 9 and 11 to 32).

This range includes ASCII control codes and space signals and excepts NL. The device does not regard these codes as ASCII control codes but the spaces and it skips those cords.

5.2.4 <PROGRAM MESSAGE>

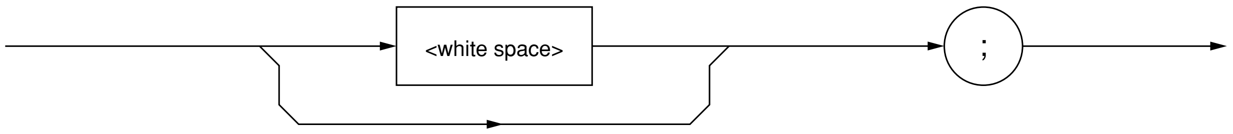
<PROGRAM MESSAGE> is defined as follows:



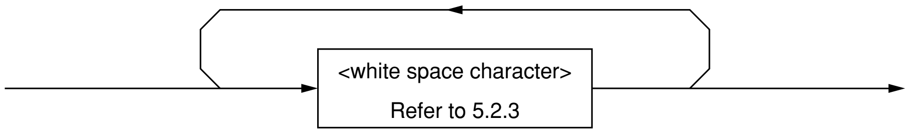
<PROGRAM MESSAGE> is zero, a <PROGRAM MESSAGE UNIT> element, or a sequence of <PROGRAM MESSAGE UNIT> elements. A <PROGRAM MESSAGE UNIT> element is a programming command or data which is sent from a controller to a device.
A <PROGRAM MESSAGE UNIT SEPARATOR> element is used to separate two or more <PROGRAM MESSAGE UNIT> elements.

5.2.5 <PROGRAM MESSAGE UNIT SEPARATOR>

<PROGRAM MESSAGE UNIT SEPARATOR> is defined as follows:



<white space> is defined as follows:

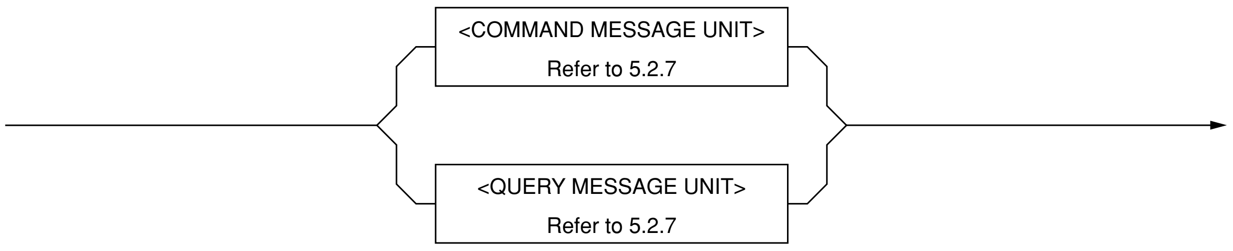


<PROGRAM MESSAGE UNIT SEPARATOR> divides a sequence of <PROGRAM MESSAGE UNIT> elements within the range of <PROGRAM MESSAGE>.
A device interprets a semi-colon (;) as the separator between <PROGRAM MESSAGE UNIT> elements. Accordingly, <white space character> placed before and after the semi-colon (;) is ignored, although <white space character> improves program readability. <white space> following a semi-colon (;) is also used as a <white space> for the next <PROGRAM HEADER>.

Section 5.2.8

5.2.6 <PROGRAM MESSAGE UNIT>

<PROGRAM MESSAGE UNIT> is defined as follows:

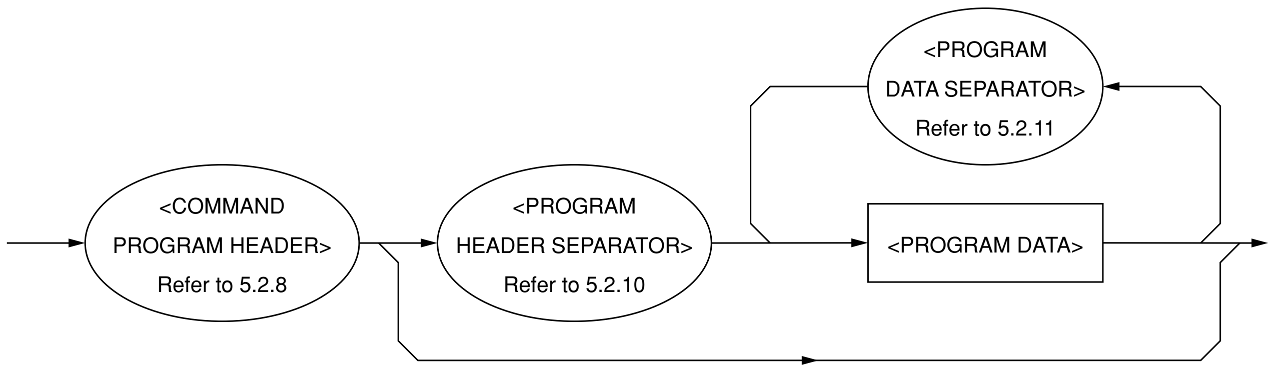


<PROGRAM MESSAGE UNIT> is a single command message received by a device.
It consists of <COMMAND MESSAGE UNIT> or <QUERY MESSAGE UNIT>, which is a single query message.

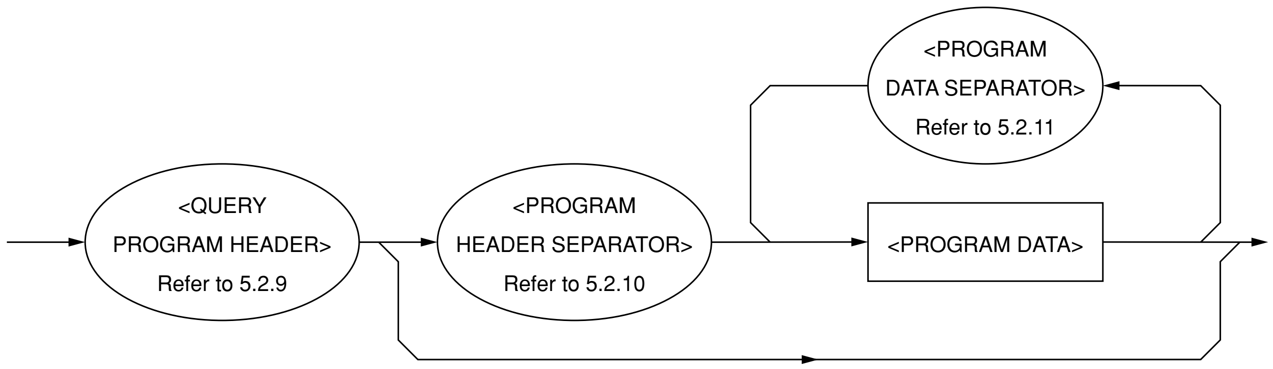
Refer to the Section 5.2.7 "<COMMAND MESSAGE UNIT>/<QUERY MESSAGE UNIT>" for more details.

5.2.7 <COMMAND MESSAGE UNIT>/<QUERY MESSAGE UNIT>

<COMMAND MESSAGE UNIT> is defined as follows:


<COMMAND PROGRAM HEADER> Refer to 5.2.8
<PROGRAM HEADER SEPARATOR> Refer to 5.2.10
<PROGRAM DATA SEPARATOR> Refer to 5.2.11
<PROGRAM DATA>

<QUERY MESSAGE UNIT> is defined as follows:

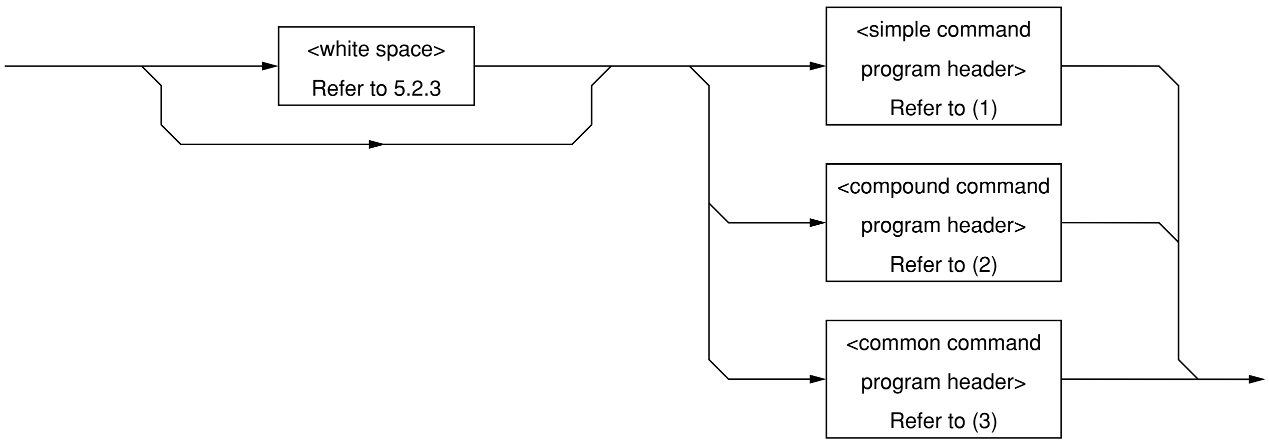

<QUERY PROGRAM HEADER> Refer to 5.2.9
<PROGRAM HEADER SEPARATOR> Refer to 5.2.10
<PROGRAM DATA SEPARATOR> Refer to 5.2.11
<PROGRAM DATA>

When a <PROGRAM HEADER> of <COMMAND MESSAGE UNIT> or <QUERY MESSAGE UNIT> is followed by <PROGRAM DATA>, a space is inserted between these unit. A <PROGRAM HEADER> indicates the application, function, and operation of the program. If a <PROGRAM HEADER> is not followed by <PROGRAM DATA>, the <PROGRAM HEADER> solely indicates the application, function, and operation to be performed in the device.

Among <PROGRAM HEADER> elements, <COMMAND PROGRAM HEADER> is a control command issued from a controller to a device and <QUERY PROGRAM HEADER> is a query command that is issued from a controller to a device in advance so that the controller can receive responses from the device. <QUERY PROGRAM HEADER> always ends with a query indicator, or a question mark (?).

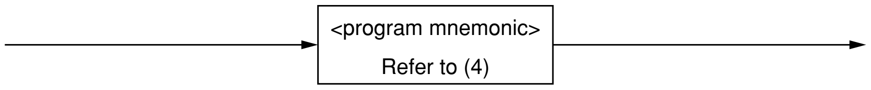
5.2.8 <COMMAND PROGRAM HEADER>

<COMMAND PROGRAM HEADER> is defined below.
Each header can be followed by <white space>.

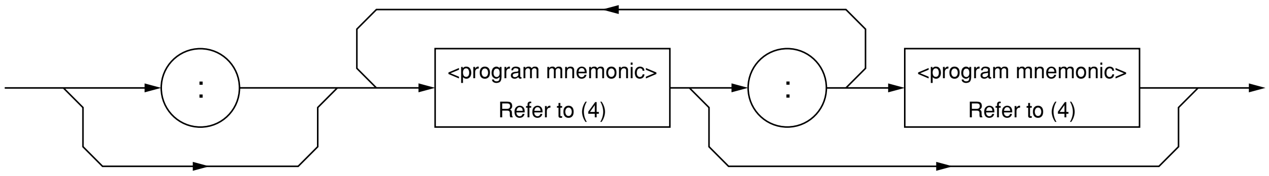


<white space> Refer to 5.2.3
<simple command program header> Refer to (1)
<compound command program header> Refer to (2)
<common command program header> Refer to (3)

(1) <simple command program header> is defined as follows:

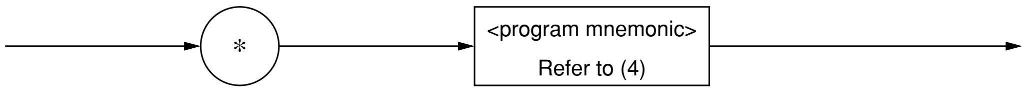


(2) <compound command program header> is defined as follows:

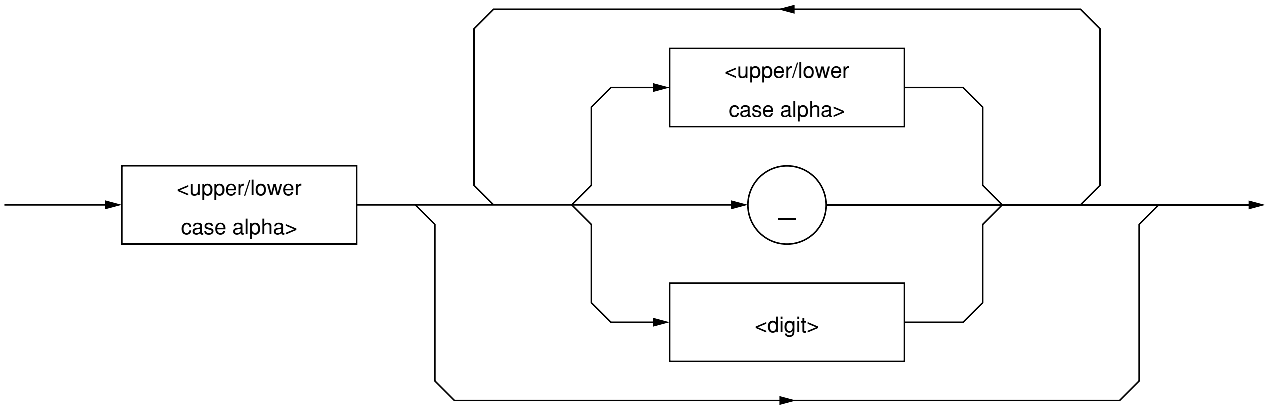


:
<program mnemonic> Refer to (4)
:
<program mnemonic> Refer to (4)

(3) <common command program header> is defined as follows:



(4) <program mnemonic> is defined as follows:



<COMMAND PROGRAM HEADER>

<COMMAND PROGRAM HEADER> indicates the application, function, and operation of the program data to be executed by the device usually followed by <PROGRAM DATA>. When it is not followed by <PROGRAM DATA>, the header solely indicates the application, function, and operation to be performed in the device.
The meanings of an application, function, or operation is represented by <program mnemonic> in ASCII cord, which is widely called a mnemonic. Mnemonics and the <COMMAND PROGRAM HEADER> defined in (1) to (3) above are explained below.

<program mnemonic>

A mnemonic begins with an uppercase or lowercase character, which is followed by an arbitrary combination of characters such as uppercase characters (A to Z) or lowercase characters (a to z), underline (_), and numeric characters (0 to 9). A mnemonic can contain a maximum of 12 characters; however, most mnemonics contain 3 to 4 characters. (No space is inserted between characters.)

<upper/lower case alpha>	One of ASCII code bytes 41 to 5A and 61 to 7A (decimal values 65 to 90 and 97 to 122 = uppercase characters A to Z and lowercase characters a to z). The device can accept a header irrespective of whether it is represented by uppercase or lowercase characters.
<digit>	One of ASCII code bytes 30 to 39 (decimal values 48 to 57 = characters 0 to 9).
(_)	An ASCII code byte, i.e., ASCII code byte 5F (decimal value 95 = underline).

<simple command program header>

The above rules for <program mnemonic> applies.

<compound command program header>

<compound command program header> is a <COMMAND PROGRAM HEADER> that executes a compound function. <program mnemonic> is always preceded by a colon (:) to separate it from <compound command program header>. When only one <compound command program header> is used, the succeeding colon (:) may be omitted.

Function:

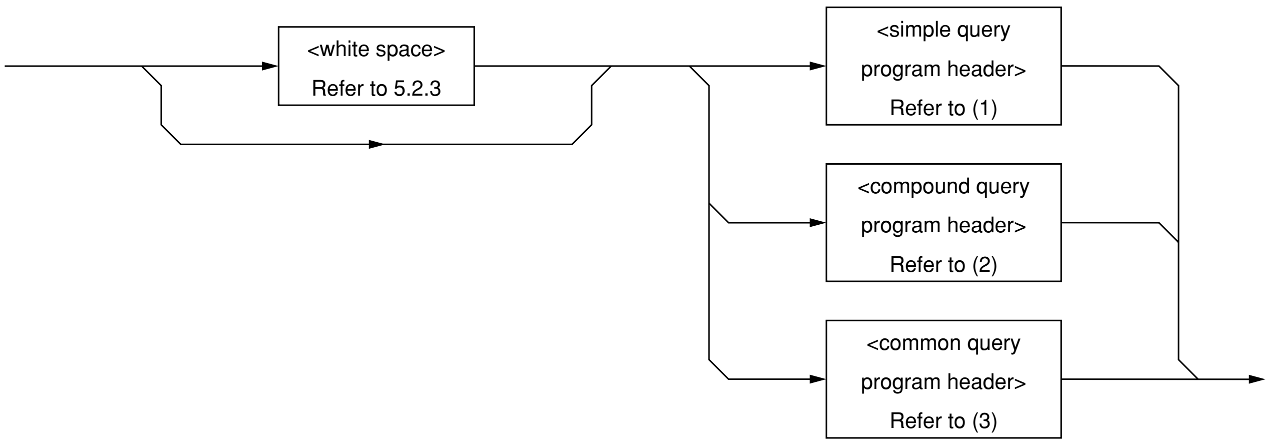
On a complex device, a device command set is organized logically by providing a compound function instead of limiting the number of unique headers. A hierarchical command structure can be handled effectively.

<common command program header>

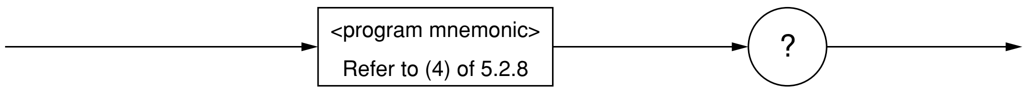
An asterisk (*) is always added before <program mnemonic> of <common command program header>. "Common" means that this command is a program command which commonly used for other IEEE 488.2-ready measuring instruments connected to the bus.

5.2.9 <QUERY PROGRAM HEADER>

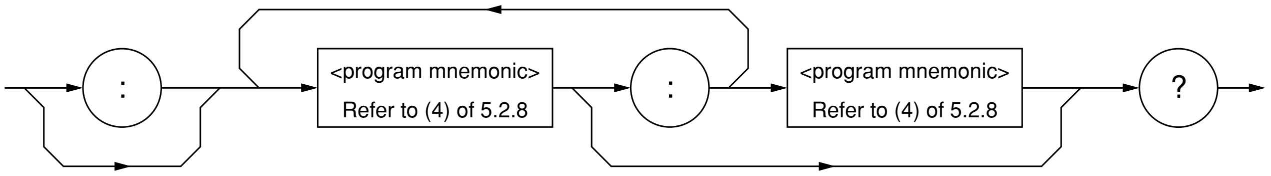
<QUERY PROGRAM HEADER> is defined as follows:
<white space> may be written before each header.


<white space>
Refer to 5.2.3
<simple query program header>
Refer to (1)
<compound query program header>
Refer to (2)
<common query program header>
Refer to (3)

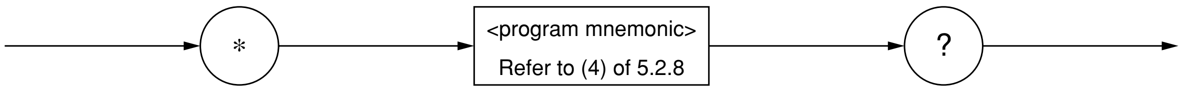
(1) <simple query program header> is defined as follows:



(2) <compound query program header> is defined as follows:



(3) <common query program header> is defined as follows:



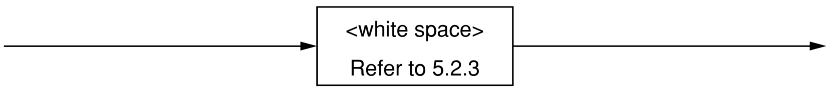
<QUERY PROGRAM HEADER>

<QUERY PROGRAM HEADER> is a query command which is sent from a controller to a device in advance so that the controller can receive response messages from the device. This header always ends with a query indicator, or a question mark (?). It is explained below using examples of programs.

The format of <QUERY PROGRAM HEADER> is the same as that of <COMMAND PROGRAM HEADER> with the exception that a query indicator, or a question mark (?), is added at the end. Refer to the Section 5.2.8 "<COMMAND PROGRAM HEADER>."

5.2.10 <PROGRAM HEADER SEPARATOR>

<PROGRAM HEADER SEPARATOR> is defined as follows:



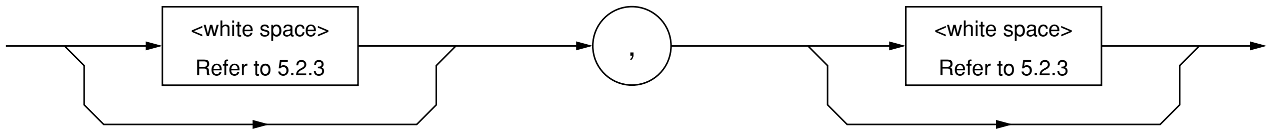
<PROGRAM HEADER SEPARATOR> is used as the separator between <COMMAND PROGRAM HEADER> or <QUERY PROGRAM HEADER> and <PROGRAM DATA>.

When there are two or more <white space character> elements between the <PROGRAM HEADER> and the <PROGRAM DATA>, the first <white space character> is interpreted as a separator and the remaining <white space character> is ignored, although <white space character> improves program readability.

At least one header separator must exist between the header and the data. One separator indicates the end of the <PROGRAM HEADER> as well as the beginning of the <PROGRAM DATA>.

5.2.11 <PROGRAM DATA SEPARATOR>

<PROGRAM DATA SEPARATOR> is defined as follows:



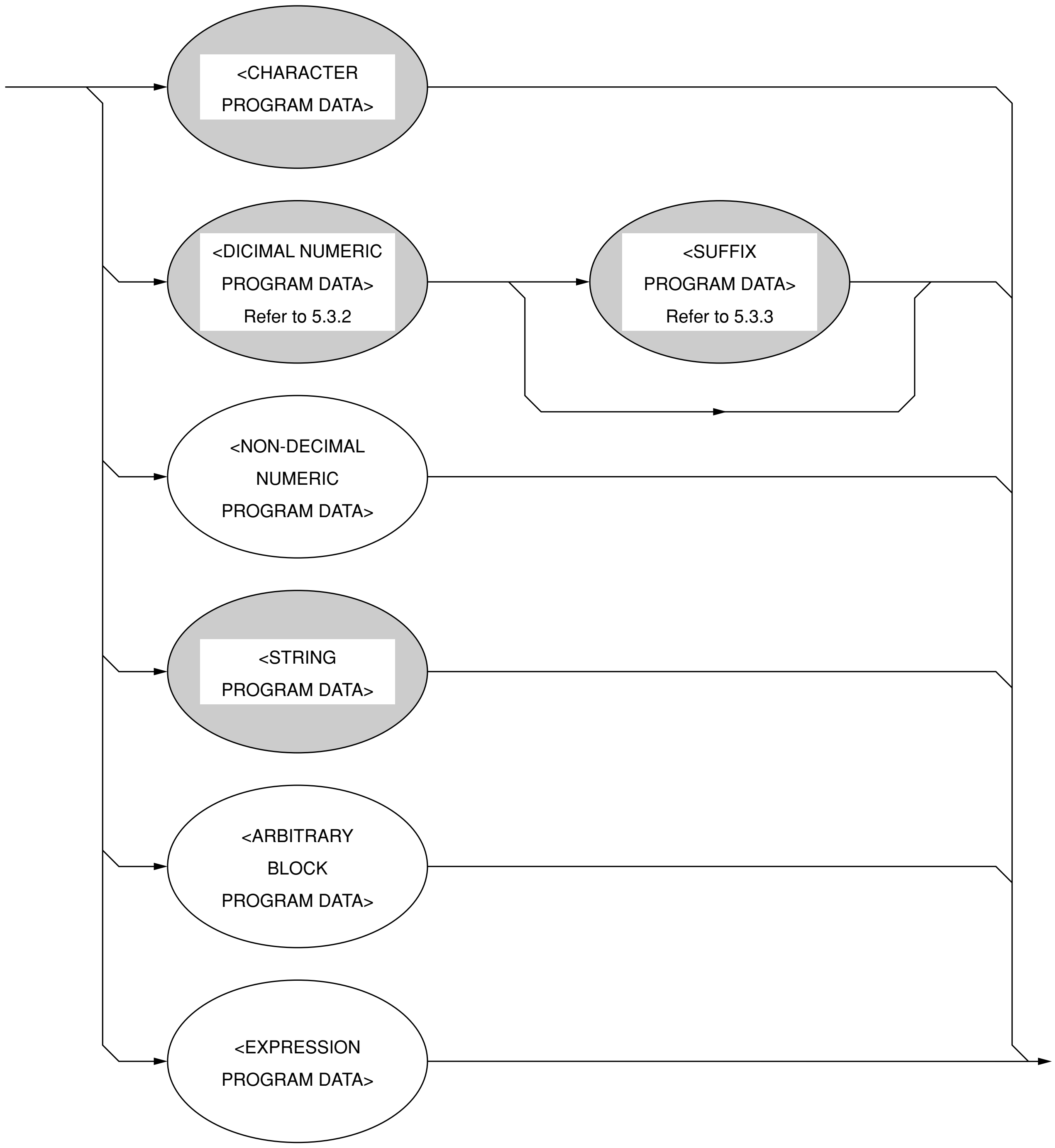
<PROGRAM DATA SEPARATOR> is used to separate the parameters, when <COMMAND PROGRAM HEADER> or <QUERY PROGRAM HEADER> has many parameters.

When this data separator is used, a comma is mandatory but <white space character> can be omissible. The <white space character> before a comma and the <white space character> after a comma are ignored, although <white space character> improves program readability.

5.3 Program Data Format

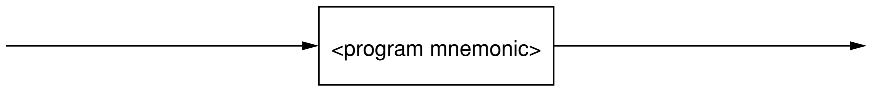
This section explains the format of the <PROGRAM DATA> shown in the functional syntactical charts in the Section 5.2.7 "<COMMAND MESSAGE UNIT>/<QUERY MESSAGE UNIT>", which is one of terminated program message formats.

The functional element of the <PROGRAM DATA> is used to transfer various types of parameters related to the <PROGRAM HEADER>. <PROGRAM DATA> types are shown below. The MT9810A accepts the program data shown in the hollow squares surrounded by a shade. For the program data not supported by the MT9810A, read this section just for reference.

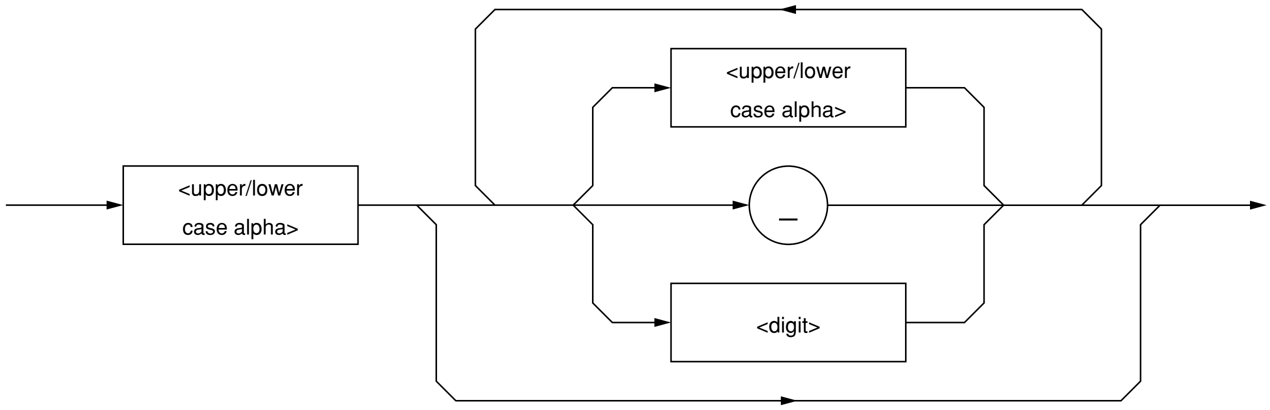

<CHARACTER PROGRAM DATA>
<DICIMAL NUMERIC PROGRAM DATA> Refer to 5.3.2
<SUFFIX PROGRAM DATA> Refer to 5.3.3
<NON-DECIMAL NUMERIC PROGRAM DATA>
<STRING PROGRAM DATA>
<ARBITRARY BLOCK PROGRAM DATA>
<EXPRESSION PROGRAM DATA>

5.3.1 <CHARACTER PROGRAM DATA>

The functional element of the <CHARACTER PROGRAM DATA> is used to perform remote control by transferring short alphabetic or alphanumeric data. It is defined as follows:



Details on character data are the same as those on <program mnemonics>. The numeric data has been focused as control data, however, the program data can also be used to perform control. A coding syntactical chart is as follows:



The data always begins with an uppercase or lowercase character, which is followed by an arbitrary combination of characters such as uppercase characters (A to Z) or lowercase characters (a to z), underline (_), and numeric characters (0 to 9). Since combinations of alphanumeric characters are used as mnemonic-like symbols, the maximum data length is 12 characters.

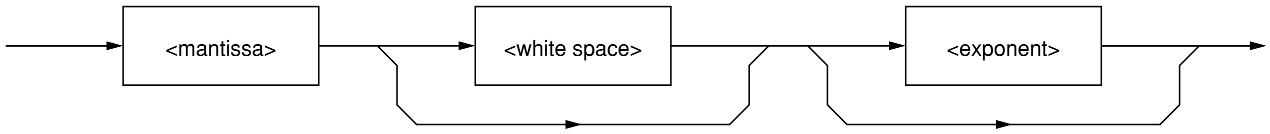
<upper/lower case alpha>	One of ASCII code bytes 41 to 5A and 61 to 7A (decimal values 65 to 90 and 97 to 122 = uppercase characters A to Z and lowercase characters a to z). The device can accept a header irrespective of whether it is represented by uppercase or lowercase characters.
<digit>	One of ASCII code bytes 30 to 39 (decimal values 48 to 57 = characters 0 to 9).
(_)	A single ASCII code byte, i.e., ASCII code byte 5F (decimal value 95 = underline).

Therefore, <CHARACTER PROGRAM DATA> is <PROGRAM DATA> used to transfer relatively short mnemonic-type alphanumeric codes.

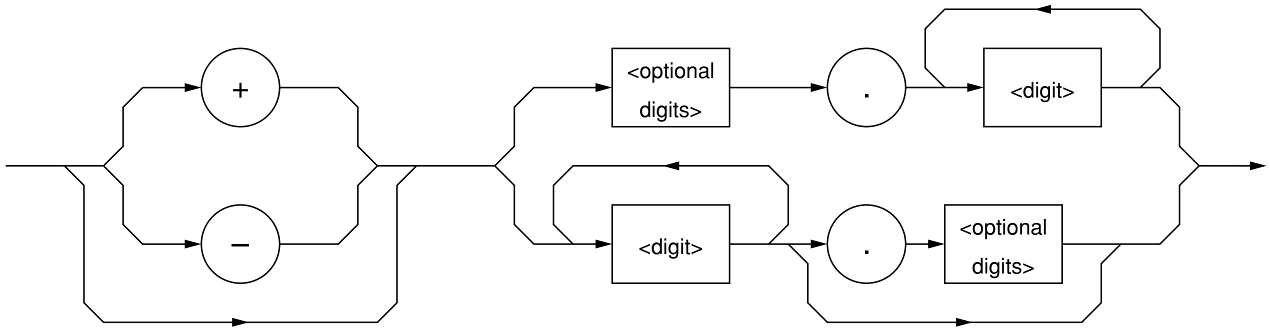
5.3.2 <DECIMAL NUMERIC PROGRAM DATA>

<DECIMAL NUMERIC PROGRAM DATA> is <PROGRAM DATA> used to transfer numeric constants represented in decimal notation. There are three types of decimal numeric representation: integer, fixed-point, and floating-point.

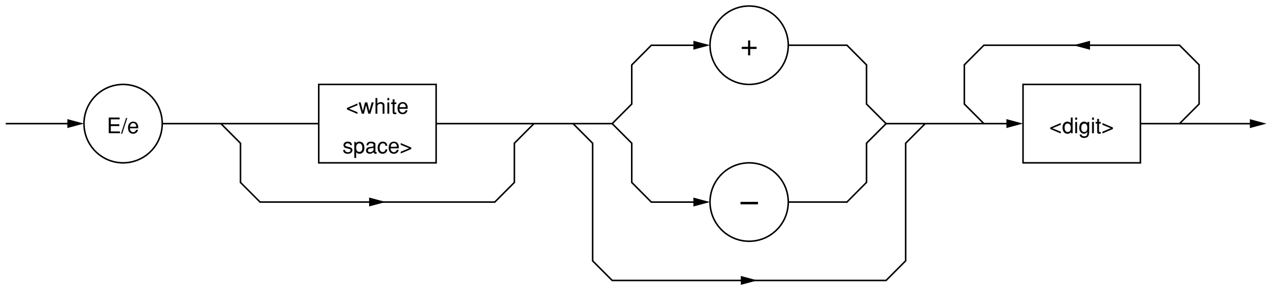
These three types of numerics represent decimal numeric program data, which can contain spaces, flexibly (NRf: flexible numeric representation). These numerics are defined as follows:



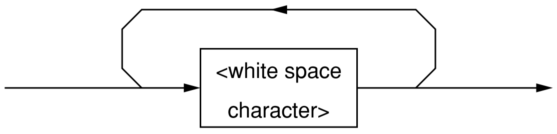
<mantissa> is defined as follows:

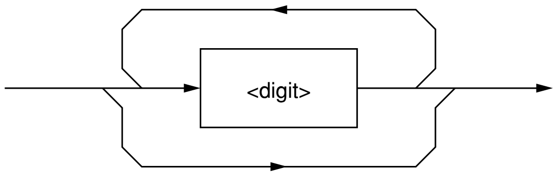


<exponent> is defined as follows:



<white space> and <optional digits> are defined as follows:





refer to the Section 5.2.3 "<white space>" for <white space>, and refer to the Section 5.3.1 "<CHARACTER PROGRAM DATA>" for <digit>.

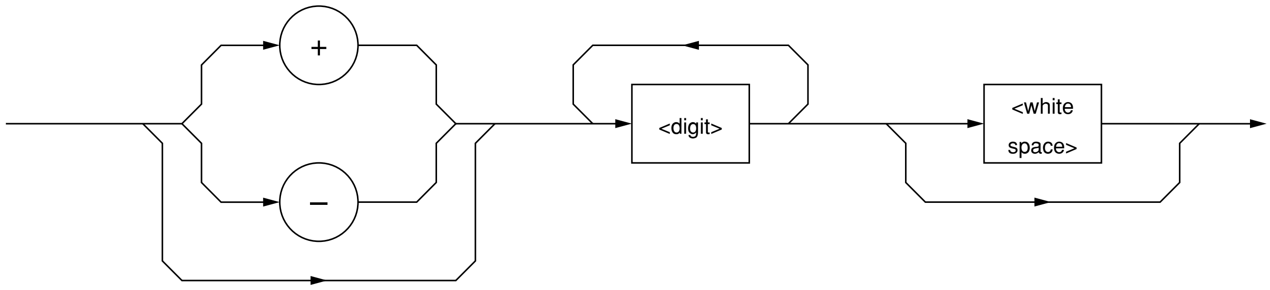
The following pages describe coding syntactical charts of decimal numeric program data with respect to integer, fixed-point, and floating-point notations respectively.

Note that the following processing is performed during transfer of any type of numeric representation:

Rounding of numeric element	When a device receives a <DECIMAL NUMERIC PROGRAM DATA> element having too many digits to handle, it ignores the sign of the element value and rounds it off.
Data outside the range	If the <DECIMAL NUMERIC PROGRAM DATA> element value is outside the range permitted in relation to the program header, an execution error is reported.

(1) Integer NR1 transfer

A decimal value not including a decimal point and exponent, i.e., an integer (NR1) in a real number, is transferred.

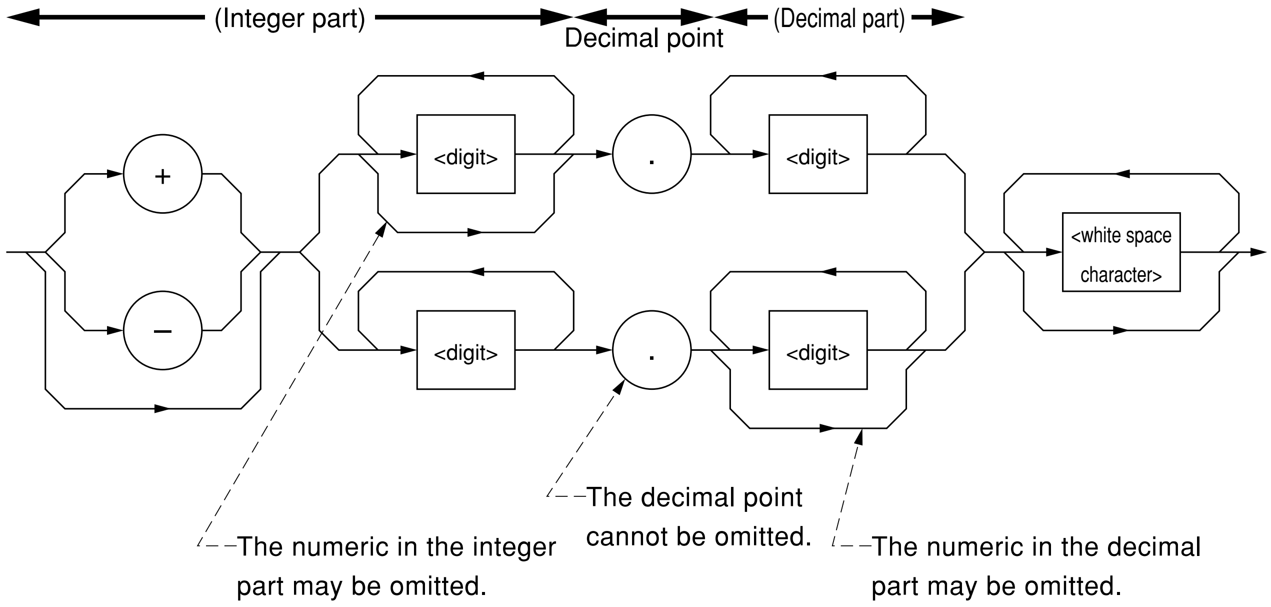


• 0 (s) may be added at the beginning	→	005, +000045
• A space (+ or –) must not be inserted between a sign and a numeric.	→	+5, +Δ5 (×)
• Spaces may be added after a numeric.	→	+5ΔΔΔ
• The + sign may be omitted.	→	+5, 5
• Commas must not be used to indicate decimal places.	→	1,234,567 (×)

(2) Fixed-point NR2 transfer

A decimal number having digits below the decimal point, i.e., an integer and a real number (NR2) except an exponent, is transferred.

The syntactical chart shows an integer part and a decimal point and a decimal part.

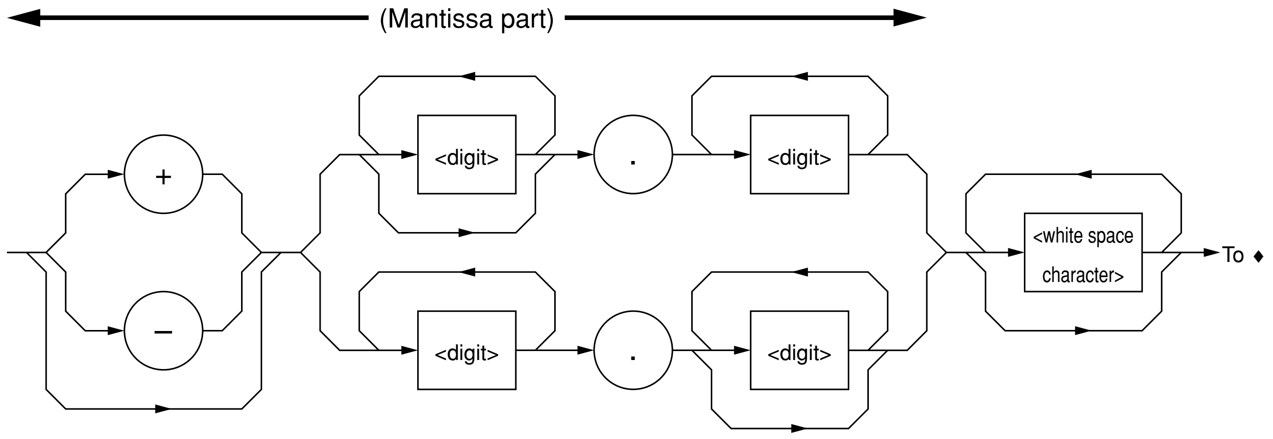


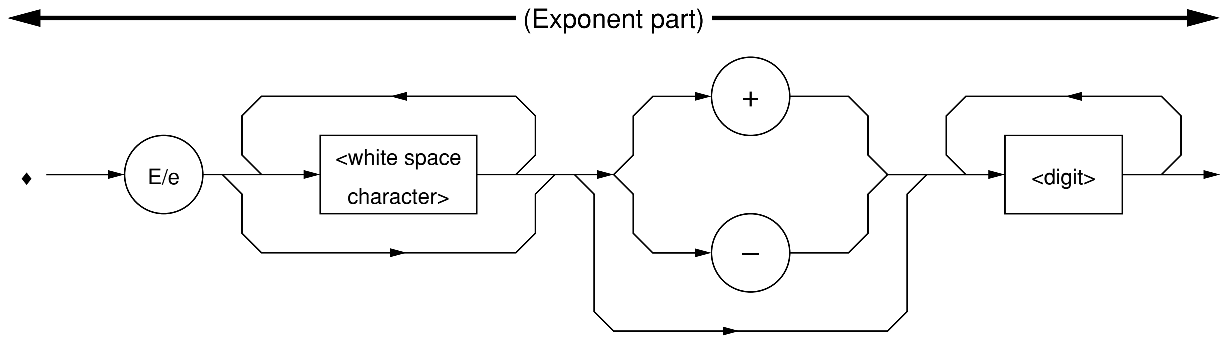
(Integer part)
Decimal point
(Decimal part)
+
−
<digit>
.
<white space character>
The numeric in the integer part may be omitted.
The decimal point cannot be omitted.
The numeric in the decimal part may be omitted.

• An integer representation is applied to the integer part.		
• A space must not be inserted between a numeric and a decimal point.	→	+753 Δ.123 (×)
• Spaces may be added after the numeric in the decimal part.	→	+753.123ΔΔΔΔ
• The decimal point need not follow a numeric.	→	.05
• A sign may be written before a decimal point.	→	+.05, –.05
• A numeric may end with a decimal point.	→	12.

(3) Floating-point NR3 transfer

A decimal numeric valve having an exponent, i.e., a real number (NR3) represented in floating-point notation, is transferred. The syntactical chart consists of a mantissa part and an exponent part. The exponent part is represented in integer and floating-point notation to indicate precision of the numeric. The exponent part begins with E. On the right of E is a number to the power of 10.



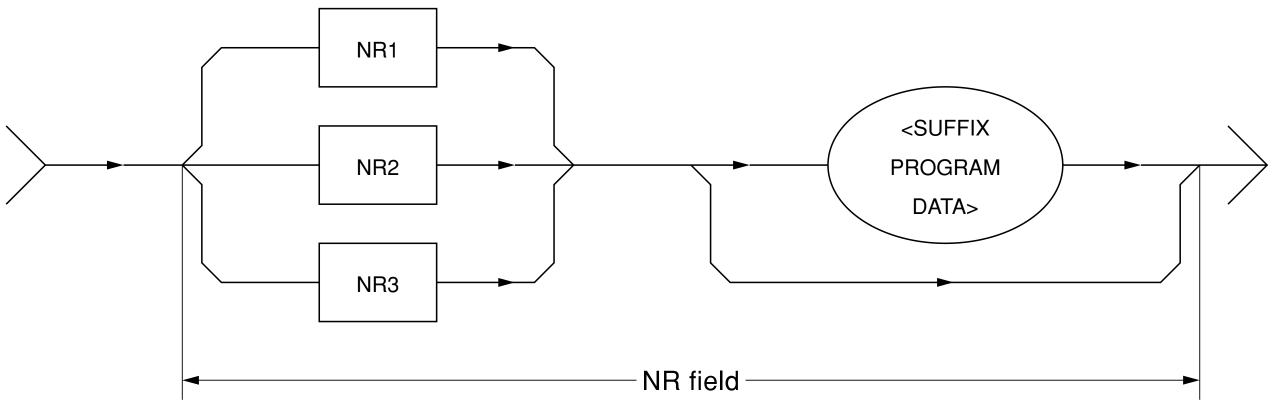


- E indicates power of 10. It indicates the beginning of the exponent part.

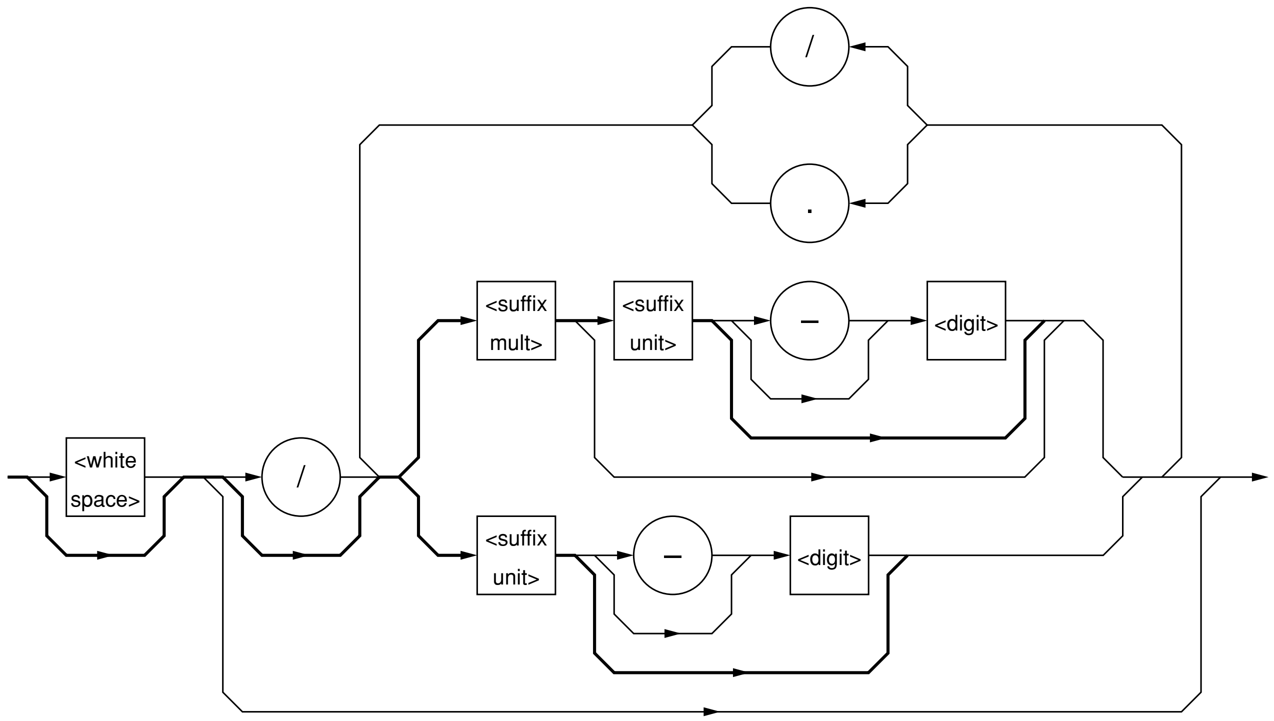
• E may be either an uppercase or lowercase character.	→	1.234E+12, 1.234e+12
• A space may be written before or after E/e.	→	1.234 Δ E Δ +12
• If the sign is +, it may be omitted in mantissa and exponent parts.	→	+1.234E+4, 1.234E4
• The numeric in the exponent part cannot be omitted.	→	–1E2, –E2 (×), –.E2 (×)

5.3.3 <SUFFIX PROGRAM DATA>

<SUFFIX PROGRAM DATA> follows <DECIMAL NUMERIC PROGRAM DATA> (integer NR1, fixed-point NR2, or floating-point NR3) described in the Section 5.3.2 "<DECIMAL NUMERIC PROGRAM DATA>." The NR1, NR2, and NR3 may be followed by a suffix.



A suffix is added at the end of decimal numeric program data only when the data requires a unit of measure. It is a combination of a suffix unit and a suffix multiplier. The syntactical chart is shown below. Bold-line routes are used frequently.



- A suffix multiplier is represented by an uppercase or lowercase character.
  For example, 1E3 Hz is represented by 1 kHz assuming 1E3 = k.

- A suffix unit is represented by an uppercase or lowercase character.

- Placing E at the beginning of <SUFFIX PROGRAM DATA> is prohibited because it may be confused with the E used for floating-point decimal numerics.

Suffix multipliers and units are listed in the Table 5-1.

(1) Suffix multipliers

Table 5-1 Suffix multipliers

Multiplier	Mnemonic	Name
1E18	EX	EXA
1E15	PE	PETA
1E12	T	TERA
1E9	G	GIGA
1E6	MA (NOTE)	MEGA
1E3	K	KILO
1E-3	M (NOTE)	MILLI
1E-6	U	MICRO
1E-9	N	NANO
1E-12	P	PICO
1E-15	F	FEMTO
1E-18	A	ATTO

NOTE:

According to convention, Hz to the sixth power of 10^6 is MHz (megahertz) and OHM to the six power of 10^6 is MOHM (megaohm). These are not listed in the above table, but listed in the Table 5-2 "Suffix units."

(2) Relative units (dB)

Decibel relative to 1 μV .... DBUV
Decibel relative to 1 μW .... DBUW
Decibel relative to 1 mW .... DBMW

For historical reasons, DBM is allowed as an alias for DBMW.

(3) Suffix units

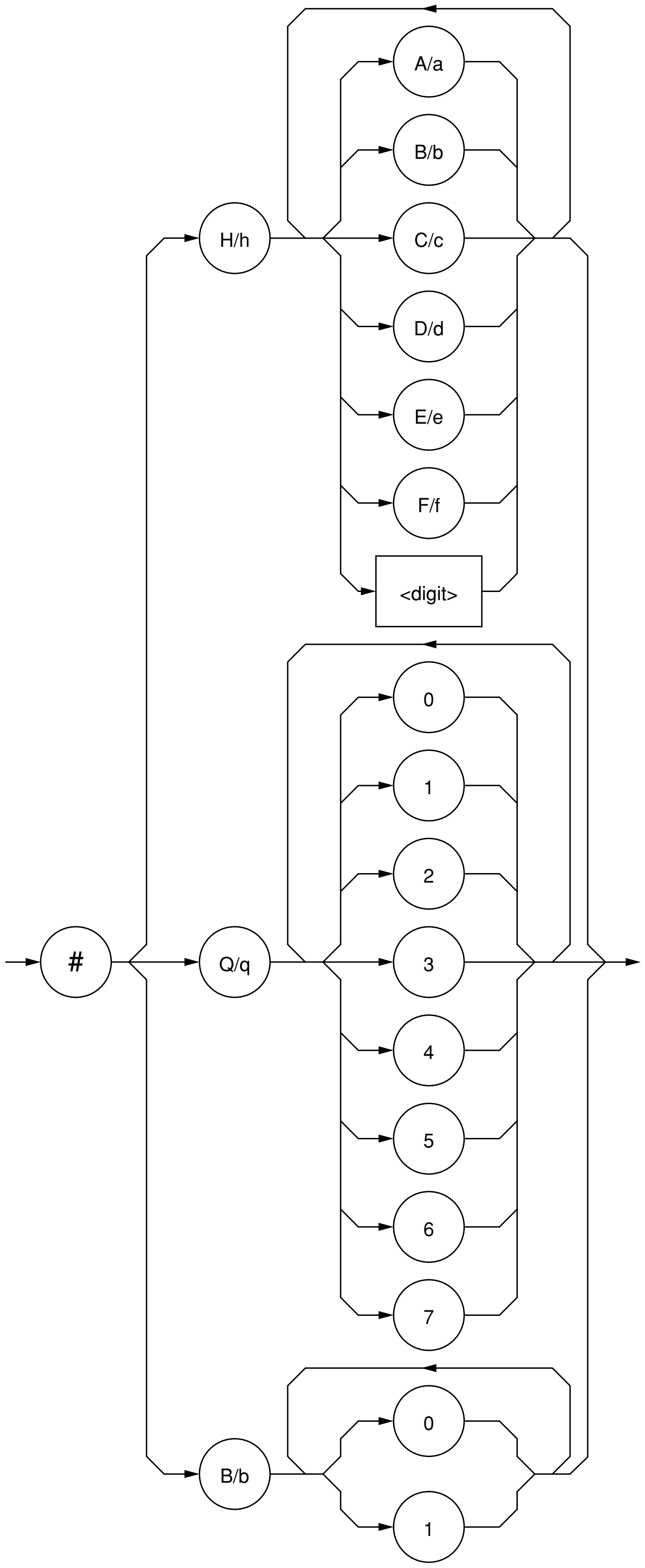
Table 5-2 Suffix units

Item	Recommended mnemonic of unit	Quasi recommended mnemonic of unit	Name
Current	A		Ampere
Atmospheric pressure	ATM		Atmosphere
Charge	C		Coulomb
Luminance	CD		Candela
Decibel	DB		Decibel
Power	DBM		Decibel milliwatt
Capacitance	F		Farad
Mass		G	Gram
Inductance	H		Henry
Frequency (hertz)	HZ		Hertz
Mercury column	INHG		Inches of mercury
Joule	J		Joule
Temperature	K		Degree Kelvin
		CEL	Degree Celsius
		FAR	Degree Fahrenheit
Volume	L		Liter
Luminance	LM		Lumen
Luminance	LX		Lux
Length (meter)	M		Meter
		FT	Feet
		IN	Inch
Frequency (1E3 Hz)		MHZ	Megahertz
Resistance		MOHM	Megaohm
Force	N		Newton
Resistance	OHM		Ohm
Pressure	PAL		Pascal
Ratio (percent)	PCT		Percent
Angle (radian)	RAD		Radian
Angle (degree)		DEG	Degree
		MNT	Minute (of arc)
Time (second)	S	SEC	Second
Conductance	SIE		Siemens
Automatic speed	T		Tesla
Pressure	TORR		Torr
Voltage	V		Volt
Power (watt)	W		Watt
Speed/hour	WB		Weber
Luminance	LM		Lumen

5.3.4 <NON-DECIMAL NUMERIC PROGRAM DATA>

<NON-DECIMAL NUMERIC PROGRAM DATA> is <PROGRAM DATA> used to transfer decimal, octal, and binary numeric data as non-decimal numeric values. Non-decimal data always begins with a number code, or a sharp (#). It is defined as shown in the coding syntactical chart below.

When an unspecified character string is sent, a command error occurs.



The character string following #H or #h is accepted by the device as a hexadecimal number.
The character strings in parentheses are decimal numbers.

#Habc1230	(11,256,099D)
#hAbC123	
#H2DC3	(11,715D)
#h2dc3	
#H8301	(33,537D)
#h8301	

The character string following #Q or #q is accepted by the device as an octal number.

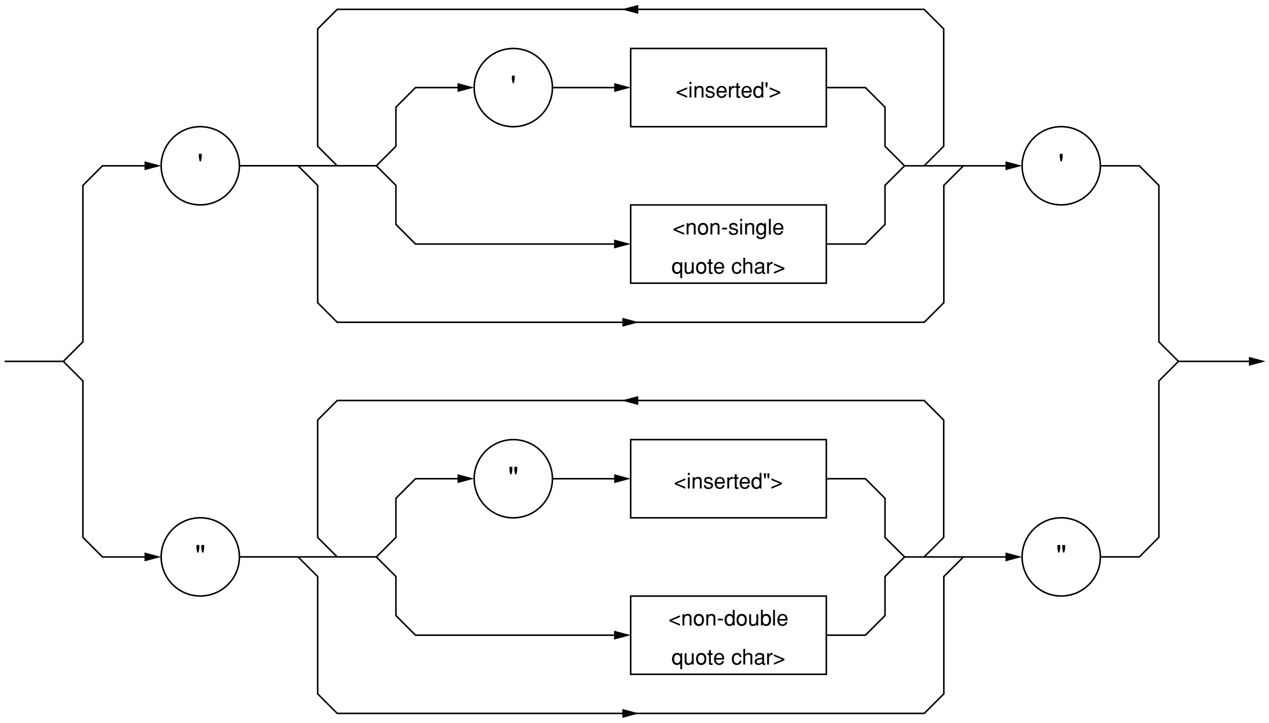
#Q37	(31D)
#q37	
#Q26703	(11,715D)
#q26703	

The character string following #B or #b is accepted by the device as a binary number.

#B101010111100000100100011	(11,256,099D)
#b0010110111000011	(11,715D)

5.3.5 <STRING PROGRAM DATA>

<STRING PROGRAM DATA> is <PROGRAM DATA> consisting of only character strings. All ASCII 7 bit codes can be used. When a character string includes single quotation mark (') or a double quotation mark ("), two identical quotation marks must be written in succession per quotation mark.



(1) A character string must be enclosed with single quotation (') or double quotation (") marks irrespective of whether the character string contains any quotation mark. For example:

It's a nice day.	→	"It's a nice day."
	→	'It' 's a nice day.'

(2) When a character string is enclosed with single quotation marks ('), each single quotation mark contained in the character string must be doubled. Other characters, including double quotation marks ("), must be written as thses are. For example:

"I shouted, 'Shame'."	→	' "I shouted,' 'Shame' '." '

(3) When a character string is enclosed with double quotation marks ("), these double quotation marks must be doubled. Other characters, including single quotation marks ('), must be written as thses are. For example:

"I shouted, 'Shame'."	→	" " "I shouted, 'Shame'." " "

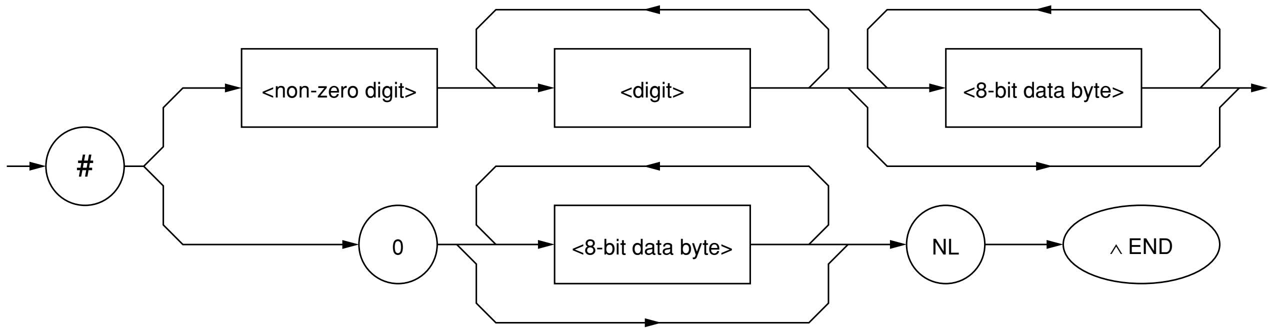
(4) <inserted '> is an single ASCII code set in ASCII code byte 27 (decimal 39 = symbol '). <inserted "> is a single ASCII code set in ASCII code byte 22 (decimal 34 = symbol "). <non-single quote char> and <non-double quote char> are single ASCII codes other than single and double quotation marks (").

5.3.6 <ARBITRARY BLOCK PROGRAM DATA>

<ARBITRARY BLOCK PROGRAM DATA> is non-decimal program data starting with a number code, or a sharp, (#). Binary data is transferred directly in 1 byte (8 bit) blocks. Differences from the non-decimal numeric program data (<NON-DECIMAL NUMERIC PROGRAM DATA>) additionally described in the Section 5.3.4 "<NON-DECIMAL NUMERIC PROGRAM DATA>" are as follows:

- Data is not limited to numeric data, but character string data and numeric data can be handled.
- The number of data bytes to be transferred can be written between a number code, or a sharp, (#) , and the first data.

The non-decimal data is program data that can specify the data bytes to be transferred.



<digit>	One of ASCII code bytes 30 to 39 (decimal values 48 to 57 = characters 0 to 9).
<non-zero digit>	One of ASCII code bytes 31 to 39 (decimal values 49 to 57 = characters 1 to 9).
<8-bit data byte>	An 8 bit byte within the range from 00 to FF (decimal values 0 to 255).

(1) When the number of data bytes to be transferred is known

The upper-right route in the above syntactical chart is applied.
Specify the number of <8-bit data byte> bytes to be transferred at the <digit> position, i.e., just before writing data. Write the number of digits of the specified number of bytes between a number cord, or sharp, (#) and <non-zero digit>. For example, to send 4 data bytes (DABs), write <ARBITRARY BLOCK PROGRAM DATA> as follows:

To send 4 bytes, specify 4 at the <digit> position.
↓
#14<DAB><DAB><DAB><DAB>
↑
The number of digits of the value 4 at the <digit> position is 4. So specify 1 at the <non-zero digit> position.

To send 4 bytes, specify 4 at the <digit> position. Leading 0s may be specified.
↓
#3004<DAB><DAB><DAB><DAB>
↑
The number of digits of the value 4 at the <digit> position is 3. Specify 3 at the <non-zero digit> position.

(2) When the number of data bytes to be transferred is unknown

The lower-right route in the above syntactical chart is applied. Write #0 before the first data and write NL^END after the last data, causing exitless termination.

#0<DAB><DAB><DAB><DAB><DAB>NL^END

(3) Handling integer-precision binary data

Integer-precision binary data is used as <ARBITRARY BLOCK>-type transfer data, whether it is program data or response data, and has the specifications summarized in the Table 5-3. Negative values are processed as two's complements.

Table 5-3

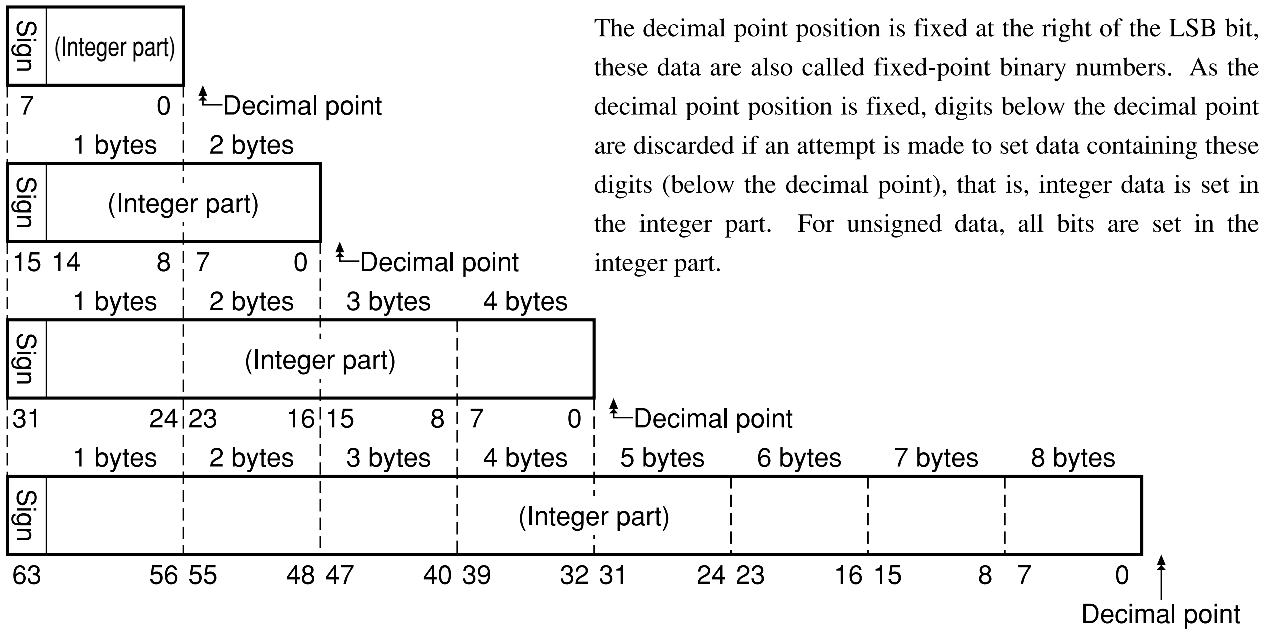
Number of transfer bytes	1, 2, 4, or 8 bytes
Byte transfer order	Bytes are transferred sequentially, starting at the most significant byte.
Signed binary code	LSD ········· Right-justify MSB ········ Sign bit When the data length is shorter than the field length, pad the remaining field with MSBs.
Unsigned binary code	LSD ········· Right-justify MSB ········ Not a sign bit Pad unused high-order bits with 0s.

Ranges of signed and unsigned 1 byte (8 bit) and 2 byte (16 bit) integer data are shown below.

8-Bit Binary	With Sign	No Sign
10000000	–128	128
10000001	–172	129
10000010	–126	130
11111101	–3	253
11111110	–2	254
11111111	–1	255
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
01111101	125	125
01111110	126	126
01111111	127	127

16-Bit Binary	With Sign	No Sign
1000000000000000	–32768	32768
1000000000000001	–32767	32769
1000000000000010	–32766	32770
1111111111111101	–3	65533
1111111111111110	–2	65534
1111111111111111	–1	65535
0000000000000000	0	0
0000000000000001	1	1
0000000000000010	2	2
0000000000000011	3	3
0111111111111101	32765	32765
0111111111111110	32766	37266
0111111111111111	32767	32767

Internal representations of signed 1, 2, 3, 4, and 8 byte integer data are shown below. When the sign bit is 0, it indicates positive data. When a sign bit is 1, it indicates negative data.



The decimal point position is fixed at the right of the LSB bit, these data are also called fixed-point binary numbers. As the decimal point position is fixed, digits below the decimal point are discarded if an attempt is made to set data containing these digits (below the decimal point), that is, integer data is set in the integer part. For unsigned data, all bits are set in the integer part.

(4) Floating-point binary data

Floating-point binary data, whether it is <PROGRAM DATA> or <RESPONS DATA>, is used as <ARBITRARY BLOCK>-type transfer data. Our products do not support floating-point binary data; however, general specifications are explained below.

Floating-point binary data must consists of the following three fields:

(a) Sign field	(sign bit)
(b) Exponent field	(exponent bit)
(c) Mantissa field	(mantissa bit)

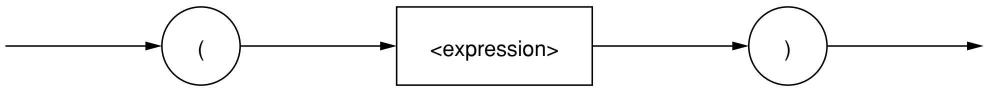
Numeric data having a decimal point is handled here. It has two types of precision: single precision and double precision. Field structures and transfer orders are shown in the Table 5-4. Meanings of symbols are as follows:

Table 5-4

Precision	Number of transfer bytes	Field structure and transfer order								
		Transfer byte	DIO line							
			8	7	6	5	4	3	2	1
Single precision	4 bytes	1st byte	S	EM	E	E	E	E	E	E
		2nd byte	EL	FM	F	F	F	F	F	F
		3rd byte	F	F	F	F	F	F	F	F
		4th byte	F	F	F	F	F	F	F	FL
		Sign bit :	1 bit							
		Exponnent bit :	8 bits (+127 to –126)							
		Mantissa bit :	23 bits							
Double precision	8 bytes	1st byte	S	EM	E	E	E	E	E	E
		2nd byte	E	E	E	EL	FM	F	F	F
		3rd to 7th byte	F	F	F	F	F	F	F	F
		8th byte	F	F	F	F	F	F	F	FL
		Sign bit :	1 bit							
		Exponnent bit :	11 bits (+1023 to –1022)							
		Mantissa bit :	52 bits							

5.3.7 <EXPRESSION PROGRAM DATA>

The <EXPRESSION PROGRAM DATA> element sends the expression for obtaining a scalar, vector, matrix, or string value to a device, allowing the device to calculate a value in place of the controller. Its coding syntactical chart is as follows:



<expression>:	A sequence of ASCII characters represented by ASCII code bytes 20-7E (decimal values = 32 to 126), excluding the following six characters:

" .... double quotation mark
# .... number code (sharp)
'........... single quotation mark
(.......... parenthesis (left)
) .... parenthesis (right)
: .... semi-colon

If a+b+c is written as <expression>, then the above syntactical chart will be expressed as

(a+b+c)

To transfer this to a device, <PROGRAM DATA> discussed on pages 4-16 to 4-35 can be used with the exception of the <INDEFINITE LENGTH ARBITRARY BLOCK PROGRAM DATA>. Upon receipt of (<expression>), the device obtains the solution to this expression.

NOTE:

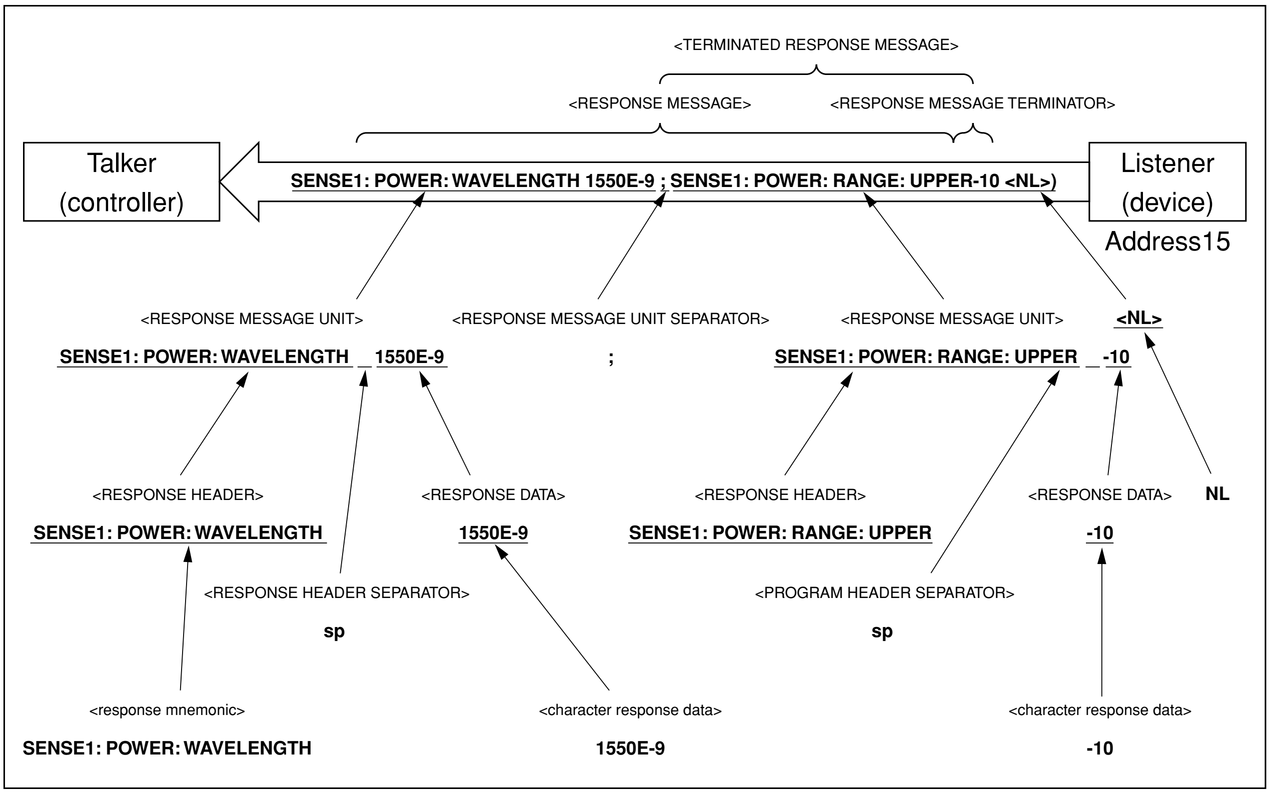
The MT9810A does not support the <expression> function. If calculation of an expression is required, the solution to the expression must be obtained by the controller and the resultant numeric data must be transferred to the device as <PROGRAM DATA>.

Section 6   Talker Output Format

Device messages transferred between the controller and devices are classified into program messages and response messages. This section explains the formats of the program messages sent from a talker to a listener.

Typical response messages are: measurement result, setting, and status information. Response messages are classified into those with header and those without header.

The following diagram shows that when the message unit of a setting wavelength query and a measurement range query is sent to the power meter unit inserted into Channel 1, each response message is sent from the device to the controller in ASCII strings with a header.



<TERMINATED RESPONSE MESSAGE>
<RESPONSE MESSAGE>
<RESPONSE MESSAGE TERMINATOR>
Talker (controller)
SENSE1: POWER: WAVELENGTH 1550E-9 ; SENSE1: POWER: RANGE: UPPER-10 <NL>)
Listener (device) Address15
<RESPONSE MESSAGE UNIT>
<RESPONSE MESSAGE UNIT SEPARATOR>
<RESPONSE MESSAGE UNIT>
<NL>
SENSE1: POWER: WAVELENGTH _ 1550E-9
;
SENSE1: POWER: RANGE: UPPER _ -10
<RESPONSE HEADER>
<RESPONSE DATA>
<RESPONSE HEADER>
<RESPONSE DATA>
NL
SENSE1: POWER: WAVELENGTH
1550E-9
SENSE1: POWER: RANGE: UPPER
-10
<RESPONSE HEADER SEPARATOR>
sp
<PROGRAM HEADER SEPARATOR>
sp
<response mnemonic>
<character response data>
<character response data>
SENSE1: POWER: WAVELENGTH
1550E-9
-10

The above operation portions can be described as a program, as shown below.
Call Send (0,15,"SENSE1:POWER:WAVELENGTH?;SENSE1:POWER:RANGE:UPPER?",NLend)†1
Call Receive (0,15,buf1,NLend)†2

NOTE †1:

Sends a query message unit of the setting wavelength and measurement range.

NOTE †2:

If the terminator NL is detected, the response message SENSE1:POWER:WAVELENGTH 1550E-9; SENSE1:POWER:RANGE:UPPER -10 are read into buf1.

A response message is a sequence of functional elements, the minimum units that can represent functions, as is the case with the program message. In the above figure, functional elements are indicated by uppercase characters enclosed in the brackets (< >). Functional elements are further classified into coding elements which are indicated by lowercase characters enclosed in the brackets (< >).

The following pages explain talker output formats focusing on the differences from listener input formats starting with the Section 6.1 "Differences in Syntax between Listener Input Formats and Talker Output formats."

6.1 Differences in Syntax between Listener Input Formats and Talker Output formats

Significant differences in syntax between the listener and the talker are as follows:

Listener format	Program can be written flexibly so that devices can accept program messages from the controller. If a program message involves some description errors, it can execute its function normally. For example, unlimited number of <white space> element can be used in order to make an easy-to-read program.
Talker format	Messages are output following strictly defined syntactical rules to allow the controller to accept the response messages from the device. Therefore, the syntax of response messages permits only one notation for a function.

The summary of the differences in output format between the listener and the talker is shown in the Table 6-1. In this table, "0/1 or more spaces" indicates <white space>.

Table 6-1

Item	Listener input program message syntax	Talker output response message syntax
Characteristic	(Flexible)	(Strict)
Alphabetic characters	No difference between uppercase	Uppercase characters only
Character before and after NR3 exponent part E	0 or more spaces + E/e + 0 or more spaces	Uppercase character E only
+ sign of NR3 exponent part	Omissible	Required
<white space>	Two or more white spaces can be written before/after a separator or before a terminator.	Not used
Message unit	(a) Header with program data (b) Header without program data	(a) Data with header (b) Data without header
Unit separator	0 or more spaces + Semicolon	Semi-colon only
Space before header	0 or more spaces + Header	Header only
Header separator	Header + 1 or more spaces	Header + One $20 [†1]
Data separator	0 or more spaces + Comma + 0 or more spaces	Comma only
Terminator	0 or more spaces + One of $\left\{\begin{array}{l}\text{NL}\\\text{EOI}\\\text{NL+EOI}\end{array}\right\}$	NL+EOI

NOTE:

ASCII code byte 20 (decimal value 32 = ASCII character SP, space)

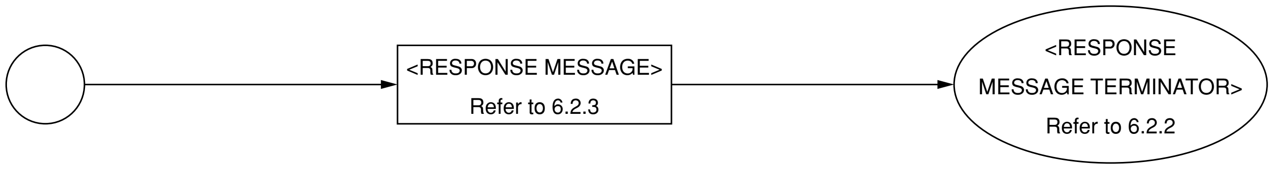
6.2 Response Message Functional Elements

Response messages output from a talker are terminated with an NL∧END signal, allowing the controller to accept these messages. Functional elements of these response messages are explained here.

Rules for syntactical chart notation are the same as those for program messages. Refer to the Section 5 "Listener Input Format" for the information. Also functional and coding elements, which are the same as those of program messages, are not explained in this section. Refer to the Section 5 "Listener Input Format" as well.

6.2.1 <TERMINATED RESPONSE MESSAGE>

<TERMINATED RESPONSE MESSAGE> is defined as follows:

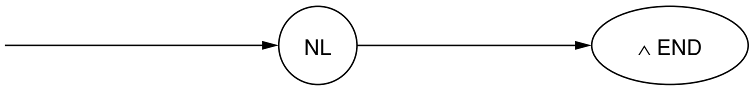


<TERMINATED RESPONSE MESSAGE> is a data message having all the necessary functional elements to be sent from a talker to a device.

To complete transfer of <RESPONSE MESSAGE>, <RESPONSE MESSAGE TERMINATOR> is added at the end of <RESPONSE MESSAGE>.

6.2.2 <RESPONSE MESSAGE TERMINATOR>

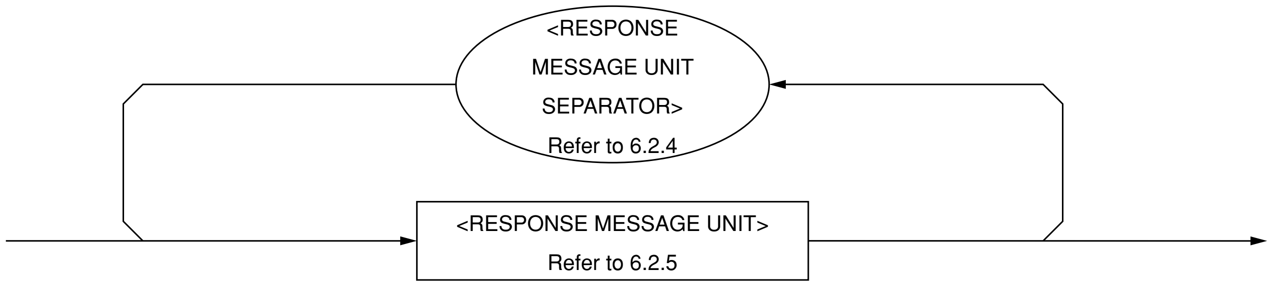
<RESPONSE MESSAGE TERMINATOR> is defined as follows:



<RESPONSE MESSAGE TERMINATOR> is placed after the last <RESPONSE MESSAGE UNIT> to terminate the sequence of one or more fixed-length <RESPONSE MESSAGE UNIT> elements.

6.2.3 <RESPONSE MESSAGE>

<RESPONSE MESSAGE> is defined as follows:

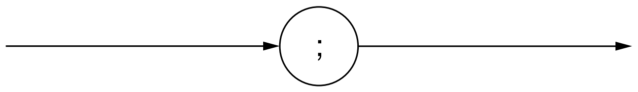


<RESPONSE MESSAGE> is a sequence of one or more <RESPONSE MESSAGE UNIT> elements.

The <RESPONSE MESSAGE UNIT> element is a single message sent from a device to a controller. A <RESPONSE MESSAGE UNIT SEPARATOR> is used as a separator for separating multiple <RESPONSE MESSAGE UNIT> elements.

6.2.4 <RESPONSE MESSAGE UNIT SEPARATOR>

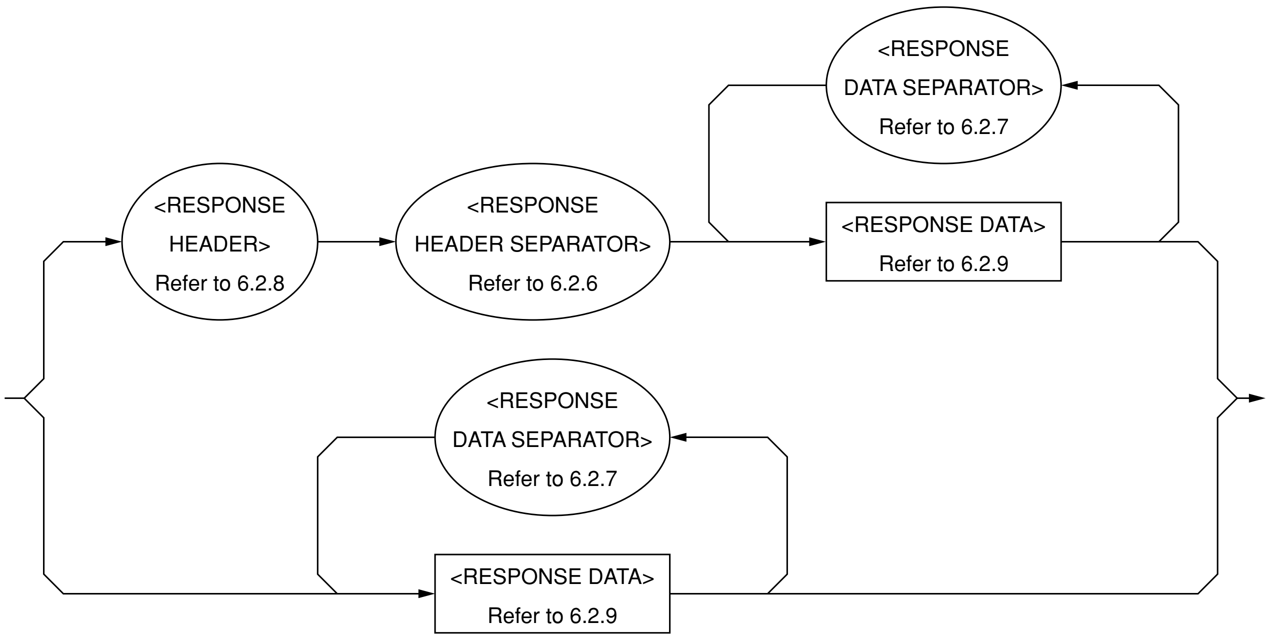
<RESPONSE MESSAGE UNIT SEPARATOR> is defined as follows:



<RESPONSE MESSAGE UNIT SEPARATOR> is used to separate <RESPONSE MESSAGE UNIT> elements with a <UNIT SEPARATOR>, or a semi-colon (;), when outputting a sequence of multiple <RESPONSE MESSAGE UNIT> elements as one <RESPONSE MESSAGE>.

6.2.5 <RESPONSE MESSAGE UNIT>

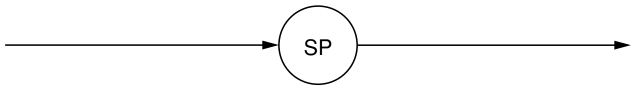
<RESPONSE MESSAGE UNIT> is defined as follows:



There are two kinds of useage for <RESPONSE MESSAGE UNIT>. One is <RESPONSE MESSAGE UNIT> with header, which returns the result of processing the program-message-set information accurately. The other is <RESPONSE MESSAGE UNIT> without header, which returns only the measurement result.

6.2.6 <RESPONSE HEADER SEPARATOR>

<RESPONSE HEADER SEPARATOR> is defined as follows:



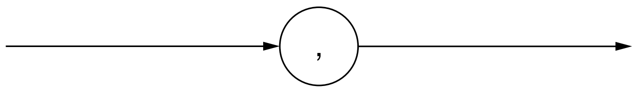
<RESPONSE HEADER SEPARATOR> is one space written after <RESPONSE HEADER> to be separated from <RESPONSE DATA>.

The space SP corresponds to ASCII code byte 20 (decimal 32).

In a <RESPONSE MESSAGE> with header, a space must always exist between the header and the data as a <RESPONSE HEADER SEPARATOR>. The separator indicates the end of the <RESPONSE HEADER> as well as the beginning of <RESPONSE DATA> at the same time.

6.2.7 <RESPONSE DATA SEPARATOR>

<RESPONSE DATA SEPARATOR> is defined as follows:



When multiple <RESPONSE DATA> elements are output, <RESPONSE DATA SEPARATOR> must be placed between thses data elements.

6.2.8 <RESPONSE HEADER>

The format of <RESPONSE HEADER> is the same as that of <COMMAND PROGRAM HEADER> described in the Section 5.2.8 "<COMMAND PROGRAM HEADER>" with the exception of the following three points:

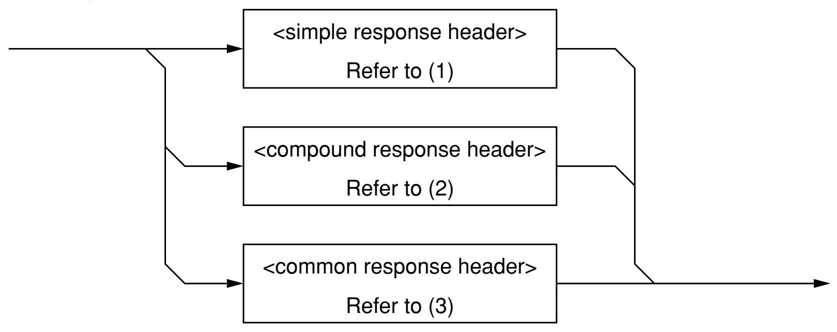
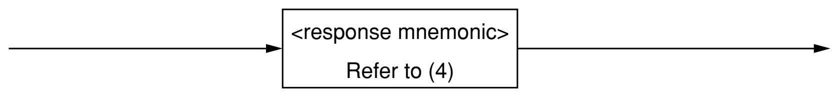
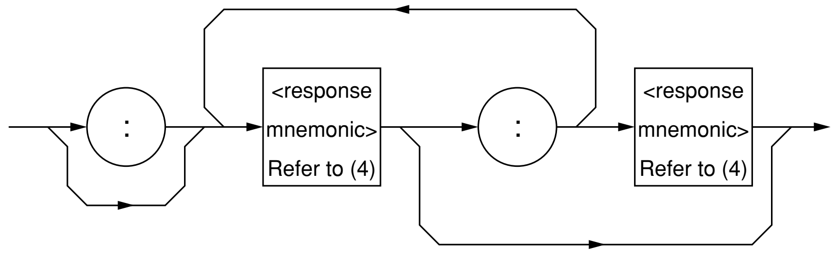
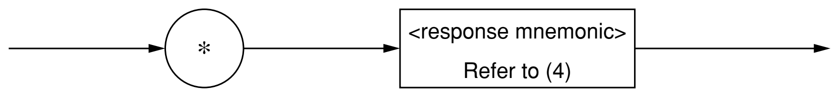
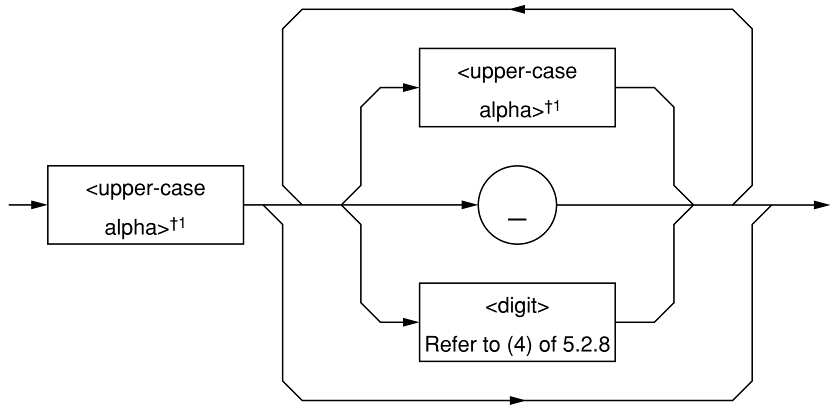
(1) Characters that can be used in <response mnemonic> are specified. For alphanumeric characters, only uppercase characters must be used. Other points are the same as those of <program mnemonic>.

(2) A space cannot be written before a <RESPONSE HEADER>, while it can be written before a <PROGRAM HEADER>.

(3) Only one space can be written before a <RESPONSE HEADER>, while two or more spaces can be written before a <PROGRAM HEADER>.

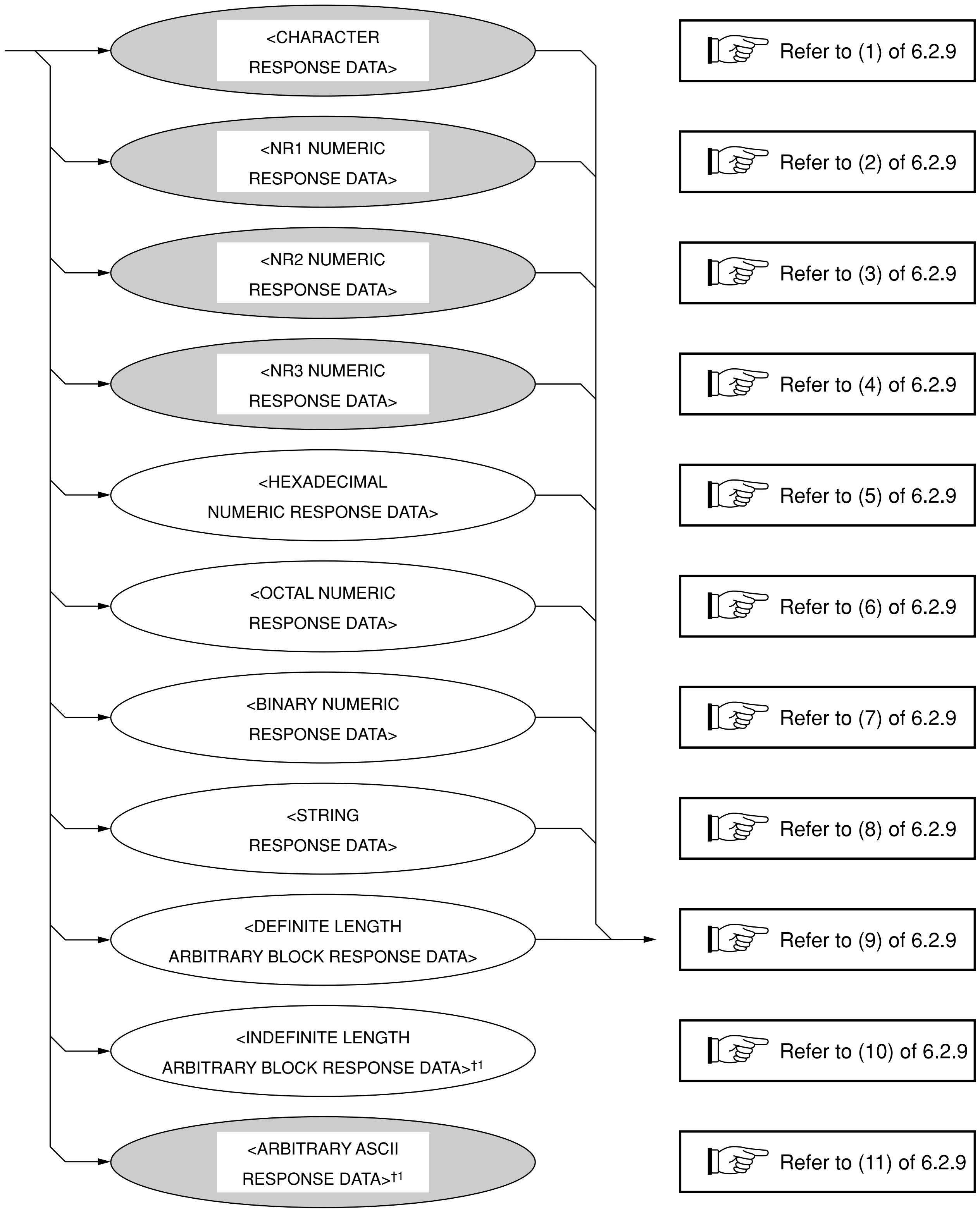
Refer to the Table 6-2 for the response header up to <response mnemonic>.
It should be noted that only uppercase characters must be used in <response mnemonic>. Other points are the same as those of <program mnemonic> described in the Section 5.2.8 "<COMMAND PROGRAM HEADER>."

Table 6-2

Item	Function
RESPONSE HEADER	A header indicates a function of <RESPONSE DATA>. It explains the function with a 12-character-long character-long character string or a <response mnemonic> element that consists of uppercase characters, numeric characters, and/or underline.  <simple response header> Refer to (1) <compound response header> Refer to (2) <common response header> Refer to (3) (1) <simple response header> is defined as follows.  (2) <compound response header> is defined as follows.  : <response mnemonic> Refer to (4) : <response mnemonic> Refer to (4) (3) <common response header> is defined as follows.  (4) <response mnemonic> is defined as follows.  <upper-case alpha>†1 <upper-case alpha>†1 _ <digit> Refer to (4) of 5.2.8 **NOTE †1:** <upper-case alpha> ASCII code bytes 41 to 5A (decimal values 65 to 90 = uppercase characters A to Z)

6.2.9 <RESPONSE DATA>

There are 11 types of <RESPONSE DATA> elements. Among these, the MT9810A transfers the <RESPONSE DATA> shown in the hollow squares surrounded by a shade. The <RESPONSE DATA> to be returned depends on the query message.



<CHARACTER RESPONSE DATA>
Refer to (1) of 6.2.9
<NR1 NUMERIC RESPONSE DATA>
Refer to (2) of 6.2.9
<NR2 NUMERIC RESPONSE DATA>
Refer to (3) of 6.2.9
<NR3 NUMERIC RESPONSE DATA>
Refer to (4) of 6.2.9
<HEXADECIMAL NUMERIC RESPONSE DATA>
Refer to (5) of 6.2.9
<OCTAL NUMERIC RESPONSE DATA>
Refer to (6) of 6.2.9
<BINARY NUMERIC RESPONSE DATA>
Refer to (7) of 6.2.9
<STRING RESPONSE DATA>
Refer to (8) of 6.2.9
<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>
Refer to (9) of 6.2.9
<INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>†1
Refer to (10) of 6.2.9
<ARBITRARY ASCII RESPONSE DATA>†1
Refer to (11) of 6.2.9

NOTE†1:

<INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> and <ARBITRARY ASCII RESPONSE DATA> is terminated with NL∧END after the last byte has been transferred.

Table 6-3

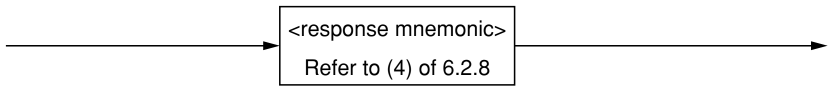
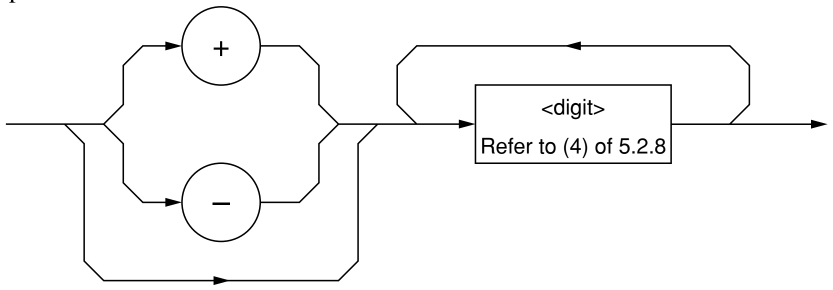
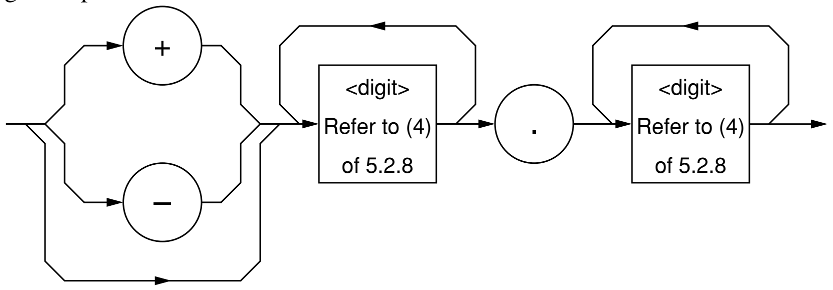
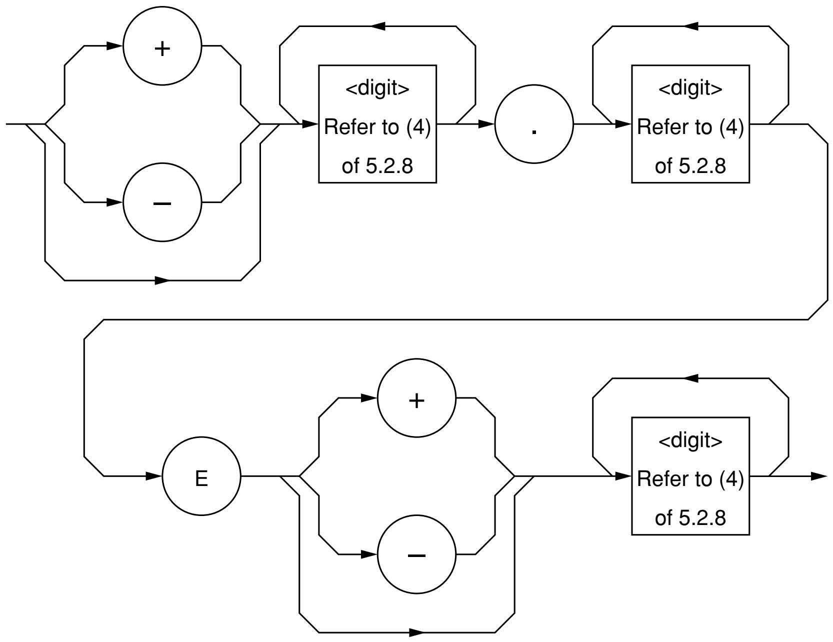
Item	Function
(1) CHARACTER RESPONSE DATA	Data consisting of the same character string as that of <response mnemonic>. Accordingly, the character string always begins with an uppercase character and its length is less than 12 characters. Numeric parameters must not be used. 
(2) NR1 NUMERIC RESPONSE DATA <Example> 123 +123 –1234	Integer data, i.e., a decimal value of an integer that has neither decimal point nor exponent. 
(3) NR2 NUMERIC RESPONSE DATA <Example> 12.3 +12.34 –12.345	Fixed-point data, i.e., a decimal value other than integers or a decimal value having an exponent.  + – <digit> Refer to (4) of 5.2.8 . <digit> Refer to (4) of 5.2.8
(4) NR3 NUMERIC RESPONSE DATA <Example> 1.23E+4 +12.34E–5 –12.345E+6 • Lowercase characters cannot be used for E. • E must not be preceded and followed by a space. • + in the exponent part is mandatory. • + in the mantissa part is mandatory.	Fixed-point data, i.e., a decimal value having an exponent.  + – <digit> Refer to (4) of 5.2.8 . <digit> Refer to (4) of 5.2.8 E + – <digit> Refer to (4) of 5.2.8

Table 6-3 (continue)

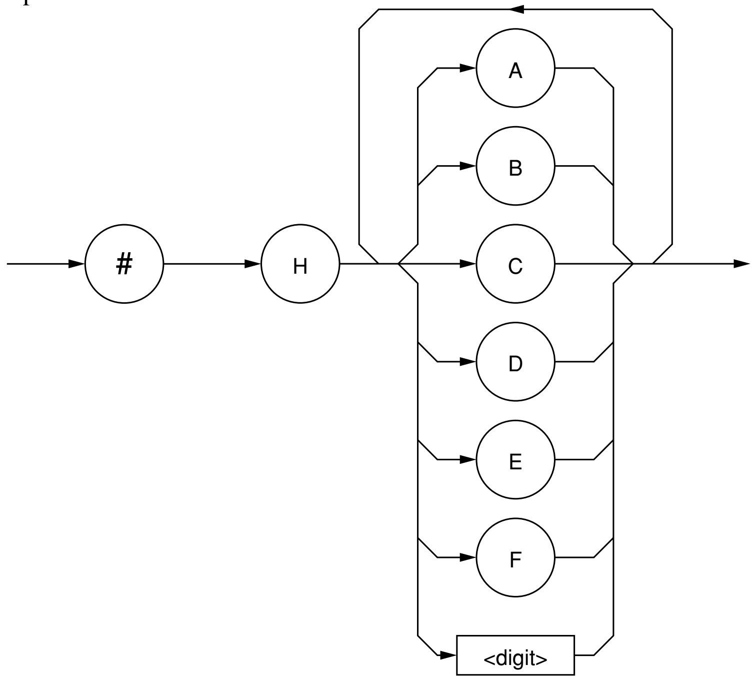
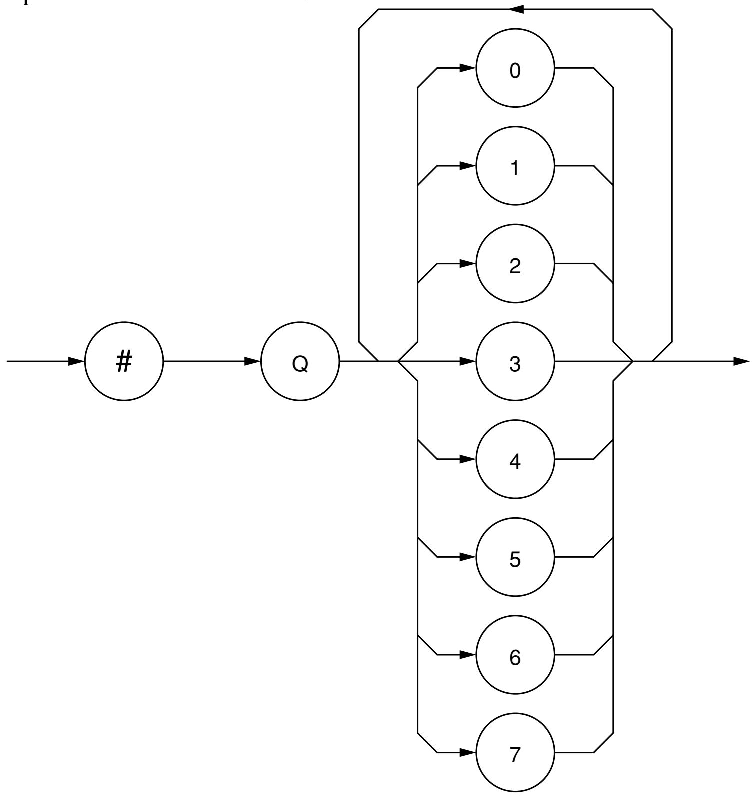
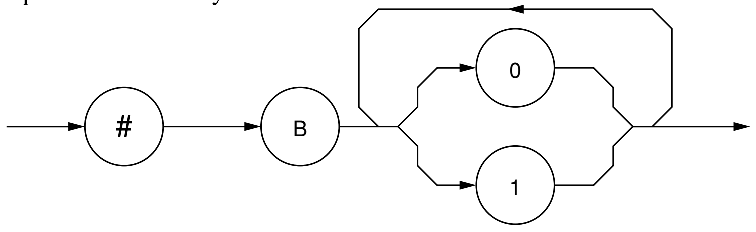
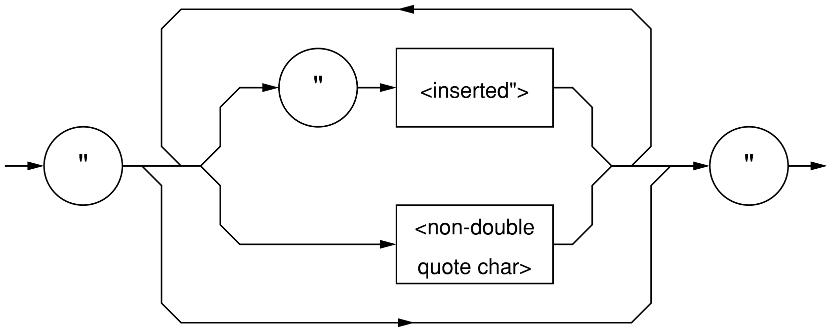
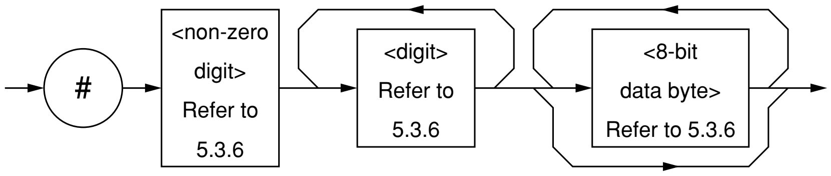
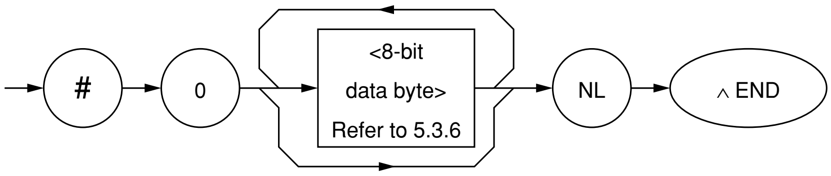
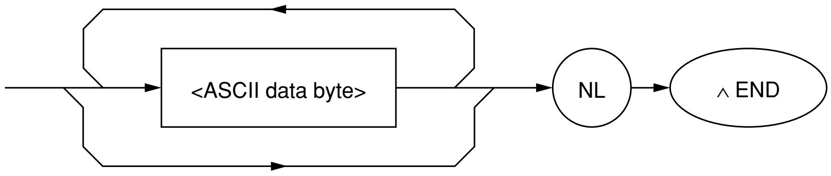
Item	Function
(5) HEXADECIMAL NUMERIC RESPONSE DATA <Example> #HABC123 #H2DC3 #H8301	Data represented in hexadecimal notation. 
(6) OCTAL NUMERIC RESPONSE DATA <Example> #Q37 #Q26703 #Q30562	Data represented in octal notation. 
(7) BINARY NUMERIC RESPONSE DATA <Example> #B011101 #B1011 #B1011	Data represented in binary notation. 

Table 6-3 (continue)

Item	Function
(8) STRING RESPONSE DATA <Example> "This is a text" "Say,""Hello""."	Any ASCII 7-bit code can be used. The character string must be enclosed with double quotation marks ("). When a character string contains double quotation marks, two identical quotation marks must be written in succession per quotation mark. Since a CR, LF, of space can be used, this element is suitable for outputting a text to the printer or CRT.  " " <inserted"> <non-double quote char> "
(9) DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA <Example> Transferring 11256099D in a 4-byte blocks ↓ #1400ABC123	Fixed-point 8-bit binary block data. It is suitable for transferring large-volume data, 8-bit extended ASCII code, and non-display data. Refer to the Section 5.3.6 "<ARBITRARY BLOCK PROGRAM DATA>" for more details on individual elements.  # <non-zero digit> Refer to 5.3.6 <digit> Refer to 5.3.6 <8-bit data byte> Refer to 5.3.6
(10) INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA <Example> Indefinite-length –250, –50, 120, ... are transferred ↓ #0FF06FFCE0078	Indefinite-length 8-bit binary block data. #0 must be written before the first data. The last data must be followed by NL∧END for termination.  # 0 <8-bit data byte> Refer to 5.3.6 NL ∧ END
(11) ARBITRARY ASCII RESPONSE DATA <Example 1> <ASCII Byte><ASCII Byte>NL∧END <Example 2> NL∧END	ASCII data bytes except NL character transferred in succession. The last data must be followed by NL∧END for termination. 

Section 7   Common Commands

This section explains common commands and common query commands specified by IEEE 488.2. These common commands are not bus commands which are used as interface messages. Like device messages, the common commands are data messages used when the bus data mode (or the ATN line) is False. These commands can be applied to all measuring instruments, including those of other companies, that comply with IEEE 488.2. IEEE 488.2 common commands always begin with an asterisk (*).

7.1 Classification of Supported Commands and References

MT9810A-supported commands discussed previously are classified by function group as shown in the Table 7-1. Details on these commands are given in alphabetical order on the next and subsequent pages.

Table 7-1

Group	Function by group	Mnemonic
System data	Information about device connected to the system (e.g., manufacturer name, type name, and serial number) is returned.	*IDN? *OPT?
Internal operation	Control inside the device: (a) Resetting of device at level 3 (b) Self-test and error detection inside the device	*RST *TST?
Synchronization	A device is synchronized with the controller by: (a) Service request wait (b) Device output queue wait (c) Forced sequential execution	*OPC *OPC? *WAI
Status and event	A status byte consists of a status summary message. Summary bits of the status summary message are set by a standard event register, output queue, and extended event register (or an extended queue). Three commands and four queries are provided to set, clear, validate, and invalidate the data in these registers and queues and to know the register settings using queries.	*CLS *ESE *ESE? *ESR? *SRE *SRE? *STB?

∗CLS Clear Status Command

(Clears status byte registers)

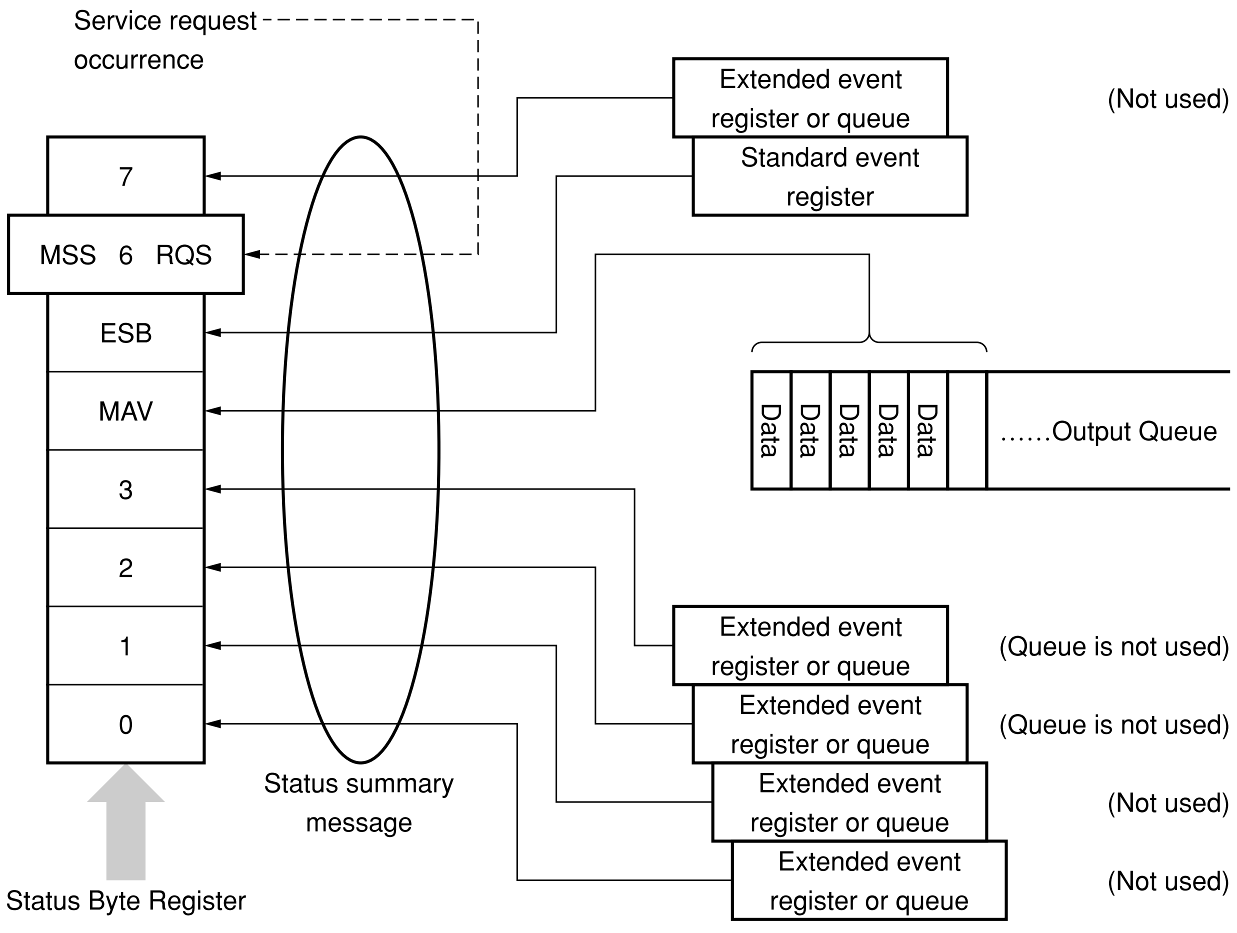
(1) Format

∗CLS

(2) Explanation

The ∗CLS common command clears all status structures (i.e., event registers and queues) except an output queue and its MAV summary messages, thus clearing the corresponding summary messages.

Issuing a ∗CLS command after <PROGRAM MESSAGE TERMINATOR> or before <QUERY MESSAGE UNIT> will clear all status bytes. With this method, all unread messages in the output queue will also be cleared. Values set in enable registers are not changed by the ∗CLS command.



∗ESE Standard Event Status Enable Command

(Sets or clears the standard event status enable register)

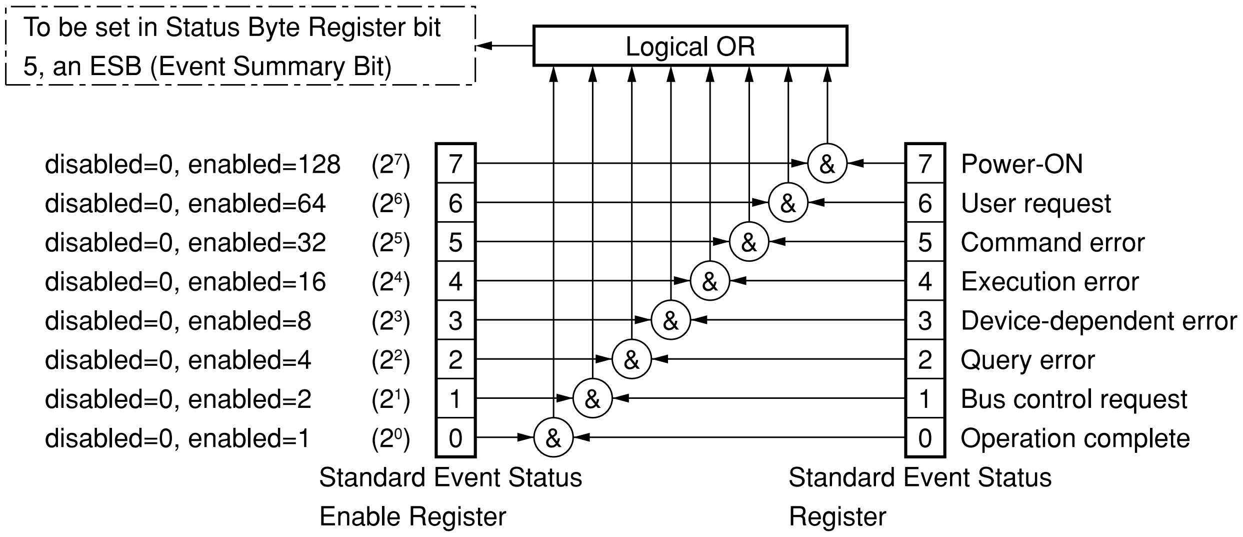
(1) Format

∗ESE<HEADER SEPARATOR><DECIMAL NUMERIC PROGRAM DATA>

In this particular format <DECIMAL NUMERIC PROGRAM DATA> is a value rounded to an integer, 0 to 255 (base is 2 and binary weights are assigned).

(2) Explanation

The total of values ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, and/or $2^7 = 128$) corresponding to the standard event status enable register bits 1, 2, 3, 4, 5, 6, and/or 7 that are to be enabled becomes program data. The value of the bit to be disabled is 0.


To be set in Status Byte Register bit 5, an ESB (Event Summary Bit)
Logical OR
disabled=0, enabled=128 (2^7) 7
disabled=0, enabled=64 (2^6) 6
disabled=0, enabled=32 (2^5) 5
disabled=0, enabled=16 (2^4) 4
disabled=0, enabled=8 (2^3) 3
disabled=0, enabled=4 (2^2) 2
disabled=0, enabled=2 (2^1) 1
disabled=0, enabled=1 (2^0) 0
&
7 Power-ON
6 User request
5 Command error
4 Execution error
3 Device-dependent error
2 Query error
1 Bus control request
0 Operation complete
Standard Event Status Enable Register
Standard Event Status Register

∗ESE? Standard Event Status Enable Quer

(Returns the current value of the standard event status enable register)

(1) Format

∗ESE?

(2) Explanation

The value (NR1) of the standard event status enable register is returned.

(3) Response message

NR1 = 0 to 255

∗ESR? Standard Event Status Register Query

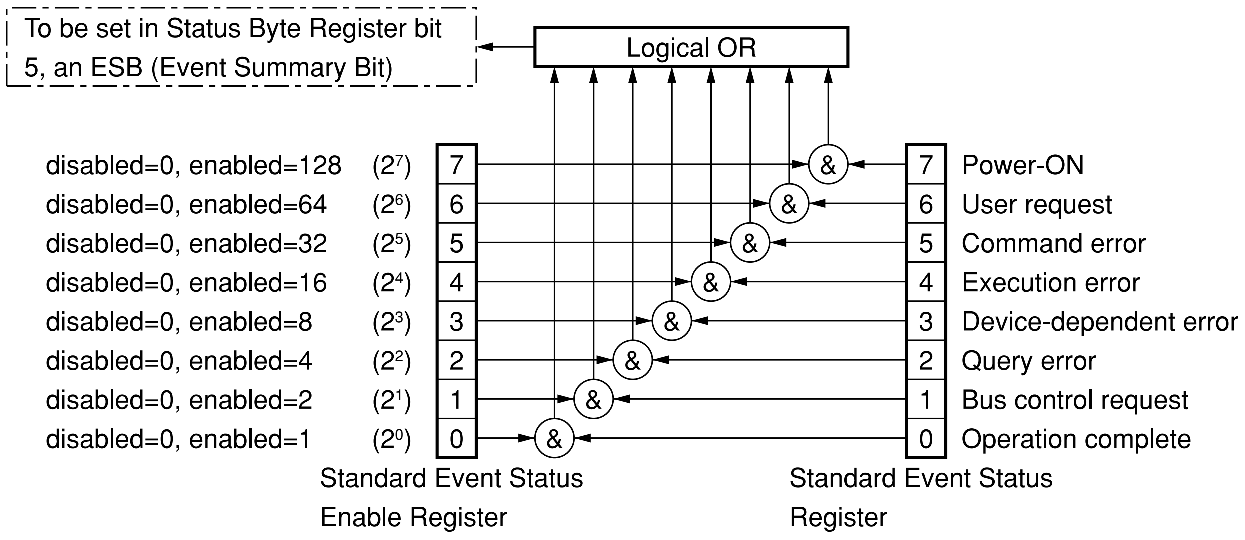
(Returns the current value of the standard event status register)

(1) Format

∗ESR?

(2) Explanation

The current value NR1 of the standard event status register is returned. The total of values ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, and/or $2^7 = 128$) corresponding to the standard event status enable register bits 1, 2, 3, 4, 5, 6, and/or 7 that are enabled becomes NR1. When the response is read (e.g., line 40), this register is cleared.


To be set in Status Byte Register bit 5, an ESB (Event Summary Bit)
Logical OR
disabled=0, enabled=128 (2^7) 7
disabled=0, enabled=64 (2^6) 6
disabled=0, enabled=32 (2^5) 5
disabled=0, enabled=16 (2^4) 4
disabled=0, enabled=8 (2^3) 3
disabled=0, enabled=4 (2^2) 2
disabled=0, enabled=2 (2^1) 1
disabled=0, enabled=1 (2^0) 0
&
7 Power-ON
6 User request
5 Command error
4 Execution error
3 Device-dependent error
2 Query error
1 Bus control request
0 Operation complete
Standard Event Status Enable Register
Standard Event Status Register

(3) Response message NR1

NR1 = 0 to 255

∗IDN? Identification Query

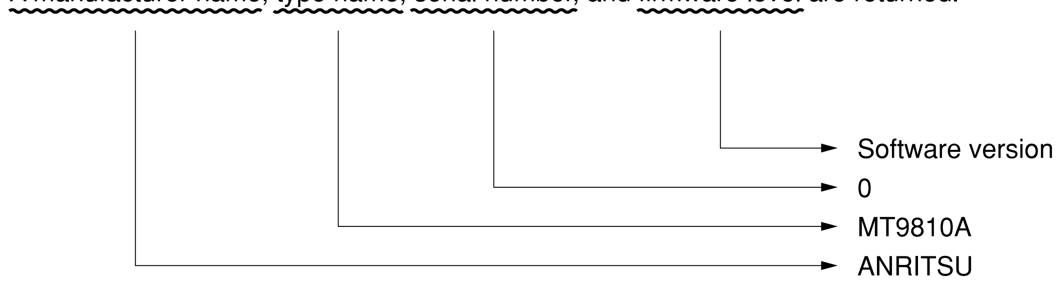
(Returns the manufacturer name, type name, serial number, and firmware level of the product)

(1) Format

∗IDN?

(2) Explanation

A manufacturer name, type name, serial number, and firmware level are returned.



When the manufacturer of the product, whose type name, serial number, and software/hardware version number are Anritsu, 0, and 1 respectively. Sending a common query ∗IDN? to a device will return a response message consist of the above four fields.

Field 1	Product manufacturer (e.g., ANRITSU)
Field 2	Type name
Field 3	Serial number (e.g., 0)
Field 4	Firmware version No. (control software version and optical software version)

Return ASCII character "0" to not to return a serial number and firmware version in fields 3 and 4.

(3) Response message

A response message which consists of the above four fields separated by commas is sent as <ARBITRARY ASCII RESPONSE DATA>.

<Field 1>,<Field 2>,<Field 3>,<Field 4>

Overall length of the response message is less than 72 characters.

∗OPC Operation Complete Command

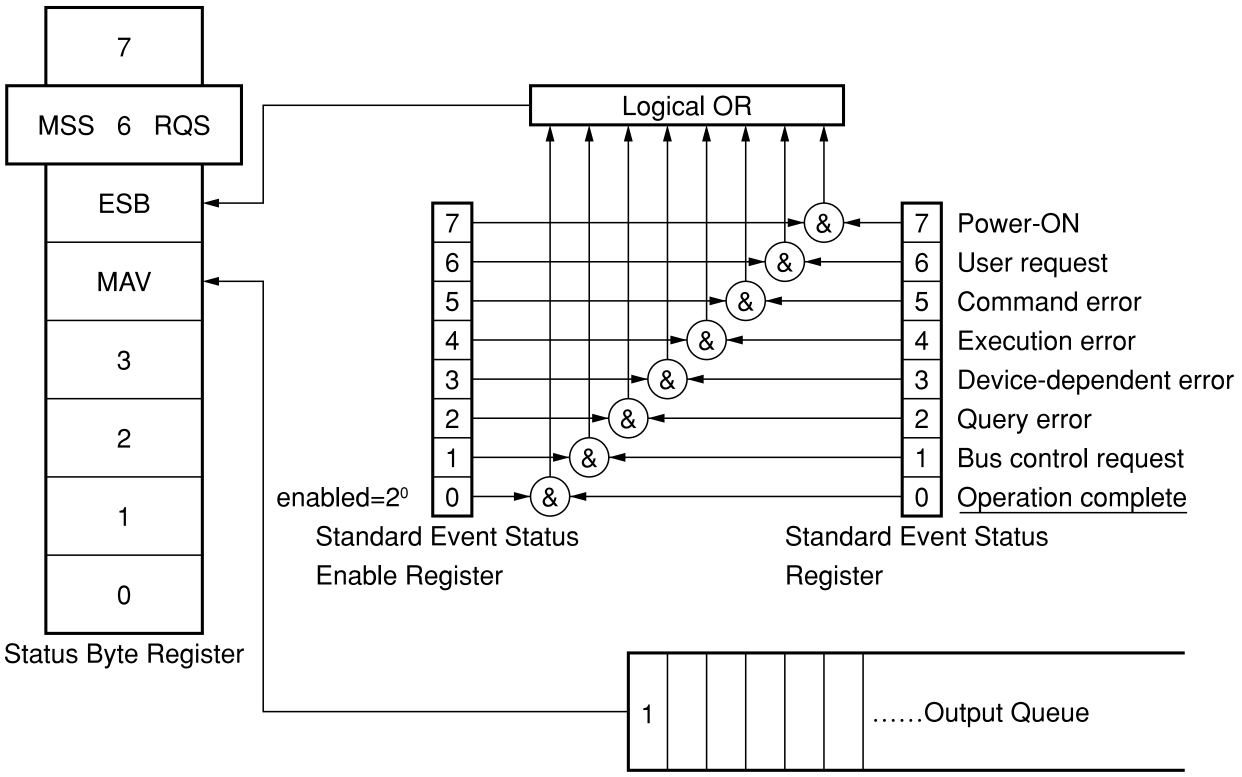
(Sets bit 0 of the standard event status register when device operations have been completed)

(1) Format

∗OPC

(2) Explanation

When all the pending device operations have been completed, standard event status register bit 0 (i.e., operation complete bit) is set. However, since the MT9810A does not have an overlap command, the ∗OPC command counts for nothing.



∗OPC? Operation Complete Query

(When device operations have been completed, sets "1" in the output queue to generate an MAV summary message)

(1) Format

∗OPC?

(2) Explanation

When all the pending device operations have been completed, "1" is set in the output queue, waiting for an MAV summary message to occur.

(3) Response message

"1" is returned as <NR1 NUMERIC RESPONSE DATA>.

∗RST Reset Command

(Rests a device at level 3)

(1) Format

∗RST

(2) Explanation

The ∗RST (Reset) command resets a device at level 3 (Refer to the Table 4-1). At level 3, the following items are initialized:

(a) Device-dependent functions and states are restored to known states irrespective of the device history.
(b) The macro defined by ∗DDT command is restored to the device-defined state.
(c) A mode in which macro operation is disabled and macros are not accepted, is set. Macro definitions are restored to the designer-specified states.
(d) The specified device is set in the OCIS. The operation complete bit cannot be set in the standard event status register.

☞ **Section 8.1**

(e) The specified device is set in the OQIS. The operation complete bit cannot be set in the output queue. The MAV bit is cleared.

The ∗RST command does not affect the following:

(a) IEEE 488.1 interface state
(b) Device address
(c) Output queue
(d) Service request enable register
(e) Standard event status enable register
(f) Power-on-status-clear flag setting
(g) Calibration data affecting device standard
(h) RS-232C interface condition

∗OPT? Option Identification Query

(Reports an installed option list)

(1) Format

∗OPT?

(2) Explanation

States of installed options are returned using 1 or 0.

(3) Response message

A response message which consists of the above three fields separated by commas is sent as <ARBITRARY ASCII RESPONSE DATA>.
Since there is no option now "0" is returned.

∗SRE Service Request Enable Command

(Sets a service request enable register bit)

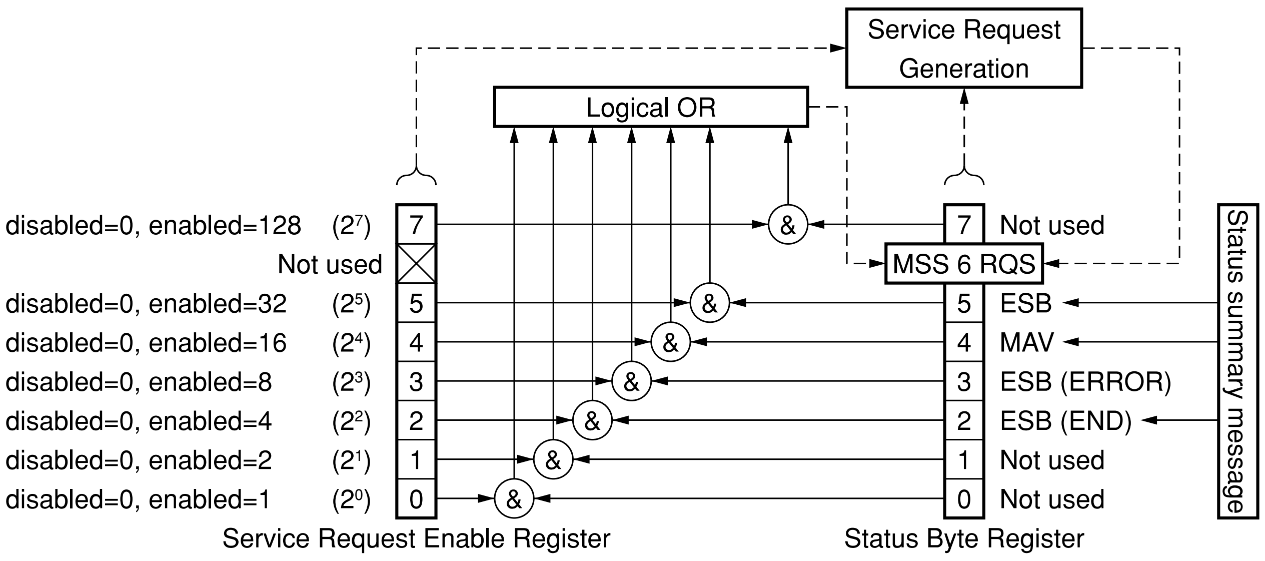
(1) Format

∗SRE<HEADER SEPARATOR><DECIMAL NUMERIC PROGRAM DATA>

In this particular format <DECIMAL NUMERIC PROGRAM DATA> is a value rounded to an integer, 0 to 255 (base is 2 and binary weights are assigned).

(2) Explanation

The total of values ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, and/or $2^7 = 128$) corresponding to the service request enable register bits 1, 2, 3, 4, 5, 6, and/or 7 that are enabled becomes NR1. The value of the bit to be disabled is 0.


Service Request Generation
Logical OR
disabled=0, enabled=128 (2^7) 7
Not used
disabled=0, enabled=32 (2^5) 5
disabled=0, enabled=16 (2^4) 4
disabled=0, enabled=8 (2^3) 3
disabled=0, enabled=4 (2^2) 2
disabled=0, enabled=2 (2^1) 1
disabled=0, enabled=1 (2^0) 0
&
7 Not used
MSS 6 RQS
5 ESB
4 MAV
3 ESB (ERROR)
2 ESB (END)
1 Not used
0 Not used
Status summary message
Service Request Enable Register
Status Byte Register

∗SRE? Service Request Enable Query

(Returns the current value of the service request enable register)

(1) Format

∗SRE?

(2) Explanation

The value NR1 of the service request enable register is returned.

(3) Response message NR1

Since NR1 = bit 6 (RQS bit) cannot be set, NR1 = 0 to 63 or 128 to 191.

∗STB? Read Status Byte Command

(Returns the current value of the status byte including the MSS bit)

(1) Format

∗STB?

(2) Explanation

The ∗STB? command returns the total of the status register value assigned binary weights and MSS (master Summary Status) summary message value as <NR1 NUMERIC RESPONSE DATA>.

(3) Response message

A response message (<NR1 NUMERIC RESPONSE DATA>) is an integer ranging from 0 to 255. It is the total of status byte register bit values. Status byte register bits 0 to 5 and 7 is assigned weights 1, 2, 4, 8, 16, 32, and 128 respectively, and the MSS bit is assigned weight 64. The MSS indicates that there is at least one reason for requesting a service. The status byte register conditions of MT9810A are summarized in the Table 7-2.

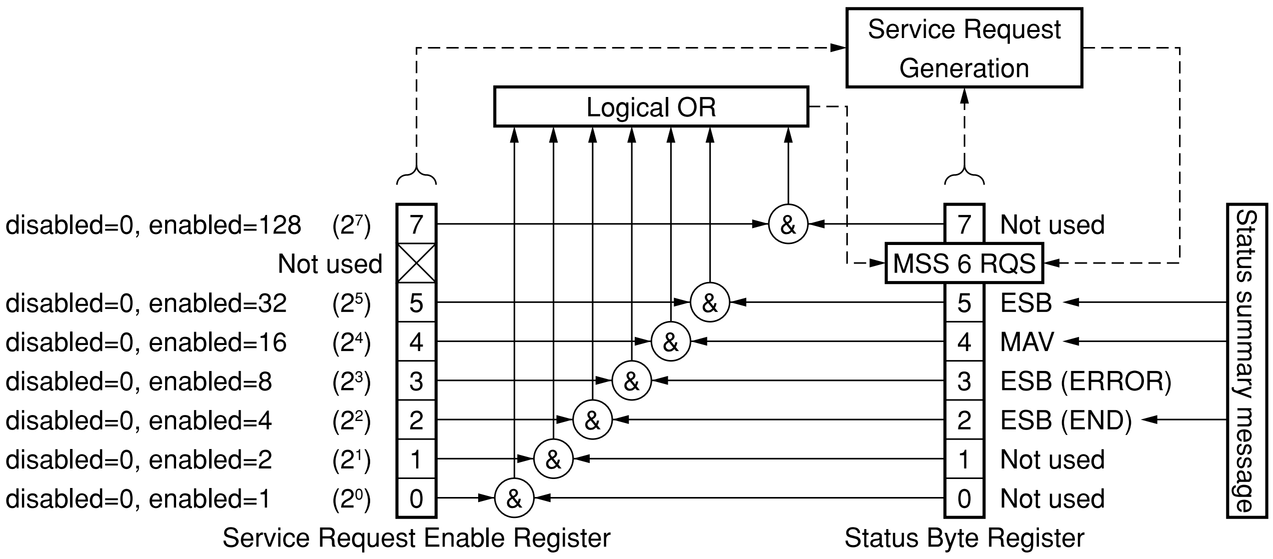

Service Request Generation
Logical OR
disabled=0, enabled=128 (2^7)
Not used
disabled=0, enabled=32 (2^5)
disabled=0, enabled=16 (2^4)
disabled=0, enabled=8 (2^3)
disabled=0, enabled=4 (2^2)
disabled=0, enabled=2 (2^1)
disabled=0, enabled=1 (2^0)
7
5
4
3
2
1
0
&
7 Not used
MSS 6 RQS
5 ESB
4 MAV
3 ESB (ERROR)
2 ESB (END)
1 Not used
0 Not used
Status summary message
Service Request Enable Register
Status Byte Register

Table 7-2

Bit	Bit weights	Bit name	Status byte register conditions	
7	128	————	0 = Not used.	
6	64	MSS	0 = No service is not requested.	1 = A service is requested.
5	32	ESB	0 = No event service has occurred.	1 = An event status has occurred.
4	16	MAV	0 = Data does not exist in the output queue.	1 = Data exists in the output queue.
3	8	ESB (ERROR)	0 = No event status has occurred.	1 = An event status has occurred.
2	4	ESB (END)	0 = No event status has occurred.	1 = An event status has occurred.
1	2	————	0 = Not used.	
0	1	————	0 = Not used.	

∗TST? Self-Test Query

(Conducts an internal self-test and indicates whether any error has occurred)

(1) Format

∗TST?

(2) Explanation

The ∗TST? command conducts a self-test inside the device. The test result is set in the output queue. The data in the output queue indicates that the test has been completed without causing any error. The self-test does not require operator intervention.

(3) Response message

A response message is sent as <NR1 NUMBER RESPONSE DATA>.
Data range = –32767 to 32767

NR1 = – .... The test has been completed without causing any error.
NR1 = 1 .... The test has not been conducted or any error occurred during the test.

∗WAI Wait-to-Continue Command

(Causes the next command to wait until the current command has been executed by the device)

(1) Format

∗WAI

(2) Explanation

The ∗WAI command executes overlap commands as sequential commands.

If the device can start executing the next command while processing a command or query from the controller, the command or query is called an overlap command.

If a ∗WAI command is executed after an overlap command, the next command must wait for the ∗WAI common command to end. This also applies to sequential commands.

However, since the MT9810A does not support overlap commands. The ∗WAI command counts for nothing.

Section 8 Status Structure

This section explains the device status data specified by IEEE 488.2, the status data structure, and the technique of synchronization between a device and a controller.

The status byte (STB) sent to the controller is specified by IEEE 488.1. The bits of the status byte represent a status summary message, providing a summary of the current contents of the data stored in a register or queue.

The following sections explain the status summary message bits, the status data structure for generating these status summary message bits, and the technique of synchronizing a device with the controller using the status messages.

These functions are used to control devices from an external controller via the GPIB interface. These functions, except a few, can also be used to control devices from an external controller via the RS-232C interface.

8.1 IEEE 488.2 Standard Status Model

The diagram shown below is the standard model of the status data structure specified by IEEE 488.2.

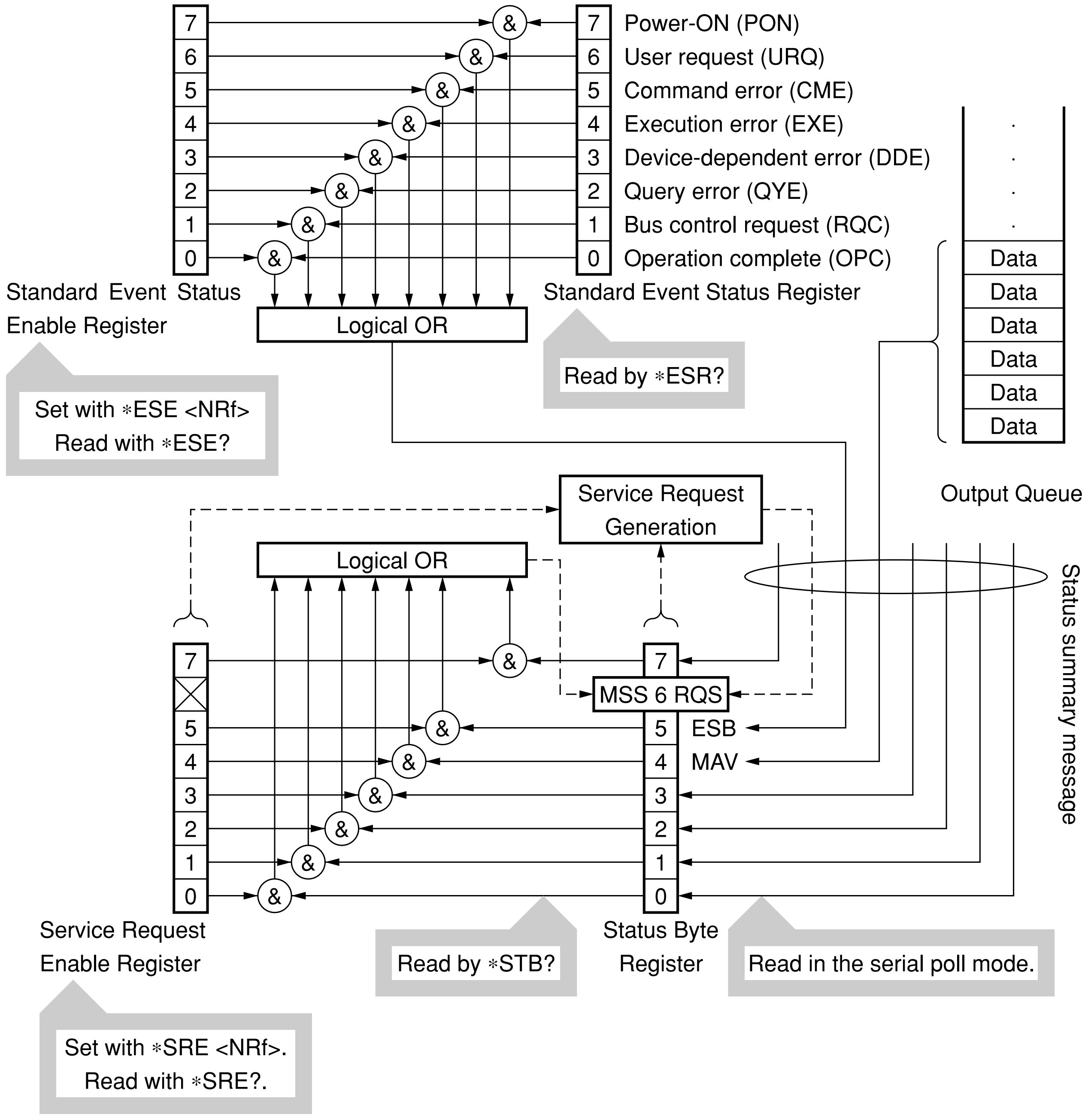


Fig. 8-1 Standard status model

The status model uses an IEEE 488.1 status byte. This status byte consists of seven summary message bits provided by the status data structure. To generate these summary message bits, the status data structure is comprised of two models: a register model and a queue model.

Register model

A pair of registers used to record an event that a device has encountered and a condition. It consists of an event status register and an event status enable register. When the results of ANDing the values of bits of these registers is not 0, the corresponding status register bits are set to 1s. In other cases, the corresponding status register bits are set to 0s. If the result of ORing the values of status register bits is 1, the summary message bit is set to 1. If the result of ORing these bits is 0, the summary message bit is set to 0.

Queue model

A data structure in which status values or information are removed in the same order of which those were entered. Only when the queue structure contains data, the corresponding bit is set to 1. If it is empty, the corresponding bit is set to 0.

Based on the concept of the above register model and queue model, the IEEE 488.2 standard status model is constructed from two types of register models and a queue model.

(1) Standard event status register and standard event status enable register

This register has the register model structure mentioned above. It has eight bits corresponding to eight standard events listed below encountered by the device.

(a) power on
(b) user request
(c) command error
(d) execution error
(e) device dependent error
(f) query error
(g) bus control request
(h) operation complete.

The result of logical OR is output to the status byte register bit 5 (DIO 4) as an event status bit (ESB) summary message.

(2) Status byte (STB) register and service request enable (SRE) register

The status byte register consists of an RQS bit and seven summary message bits for setting status summary messages from the status data structure. It is used in combination with a service request enable register. When the result of ORing the values of these two registers is 0, the SRQ is set ON. In this case, the status byte register bit "DIO 7" is reserved by the system as an RSQ bit, so this bit indicates to an external controller that a service request exists. The function of the SRQ conforms to IEEE 488.1.

(3) Output queue

This queue has the queue model structure mentioned above. Its contents are summarized and transferred to the status byte register bit 4 (DIO 5) as a MAV (message available) summary message.

8.2 Status Byte Register

The status byte register consists of device STB and RQS (or MSS) messages. IEEE 488.1 defines the method of reporting STB and RQS messages, but it does not define the setting and clearing protocols and STB meaning. IEEE 488.2 defines device status summary messages and MSS transferred to bit 6 along with an STB in response to the *STB? common query.

8.2.1 ESB and MAV Summary Message

The followings are the explanations of an ESB summary message and an MAV summary message.

(1) ESB summary message

The ESB (event summary bit) summary message is defined by IEEE 488.2. It appears in status byte register bit 5. This bit indicates whether one or more IEEE 488.2 defined events have occurred, with the service request enable register set to allow events to occur, after the standard event status register was read or cleared last. The ESB summary message bit becomes True when at least one event registered in the standard event status register becomes True with event occurrence enabled. Conversely, the ESB summary bit becomes False when none of the registered events has occurred even if event occurrence is enabled.

(2) MAV summary message

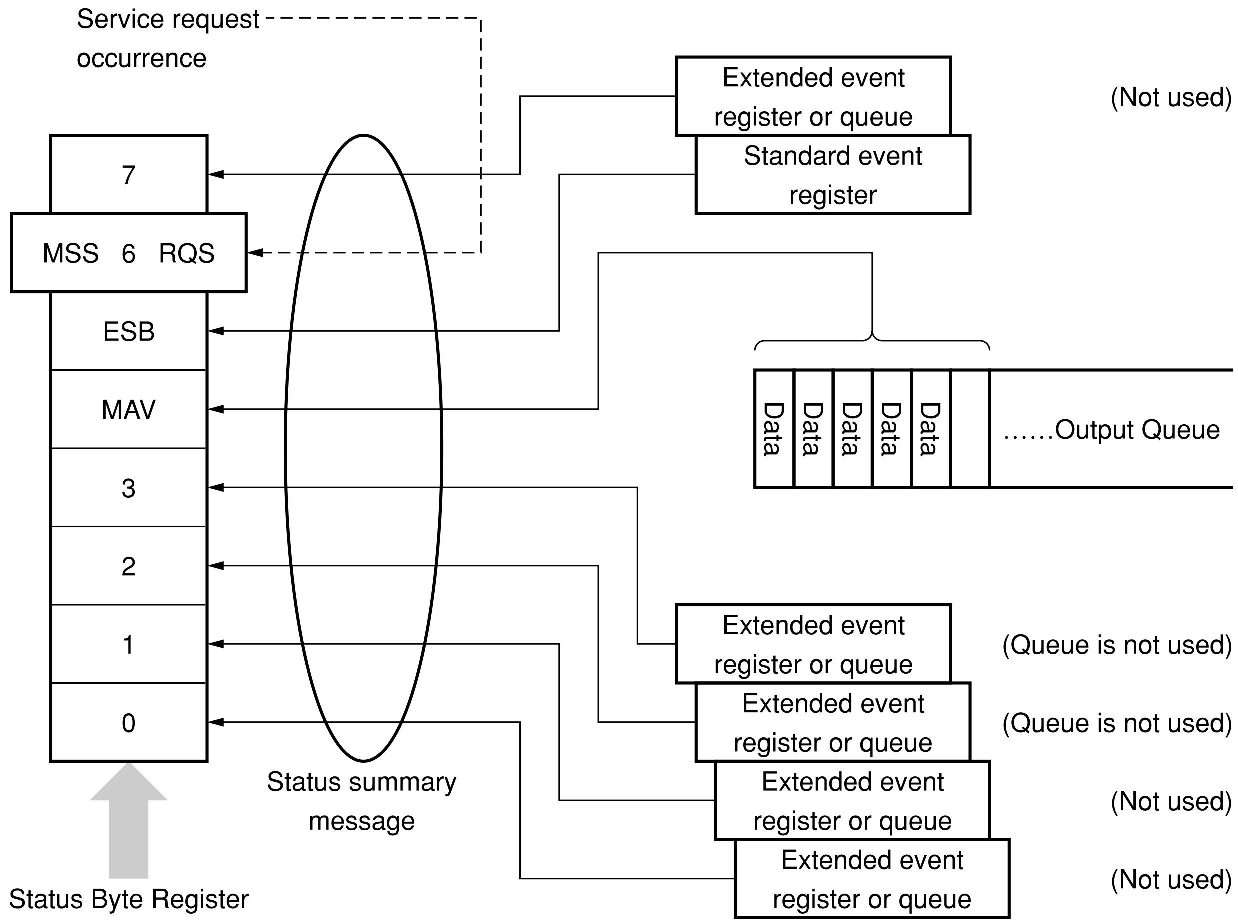
The MAV (message available) summary message is defined by IEEE 488.2. It appears in status byte register bit 4. This bit indicates whether the output queue is empty. When a device is ready for accepting response messages from the controller, the MAV summary message bit becomes 1 (True). When the output queue is empty, this bit becomes 0 (False). This message is used to synchronize information exchange with the controller. For example, the controller can send a query message to the device and wait for the MAV to become True. The controller can perform another processing while waiting for a response from the device. If the controller has started reading the output queue without checking the MAV, all system bus operations are suspended until a response is received from the device.

8.2.2 Device Dependent Summary Message

IEEE 488.2 does not define whether status register bit 7 (DIO 8) and bit 3 (DIO 4) to bit 0 (DIO 1) are used as status register summary bits or the bits indicating existence of data in the queue. Accordingly, these bits can be used as device dependent summary message bits.

Device dependent summary messages have a register model or queue model status data structure. This status register is a pair of registers used to report events and states in parallel or a queue used to report states and information sequentially. The summary bit provides a summary of the current status of the corresponding status data structure. For the register model, the summary message bit becomes True when one or more events have become True with occurrence of events enabled. For the queue model, the summary message bit becomes True when the queue is not empty.

In the MT9810A, bit 1 and bit 0 are unused, and bits 3 and 7 are used as summary bits of the status register; bit 2 is allocated to the queue as shown in the following diagram. Therefore, there are four types (two types for extension) of register models and there are two types (one type for extension) of queue models.



8.2.3 Reading and Clearing the Status Byte Register

Status byte register contents can be read using serial polling or an ∗STB? common inquiry. IEEE 488.1 defined STB messages can be read by either method, but the value transferred to bit 6 (position) varies depending on the method. status byte register contents can be cleared using a ∗CLS command.

(1) Reading the status byte register using serial polling (only when a GPIB interface bus is used)

When IEEE 488.1 defined serial polling is carried out, the device must return a 7 bit status byte and IEEE 488.1 defined RQS message bit. According to IEEE 488.1, the RQS message indicates whether the device has issued SRQs in the True state. The status byte value is not affected by serial polling. Immediately after being polled, the device must set the rsv message in the False state. If the device is polled again before a cause of issuing a new service request occurs, the RQS message has already been set in the False state.

(2) Reading the status byte register using an ∗STB? common query

The ∗STB? common query causes the device to output status byte register contents and one <NR1 NUMERIC RESPONSE DATA> from the MSS summary message. The response is the total of the status register value assigned binary weights and MSS summary message value. Status byte register bits 0 to 5 and 7 are assigned weighs 1, 2, 4, 8, 16, 32, and 128 respectively, and the MSS is assigned weights 64. The response to the ∗STB? is the same as that to serial polling with the exception that an MSS summary message appears in bit 6 instead of an RQS message.

(3) Definition of MSS (Master Summary Status)

The MSS indicates that the device has at least one cause of issuing a service request. In the device's response to the ∗STB? query, the MSS message appears in bit 6. However, it does not appear in the response to serial polling. It must not be regarded as part of the IEEE 488.1 defined status byte. The MSS is the result of ORing the values of status byte register and SRQ enable (SRE) register bits totally. Specifically, the MSS is defined as follows:

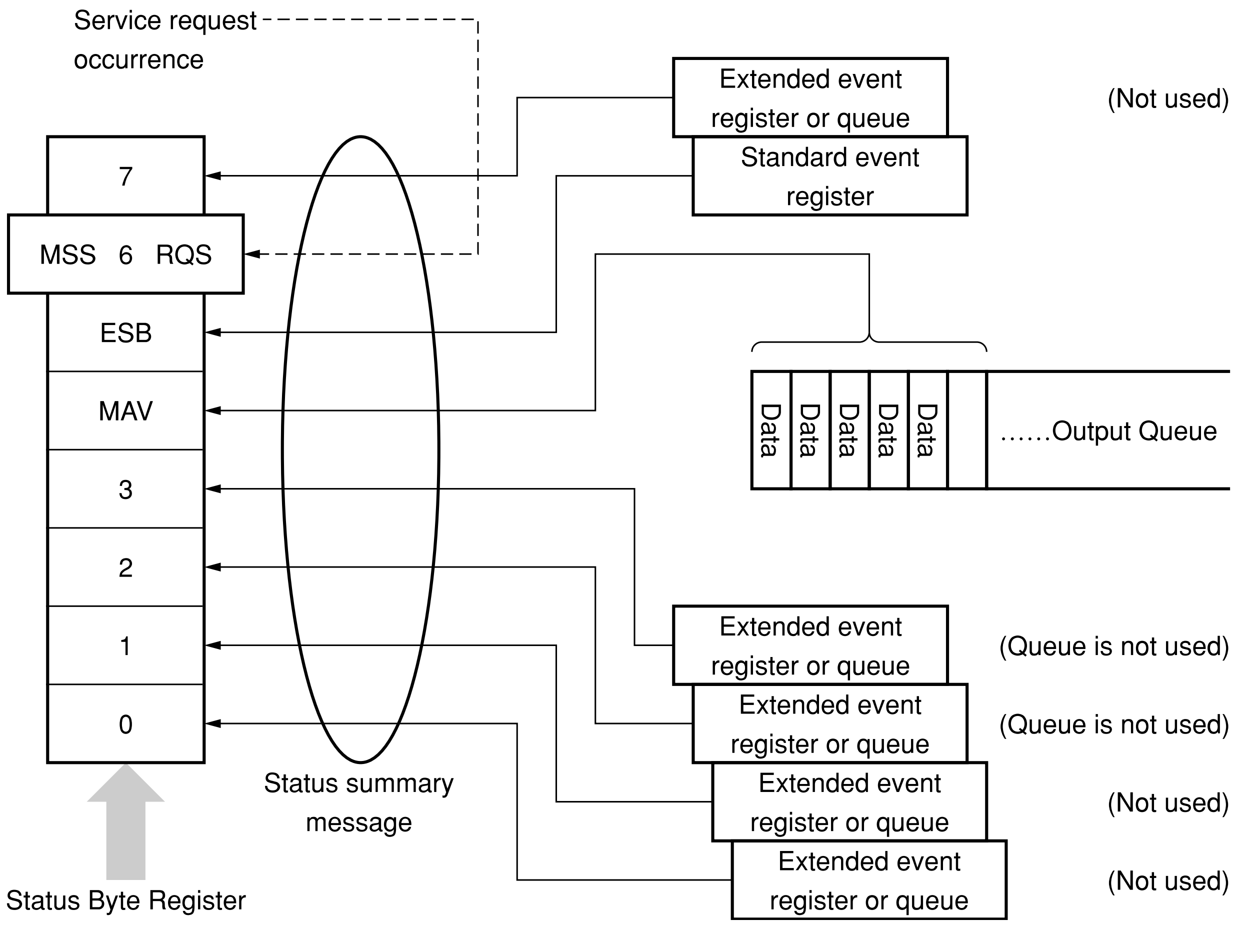
```
(STB Register bit 0 AND SRE Register bit 0)
                    OR
(STB Register bit 1 AND SRE Register bit 1)
                    OR
                     :
                     :
(STB Register bit 5 AND SRE Register bit 5)
                    OR
(STB Register bit 7 AND SRE Register bit 7)
```

In the definition of the MSS, the values of bits 6 of the status byte register and SRQ enable register are ignored. Accordingly, when calculating the MSS value, the status byte may be handled assuming that it is represented by 8 bits and bit 6 is always 0.

(4) Clearing the status byte register using a ∗CLS common command

The ∗CLS common command clears all status structures, except the output queue and MAV summary message (i.e., event registers and queues), and the corresponding summary messages.

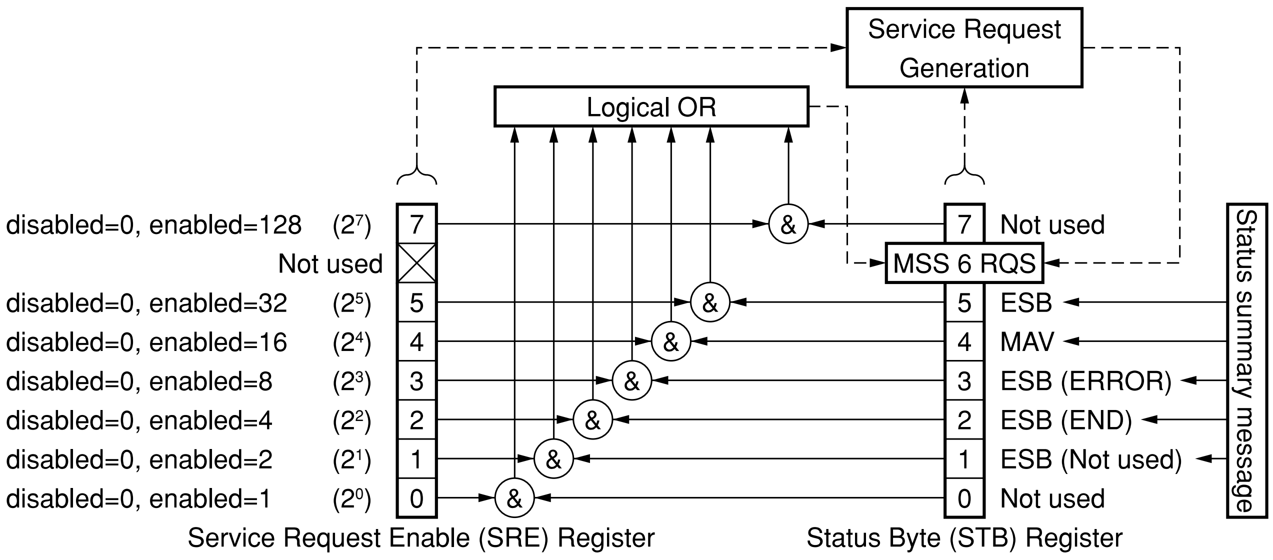
Issuing a ∗CLS command after the <PROGRAM MESSAGE TERMINATOR> element or before the <Query MESSAGE UNIT> element clears all status bytes. With this method, all unread messages in the output queue are cleared and the MAV message becomes False. When replying to the ∗STB?, the MSS message becomes False, too. Values of enable registers are not affected by ∗CLS.



8.3 Enabling the SRQ

Enabling the SRQ allows a summary message in the status byte register to be selected in response to a service request. The service request enable (SRE) register shown below can be used to select a summary message.

Bits of the service request enable register correspond to the bits of the status byte (STB) register. When 1 is set in a status byte bit corresponding to a significant bit of the service request enable register, the devices sets the RQS bit to 1 and issues a service request to the controller. For example, when bit 4 of the service request enable register is set (enabled) in advance, a service request can be issued to the controller each time the MAV bit is set to 1 (if the output queue has data).



Service Request Generation
Logical OR
disabled=0, enabled=128 (2^7)
Not used
disabled=0, enabled=32 (2^5)
disabled=0, enabled=16 (2^4)
disabled=0, enabled=8 (2^3)
disabled=0, enabled=4 (2^2)
disabled=0, enabled=2 (2^1)
disabled=0, enabled=1 (2^0)
7
5
4
3
2
1
0
&
MSS 6 RQS
Not used
ESB
MAV
ESB (ERROR)
ESB (END)
ESB (Not used)
Not used
Status summary message
Service Request Enable (SRE) Register
Status Byte (STB) Register

(1) Reading the service request enable register

service request enable register contents can be read using an ∗SRE? common inquiry. The response message to this query is <NR1 NUMERIC RESPONSE DATA>, an integer ranging from 0 to 255. It is a total of values of the service request enable register. Service request enable register bits 0 to 5 and 7 are assigned weights 1, 2, 4, 8, 16, 32, and 128, respectively. Unused bit 6 must always be 0.

(2) Updating the service request enable register

The service request enable register is written using an ∗SRE common command. The ∗SRE common instruction is followed by a <DECIMAL NUMERIC PROGRAM DATA> element. <DECIMAL NUMERIC PROGRAM DATA> is rounded to an integer. It is represented in binary notation using a base 2, indicating the total of values of service request enable register bits (weight value). When the value of this bit is 1, it indicates the enabled state. When the value of this bit is 0, it indicates the disabled state. The value of bit 6 must always be ignored.

(3) Clearing the service request enable register

The service request enable register can be cleared by executing an ∗SRE common command or turning on the power.

When an ∗SRE common command is used, the service request enable register can be cleared by bringing the <DECIMAL NUMERIC PROGRAM DATA> element value to 0. Clearing the service request enable register disables the status information to generate an rsv local message, suppressing issue of a service request.

When the power is turned on, the service request enable register is cleared if the Power-ON status clear flag is True and the ∗PSC command for disabling clearing of this register is not supported.

8.4 Standard Event Status Register

8.4.1 Definition of Standard Event Status Register Bits

Any device conforming to IEEE 488.2 must have the standard event status register. Operation of the standard event register model is shown below, and the meaning of standard event status register bits given in IEEE 488.2 is explained in the Table 8-1.

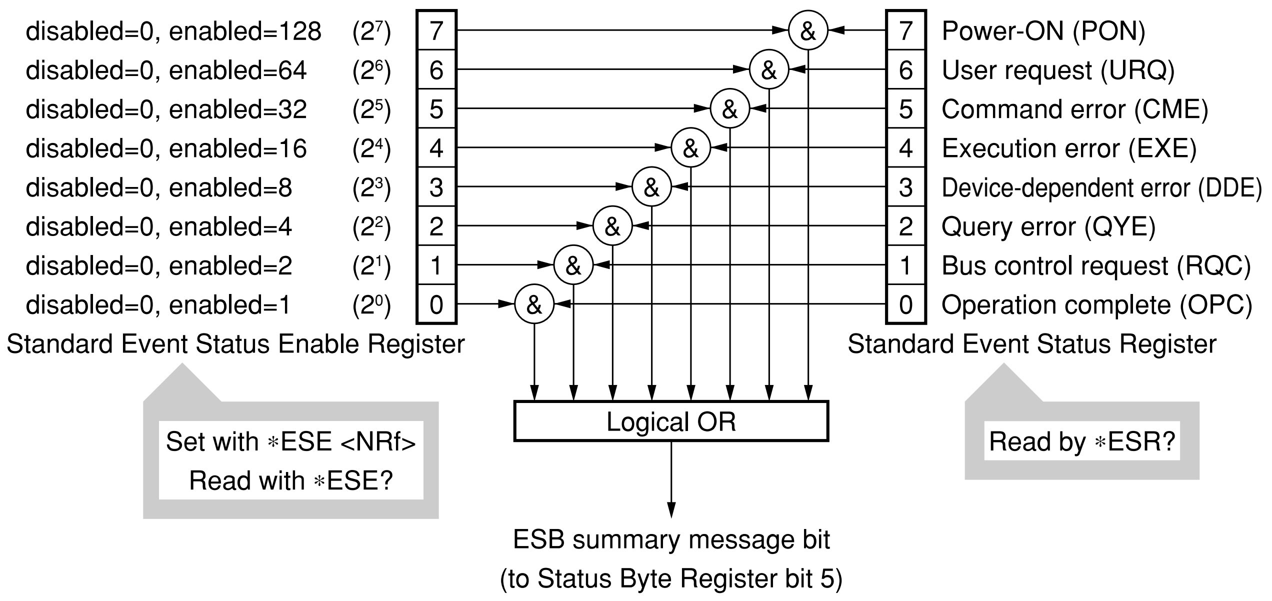

disabled=0, enabled=128 (2^7)
disabled=0, enabled=64 (2^6)
disabled=0, enabled=32 (2^5)
disabled=0, enabled=16 (2^4)
disabled=0, enabled=8 (2^3)
disabled=0, enabled=4 (2^2)
disabled=0, enabled=2 (2^1)
disabled=0, enabled=1 (2^0)
7 6 5 4 3 2 1 0
&
Standard Event Status Enable Register
Set with ∗ESE <NRf>
Read with ∗ESE?
Logical OR
ESB summary message bit
(to Status Byte Register bit 5)
Power-ON (PON)
User request (URQ)
Command error (CME)
Execution error (EXE)
Device-dependent error (DDE)
Query error (QYE)
Bus control request (RQC)
Operation complete (OPC)
Standard Event Status Register
Read by ∗ESR?

Table 8-1

Bit	Event name	Description
7	Power-ON (PON)	The power has been turned ON.
6	User request (URQ)	Local control is requested. This bit is set irrespective of the remote/local state of the device. Since this bit is not supported by MT9810A, it is always 0.
5	Command error (CME)	A program message including a syntax error or a misspelled command has been received or a GET command has been received in a program message.
4	Execution error (EXE)	A program message which is syntactically correct but cannot be executed has been received.
3	Device-dependent error (DDE)	An error other than CME, EXE, and QYE has occurred.
2	Query error (QYE)	An attempt was made to read data from the output queue while it has no data, or the data in the output queue has been lost due to overflow, etc.
1	Request control (RQC)	The device is required to be an active controller. Since this bit is not used by MT9810A, it is always 0.
0	Operation complete (OPC)	The device has completed the specified pending operation and ready for receiving a new instruction. This bit responds only to the ∗OPC command and sets the operation complete bit.

8.4.2 Details on Query Errors

Table 8-2

No.	Item	Description
1	Incomplete program message	When a device receives an MTA from the controller before receiving a program message terminator, it discards the incomplete message which has been received so far and waits for the next program message. To discard the incomplete program message, the device clears the input/output buffer, reports a query error to the status report part, and sets the standard status register bit 2 (query error bit).
2	Interruption of response message output	When a device receives an MLA from the controller before completing output of a response message terminator, it automatically interrupts output of the response message and waits for a next program message. To interrupt output of the response message, the device clears the input/output buffer, reports a query error to the status report part, and sets the standard status register bit 2 (query error bit).
3	When the next program message is sent without reading a response message	When the device cannot output a response message because the controller has output a program message (including a query message) and the next program message in succession, the device discards the response message and waits for the next program message. A query error is reported to the status report part like item No. 2.
4	Output queue overflow	When a program message containing many query messages is executed one after another, too many response messages to be stored in the output queue (256 bytes) may be generated. If more query messages are input and the response messages to queries must be output, the output queue overflows. When this happens, the device clears the output queue and resets the response message generation part. The device also sets the standard event status register bit 2 (query error bit) in the status report part.

8.4.3 Reading, Writing, and Clearing the Standard Event Status Register

Table 8-3

Read	This register is read destructively in response to the ∗ESR? common command. In other words, this register is cleared after being read. The event bit assigned binary weights and converted to a decimal value <NR1> is the response message.
Write	This register cannot be written externally; however, it can be cleared.
Clearing	This register is cleared in the following cases: (1) A ∗CLS command is received. (2) The power is turned on if the Power-ON status clear flag is True. The device executing a Power-ON sequence first clears the standard event status register, then records the events that have occurred in this sequence (e.g., PON event bit setting). (3) An event is read in response to an ∗ESR? query command.

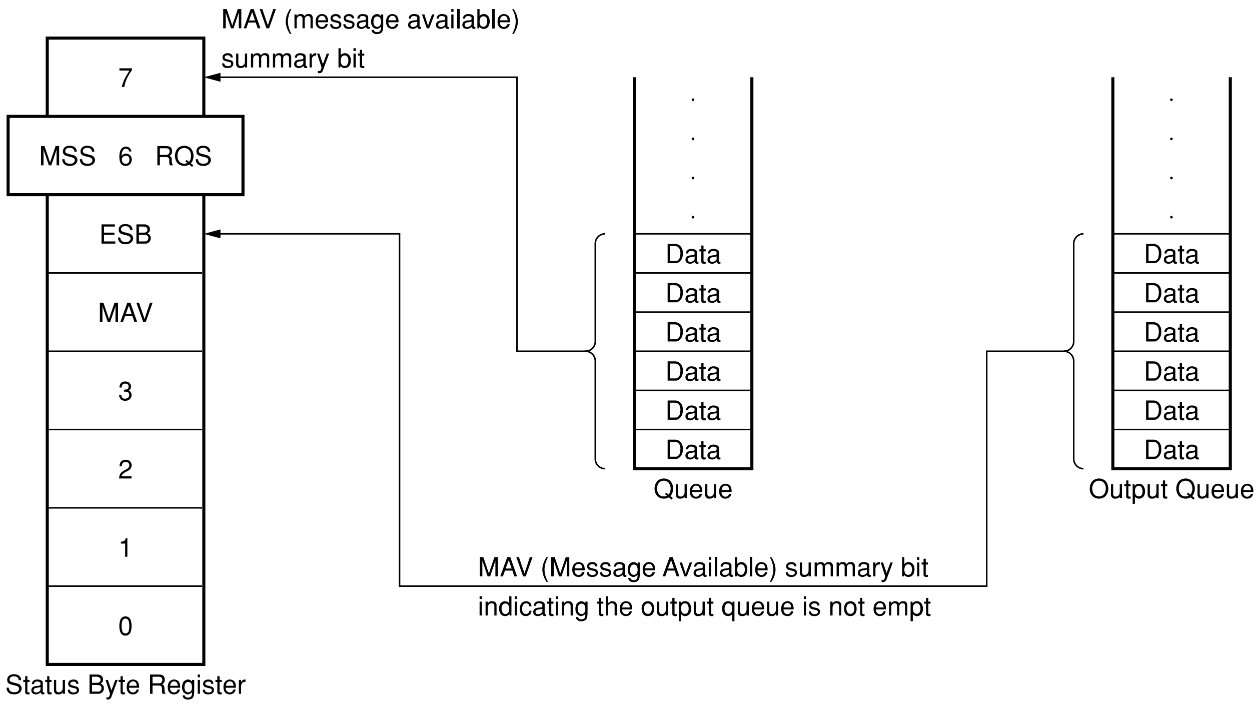
8.4.4 Reading, Writing, and Clearing the Standard Event Status Enable Register

Table 8-4

Read	This register is read non-destructively in response to the ∗ESR? common command. In other words, this register is not cleared after being read. The response message is assigned binary weights, converted from a binary value to a decimal value <NR1>, and returned.
Write	This register is written using an ∗ESS common command. Register bits 0 to 8 are assigned weights 1, 2, 4, 8, 16, 32, 64, and 128 respectively, so a total of values of the desired write data bits is sent as <DECIMAL NUMERIC PROGRAM DATA>.
Clearing	This register is cleared in the following cases: (1) An ∗ESE command with its data value being 0 is received. (2) The power is turned on with the Power-ON status clear flag in the True state or the power is turned on when a ∗PSC command is not supported. The standard event status register is not affected by the following: (1) Change in the state of the IEEE 488.1-defined device clear function (2) Reception of an ∗RST common command (3) Reception of a ∗CLS common command

8.5 Queue Model

The right-hand side of the figure below shows a queue model having a status data structure. A queue is a data structure in which data is arranged sequentially, providing information such as sequential status. A summary message indicates that such information exists in the queue. Queue contents are read by an handshake when the device is in TACS (talker active state).



The queue that outputs an MAV summary bit to status byte register bit 4 is called an "output queue." This queue is mandatory. The queue that can output an MAV summary message to one of status byte register bits 0 to 3 and 7 is simply called a "queue." It is optional. A summary message from the register model can also be output to status byte register bits 0 to 3 and 7, so the summary message type depends on the device type.

Refer to the Table 8-5 for a comparison of the output queue to general queues.

Table 8-5 Comparison of Output Queue to General Queues

Item	Output queue	Queue
Data input/output type	FIFO type	Not necessary to be FIFO type
Read	Response message units are read using only an IEEE 488.2 message exchange protocol. The type of these response message units depends on the query type.	Response message units are read with device-dependent query commands. These response message units must be of the same type.
Write	Program message elements are not written directly. This queue communicates with the system interface using only an IEEE 488.2 message exchange protocol.	Program message elements are not written directly. Coded device information is indicated.
Summary message	When the output queue is not empty, the summary message bit becomes True (1). When it is empty, the summary message bit becomes False (0). The MAV summary message is used to synchronize information exchange between a device and the controller.	When the queue is not empty, the summary message bit becomes True (1). When it is empty, the summary message bit becomes False (0).
Clearing	This queue is cleared in the following cases: (a) All items in the queue are read. (b) A DCL bus command is received for message exchange. (c) The PON bit becomes True at Power-ON. (d) Operation is unterminated or interrupted.	This queue is cleared in the following cases: (a) All items in the queue are read. (b) A *CLS command is received. (c) Other device-dependent means

8.6 Extended Status Bytes

In the SCPI standard, bit 7 in the status byte is used as "OPERation Status" and bit 3 is used as "QUEStionable Status". Bit 2 is allocated to "Error/Event Queue."
Each status register has the following configuration.

(1) CONDITION REGISTER

The condition register remains unchanged even after reading from the external device (controller). It cannot be set by any of the commands from the external device (controller) and can be set only by the state change in the measuring instrument.

(2) TRANSITION FILTER

The transition filter is used to determine whether to report the state change reported to the condition register to the event status register.
The filter for change from 0 to 1 is called the P-transition filter, while the filter for change from 1 to 0 is called the N-transition filter. These filters are rewritten as a mask pattern in accordance with the request from the external device (controller) (set/clear for each bit). These mask patterns remain unchanged even after the reading from the controller.

(3) EVENT REGISTER

The event register can be set indirectly through the condition register or the P/N-transition filter from the inside of the measuring instrument. The event resister cannot be directly accessed from an application program.

(4) EVENT ENABLE REGISTER

An event enable register for the event register.

(5) ERROR/EVENT QUEUE

While a message is stored in this queue, the corresponding bit in the status byte register is set. If a message goes out of the message queue, the corresponding bit in the status byte register is cleared.

8.6.1 Status register

STATus:PRESet

(1) Function

Initialization of the enable register and transition filter

(2) Program message

```
STATus:PRESet
```

(3) Explanation

This command initializes the enable register and transition filter. Each register is set as shown in the Table 8-6.

Table 8-6

Register	Filter/Enable	Preset Value
Operation	Enable	all 0
	PTR	all 1
	NTR	all 0
Questionable	Enable	all 0
	PTR	all 1
	NTR	all 0
ISUMmary	Enable	all 0
	PTR	all 1
	NTR	all 0
INSTrument	Enable	all 1
	PTR	all 1
	NTR	all 0
all others	Enable	all 1
	PTR	all 1
	NTR	all 0

<node>:CONDition

(1) Function

Checking of the condition register

(2) Program message

```
<node>:CONDition?
```

(3) Response message

```
<code>
```

(4) Parameter

<code>:= $\{n \mid 0 \leq n \leq 32767\}$

(5) Explanation

This command returns the sum total of the values of the condition register. The item of the condition register to be specified is determined with <node>.

<node>:ENABle

(1) Function

Setting of the event enable register

(2) Program message

```
<node>:ENABle <mask>
<node>:ENABle?
```

(3) Response message

```
<mask>
```

(4) Parameter

`<mask>`:= {Logical sum of $2^n|0 \le n \le 15$}

(5) Explanation

This command finds the sum total of the bit digit values when the bit to be enabled in the event enable register becomes the parameter. The bit digit value to be disabled is zero. The item of the event enable register to be specified is determined with <node>.

<node>[:EVENt]

(1) Function

Checking of the event register

(2) Program message

```
<node>[:EVENt]?
```

(3) Response message

```
<code>
```

(4) Parameter

`<code>`:= {Logical sum of $2^n|0 \le n \le 15$}

(5) Explanation

This command returns the sum total of the values of the event register. The item of the event register to be specified is determined with <node>.

<node>:NTRansition

(1) Function

Setting of the N-transition register

(2) Program message

```
<node>:NTRansition <mask>
<node>:NTRansition?
```

(3) Response message

```
<mask>
```

(4) Parameter

`<mask>`:= {Logical sum of $2^n|0 \leq n \leq 15$}

(5) Explanation

This command finds the sum total of the bit digit values when the bit to be enabled in the N-transition register becomes the parameter. The bit digit value to be disabled is zero. The item of the N-transition register to be specified is determined with <node>.

<node>:PTRansition

(1) Function

Setting of the P-transition register

(2) Program message

```
<node>:PTRansition <mask>
<node>:PTRansition?
```

(3) Response message

```
<mask>
```

(4) Parameter

`<mask>`:= {Logical sum of $2^n|0 \leq n \leq 15$}

(5) Explanation

This command finds the sum total of the bit digit values when the bit to be enabled in the P-transition register becomes the parameter. The bit digit value to be disabled is zero. The item of the P-transition register to be specified is determined with <node>.

8.6.2 Operation Status Register

The operation status register indicates the state of the equipment.
The commands are shown below. Insert these commands into the <node> portion in the status register.

Command	Description
STATus:OPERation	Operation status register
STATus:OPERation:SETTling	State of temperature of light source unit
STATus:OPERation:MEASuring	Measuring condition of optical sensor unit
STATus:OPERation:CORRection	State of zero-set operation of optical sensor unit
STATus:OPERation:AVERage	State of averaging operation of optical sensor unit

STATus:OPERation

(1) Function

Indication of the operation status register reference

(2) Explanation

This command makes the references of the operation status register.
The state of the equipment is indicated by allocating to bits. Each bit indicates the following.

Bit	Description
1	State of temperature of light source unit
4	Measuring condition of optical sensor unit
7	State of zero-set operation of optical sensor unit
8	State of averaging operation of optical sensor unit

STATus:OPERation:SETTling

(1) Function

Indication of the state of temperature of light source unit

(2) Explanation

This command indicates the state of the temperature of the light source unit and indicates whether it can be used. The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the light source unit can be used is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	The light source unit cannot be used
1	The light source unit can be used

STATus:OPERation:MEASuring

(1) Function

Indication of the measuring condition of optical sensor unit

(2) Explanation

This command indicates the measuring condition of the optical sensor unit.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the optical sensor unit is in measurement is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	The optical sensor unit is not measuring
1	The optical sensor unit is measuring

STATus:OPERation:CORRection

(1) Function

Indication of the state of zero-set operation of optical sensor unit

(2) Explanation

This command indicates the state of zero-set operation of the optical sensor unit.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the optical sensor unit is in zero-set is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	Zero-set is not being performed.
1	Zero-set is being performed.

STATus:OPERation:AVERage

(1) Function

Indication of the state of averaging operation of optical sensor unit

(2) Explanation

This command indicates the state of averaging operation of the optical sensor unit.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the optical sensor unit is in averaging operation is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	Averaging operation is not being performed.
1	Averaging operation is being performed.

8.6.3 QUESTIONABLE Status Register

The commands of the QUESTIONABLE status register are shown below. Insert these commands into the <node> portion in the status register.

Command	Description
STATus:QUEStionable: POWer	QUESTIONABLE status register
STATus:QUEStionable: POWer:OVERRange	Over range of optical sensor unit
STATus:QUEStionable: POWer:UNDerrange	Under range of optical sensor unit
STATus:QUEStionable: POWer:CURRent	Current abnormality
STATus:QUEStionable: POWer:ENVTemp	Temperature abnormality
STATus:QUEStionable: POWer:POWer	Power supply abnormality

STATus:QUEStionable:POWer

(1) Function

Indication of the QUESTIONABLE status register reference

(2) Explanation

This command makes the references of the QUESTIONABLE status register.
The state of the device is indicated by allocating to bits. Each bit indicates the following.

Bit	Description
0	Over range of optical sensor unit
1	Under range of optical sensor unit
2	Remote interlock
6	Current abnormality
7	Temperature abnormality
8	Power supply abnormality

STATus:QUEStionable:POWer:OVERRange

(1) Function

Indication of the over range of optical sensor unit

(2) Explanation

This command indicates the over range of the optical sensor unit.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the optical sensor unit is in over range is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	The optical sensor unit is not over range
1	The optical sensor unit is over range

STATus:QUEStionable:POWer:UNDerrange

(1) Function

Indication of the under range of optical sensor unit

(2) Explanation

This command indicates the under range of the optical sensor unit.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether the optical sensor unit is in under range is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	The optical sensor unit is not under range
1	The optical sensor unit is under range

STATus:QUEStionable:POWer:CURRent

(1) Function

Indication of the current abnormality

(2) Explanation

This command indicates the occurrence of current abnormality.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether current abnormality is occurring is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	Current abnormality is not occurring.
1	Current abnormality is occurring.

STATus:QUEStionable:POWer:ENVTemp

(1) Function

Indication of the temperature abnormality

(2) Explanation

This command indicates the occurrence of temperature abnormality.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether temperature abnormality is occurring is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	Temperature abnormality is not occurring.
1	Temperature abnormality is occurring.

STATus:QUEStionable:POWer:POWer

(1) Function

Indication of the power supply abnormality

(2) Explanation

This command indicates the occurrence of power supply abnormality.
The bits correspond one for one with the channels in order with bit 0 as Channel 1. Depending on the state, whether power supply abnormality is occurring is indicated.

Bit	Corresponding channel
0	Channel 1
1	Channel 2

State	Description
0	Power supply abnormality is not occurring.
1	Power supply abnormality is occurring.

Section 9 Details on Device Messages

9.1 Main Frame

9.1.1 DISPlay:BRIGhtness

(1) Function

Brightness setting

(2) Program message

```
DISPlay:BRIGhtness <ratio>
DISPlay:BRIGhtness?
```

(3) Response message

```
<ratio>
```

(4) Parameter

`<ratio>:=` $\{f \mid 0.1 \le f \le 1.0\}$

(5) Explanation

This command sets the brightness on the display.
When <ratio> is set to 0.1, the brightness is the lowest; when it is set to 1, the brightness is the highest.
The brightness can be set in ten steps.

```
 0.1  ← <ratio> →  1
Dark              Bright
```

9.1.2 DISPlay[:STATe]

(1) Function

Turns ON/OFF the display

(2) Program message

```
DISPlay[:STATe] <sw>
DISPlay[:STATe]?
```

(3) Response message

```
<status>
```

(4) Parameter

`<sw>:= {ON,OFF,1,0}`
`<status>:= {1,0}`
1 .... ON
0 .... OFF

(5) Explanation

This command switches the display/non-display of the display.

9.1.3 SYSTem:BEEPer:STATe

(1) Function

Buzzer setting

(2) Program message

```
SYSTem:BEEPer:STATe <level>
SYSTem:BEEPer:STATe?
```

(3) Response message

```
SYSTEM:BEEPER:STATE <level>
```

(4) Parameter

```
<level>:= {0,1,2,3,4}
```

(5) Explanation

This command sets the level of the buzzer sound.
The buzzer sound is set as shown below depending on <level>.

<level>		Meaning
0		Buzzer OFF
1	↑	Small Level
2	│	
3	│	
4	↓	Large Level

9.1.4 SYSTem:CHANnel:STATe

(1) Function

Inquires the inserted unit

(2) Program message

```
SYSTem:CHANnel:STATe?
```

(3) Response message

```
SYSTEM:CHANNEL:STATE <uid> (@ <uno>){,<uid> (@ <uno>)}
```

(4) Parameter

```
<uid>:= {OPM,OLS}
<uno>:= {1,2}
```

(5) Explanation

This command outputs the types and ID numbers for all units inserted currently. If no unit is inserted, "NOUNIT" is returned as response data.
The unit type is indicated with <uid> and it is interrupted as shown below.

<uid>	Unit name
OPM	Optical sensor unit
OLS	Light source unit

9.1.5 SYSTem:COMMunicate:GPIB:HEAD

(1) Function

Specifies whether to attach a header

(2) Program message

```
SYSTem:COMMunicate:GPIB:HEAD <flag>
SYSTem:COMMunicate:GPIB:HEAD?
```

(3) Response message

```
SYSTEM:COMMUNICATE:GPIB:HEAD <status>
```

(4) Parameter

```
<flag>:= {ON,OFF,1,0}
<status>:= {1,0}
```

1	ON
0	OFF

(5) Explanation

This command specifies whether to attach a header to the response message
By default, no header is attached.
The same setting item exists in both GPIB and serial (SYSTem:COMMunicate:GPIB:HEAD and SYSTem:COMMunicate:SERial:HEAD). These are not independent of each other. Therefore, if one item is set, the other is set to the same condition.

9.1.6 SYSTem:COMMunicate:SERial:HEAD

(1) Function

Specifies whether to attach a header

(2) Program message

```
SYSTem:COMMunicate:SERial:HEAD <flag>
SYSTem:COMMunicate:SERial:HEAD?
```

(3) Response message

```
SYSTEM:COMMUNICATE:SERIAL:HEAD <status>
```

(4) Parameter

`<flag>:= {ON,OFF,1,0}`
`<status>:= {1,0}`
`1` .... ON
`0` .... OFF

(5) Explanation

This command specifies whether to attach a header to the response message.
Default does not attach a header.
The same setting item exists in both GPIB and serial (SYSTem:COMMunicate:GPIB:HEAD and SYSTem:COMMunicate:SERial:HEAD). These items are not independent of each other. Therefore, if one item is set, the other is set to the same condition.

9.1.7 SYSTem:DATE

(1) Function

Sets the calendar

(2) Program message

```
SYSTem:DATE <year>,<month>,<day>
SYSTem:DATE?
```

(3) Response message

```
<year>,<month>,<day>
```

(4) Parameter

`<year>:=` $\{n|1990 \le n \le 2089\}$
`<month>:=` $\{n|1 \le n \le 12\}$
`<day>:=` $\{n|1 \le n \le 31\}$

(5) Explanation

This command sets the calendar of the system.
<year>, <month>, and <day> indicate year, month, and day, respectively.

9.1.8 SYSTem:ERRor

(1) Function

Inquires the error value

(2) Program message

```
SYSTem:ERRor?
```

(3) Response message

```
<code>
```

(4) Explanation

As a response to SYSTem:ERRor, SCPI specifies the codes and messages corresponding to the errors. The error messages supported by this product are described in the Section 9.4 "Error Message."

9.1.9 SYSTem:TIME

(1) Function

Sets the time

(2) Program message

```
SYSTem:TIME <hour>,<minute>,<second>
SYSTem:TIME?
```

(3) Response message

```
<hour>,<minute>,<second>
```

(4) Parameter

```
<hour>:= {n|0 ≤ n ≤ 23}
<minute>:= {n|0 ≤ n ≤ 59}
<second>:= {n|0 ≤ n ≤ 59}
```

(5) Explanation

This command sets the clock of the system to the specified time.
The time is specified in 24-hour unit. <hour>, <minute>, and <second> indicate hour, minute, and second, respectively.

9.2 Optical Sensor

[1|2] indicates the channel number into which the optical sensor to be controlled is inserted. If the optical sensor is inserted into Channel 1, it can be omitted. The brackets ([ ]) are not required.

Example: `ABORT1 FETCH2:SCALAR:POWER:DC  SENSE:CORRECTION:COLLECT:ZERO` etc.

9.2.1 ABORt[1|2]

(1) Function

Stops measurement

(2) Program message

```
ABORt[1|2]
```

(3) Explanation

This command stops the logging.

9.2.2 FETCh[1|2][:SCALar]:POWer[:DC]

(1) Function

Inquires the measurement data

(2) Program message

```
FETCh[1|2][:SCALar]:POWer[:DC]?
```

(3) Response message

```
<level>
```

(4) Parameter

```
<level>:= <NR3>
```

(5) Explanation

This command returns the current measurement data.
The unit of the measurement data may be dBm, W, or dB in accordance with the current unit.

9.2.3 SENSe[1|2]:AVERage:COUNt

(1) Function

Sets the number of times of averaging

(2) Program message

```
SENSe[1|2]:AVERage:COUNt <count>
SENSe[1|2]:AVERage:COUNt?
```

(3) Response message

```
SENSE1|2:AVERAGE:COUNT <count>
```

(4) Parameter

`<count>:= {1,2,5,10,20,50,100,200,500,1000}`

(5) Explanation

This command sets the number of times of averaging in the averaging operation.

9.2.4 SENSe[1|2]:BANDwidth

(1) Function

Sets the bandwidth

(2) Program message

```
SENSe[1|2]:BANDwidth <bw> [<unit>]
SENSe[1|2]:BANDwidth?
```

(3) Response message

```
SENSE1|2:BANDWIDTH <bw>
```

(4) Parameter

`<bw>:= {0.1,1,10,100,1000,10000,100000}` (Unit: Hz)
`<unit>:= {HZ,KHZ}`

(5) Explanation

This command sets the bandwidth to the value set in <bw>.
In the program message, supplementary units may be used.
In the response message, the value is always output in Hz.

9.2.5 SENSe[1|2]:BANDwidth:AUTO

(1) Function

Sets the auto bandwidth

(2) Program message

```
SENSe[1|2]:BANDwidth:AUTO <sw>
SENSe[1|2]:BANDwidth:AUTO?
```

(3) Response message

```
SENSE1|2:BANDWIDTH:AUTO <status>
```

(4) Parameter

```
<sw>:= {ON,OFF,1,0}
<status>:= {1,0}
```

1	ON
0	OFF

(5) Explanation

This command sets the bandwidth setting to auto.

9.2.6 SENSe[1|2]:CORRection:COLLect:ZERO

(1) Function

Executes zero-set

(2) Program message

```
SENSe[1|2]:CORRection:COLLect:ZERO
SENSe[1|2]:CORRection:COLLect:ZERO?
```

(3) Response message

```
<result>
```

(4) Parameter

```
<result>:= <NR1>
```

(5) Explanation

This command executes zero-set.
The response message has the following value.
For the error code, refer to the Section 9.4 "Error Message."

<result>	State
0	Normal end
1	Zero-set from remote is not executed.
2	Zero-set is being executed.
Negative number	Error

9.2.7 SENSe[1|2]:CORRection[:LOSS[:INPut[:MAGNitude]]]

(1) Function

Sets the calibration factor

(2) Program message

```
SENSe[1|2]:CORRection[:LOSS[:INPut[:MAGNitude]]] <cal>[DB]
SENSe[1|2]:CORRection[:LOSS[:INPut[:MAGNitude]]]?
```

(3) Response message

```
<cal>
```

(4) Parameter

```
<cal>:= {f| -199.99 ≤ n ≤ 199.99}
```

(5) Explanation

This command sets the calibration factor to <cal>.
<cal> is always accepted in dB. The unit may be omitted.

9.2.8 SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MAXimum

(1) Function

Reads the maximum value

(2) Program message

```
SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MAXimum?
```

(3) Response message

```
SENSE1|2:FETCH:SCALAR:POWER:DC:MAXIMUM <level>
```

(4) Parameter

```
<level>:= <NR3>
```

(5) Explanation

This command utputs the maximum value of the measured data during the period from the start of statistical measurement up to now.
The unit of the measurement data is dBm or W in accordance with the present measurement value.

9.2.9 SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MINimum

(1) Function

Reads the minimum value

(2) Program message

```
SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:MINimum?
```

(3) Response message

```
SENSE1|2:FETCH:SCALAR:POWER:DC:MINIMUM <level>
```

(4) Parameter

```
<level>:= <NR3>
```

(5) Explanation

This command outputs the minimum value of the measurement data during the period from the start of statistical measurement up to now.
The unit of the measured data is dBm or W in accordance with the present measured value.

9.2.10 SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:PTPeak

(1) Function

Reads the difference between the maximum and minimum values

(2) Program message

```
SENSe[1|2]:FETCh[:SCALar]:POWer[:DC]:PTPeak?
```

(3) Response message

```
SENSE1|2:FETCH:SCALAR:POWER:DC:PTPEAK <level>
```

(4) Parameter

```
<level>:= <NR3>
```

(5) Explanation

This command outputs the difference between the maximum and minimum values of the measurement data during the period from the start of statistical measurement up to now.
The unit of the measurement data is dB.

9.2.11 SENSe[1|2]:FILTer:BPASs:FREQuency

(1) Function

Sets the modulation frequency

(2) Program message

```
SENSe[1|2]:FILTer:BPASs:FREQuency CW|<freq>[<unit>]
SENSe[1|2]:FILTer:BPASs:FREQuency?
```

(3) Response message

```
SENSE1|2:FILTER:BPASS:FREQUENCY <freq>
```

(4) Parameter

`<freq>:= {0,270,1000,2000}` (Unit: Hz)
`<unit>:= {HZ,KHZ}`

(5) Explanation

This command sets the modulation frequency to be measured.
If the unit of <freq> is omitted, Hz is assumed. If the unit is specified in <unit>, set in the unit.
For 0 Hz, CW is set.
The response message is always output in Hz.

9.2.12 SENSe[1|2]:INITiate[:IMMediate]

(1) Function

Starts the logging

(2) Program message

```
SENSe[1|2]:INITiate[:IMMediate]
```

(3) Explanation

This command makes the measurements by the number of times specified.
The number of times is set in "SENSe:TRIGger:COUNt."

9.2.13 SENSe[1|2]:MEMory:COPY[:NAME]

(1) Function

Stores/Reads the measurement conditions

(2) Program message

```
SENSe[1|2]:MEMory:COPY[:NAME] MC,<no>|<no>,MC
```

(3) Parameter

```
<no>:= {0,1,2,3,4,5,6,7,8,9}
```

(4) Explanation

This command stores or reads the measurement conditions using the memory number specified with <no>.
"MC, <no>" stores the measurement condition and "<no>, MC" reads the measurement condition.
If "0" is specified for <no>, only reading is effective because it is the initial condition setting.

9.2.14 SENSe[1|2]:MEMory:DATa

(1) Function

Reads the logging data

(2) Program message

```
SENSe[1|2]:MEMory:DATa? MD[,<start>[,<number>]]
```

(3) Response message

```
SENSE1|2:MEMORY:DATA <number>{,<level>}*
```

(4) Explanation

This command reads the logging data.
<start> and <number> indicate the starting point of reading and the number of data items to be read, respectively.
If these are omitted, operation is started with <start> as 1 and <number> as the number of data items measured.
For <start>, a value larger than the number of data items measured cannot be specified. For <number>, the number of data items measured or a number larger than the number of data items measured after <start> may be specified. However, the number of data items that can be taken out is the effective number of data items.
For the response message, the number of data items actually taken out and the measurement data of these data items are output. Though the measurement data does not have a unit, it is the unit used when it is recorded (dBm or W).
This unit can be known from "SENSe:MEMory:DATa:INFO." If the unit is W, the exponential notation is used instead of the supplementary unit.
If no measurement data exists, the response message only with <number> as "0" is output.

9.2.15 SENSe[1|2]:MEMory:DATa:INFO

(1) Function

Logging data information

(2) Program message

```
SENSe[1|2]:MEMory:DATa:INFO?
```

(3) Response message

```
SENSE1|2:MEMORY:DATA:INFO V1.0,"<info>"
```

(4) Explanation

This command reads the detailed information of the logging data.
"V1.0" at the head of the response message is used to identify the succeeding information. At present, only "V1.0" used.
In <info>, the following information are listed and separated by a comma (,),

Information	Description
unit model name	the model name of the unit measured
measurement date/time	the date and time of measurement are described as shown below: `YY/MM/DD, hh:mm:ss`
averaging count	the number of times of averaging made at the time of measurement
interval time	measurement interval used at the time of measurement
measurement data count	the number of data items measured
data unit	"DBM" or "W"
statistical data	a list of the maximum value, minimum value, peak-to-peak value, and average value separated by a comma (,).

Thought the units of these values are omitted, thses are output in the values in accordance with the units of the data. If the data unit is "DBM," the maximum value, minimum value, and average value are output in dBm and the peak-to-peak value is output in W. If the data unit is "W," the maximum value, minimum value, and average value are output in W and the peak-to-peak value is output in %.
When the data unit is "W," the watt value described in the exponential notation instead of the supplementary unit.
If no measurement data exists, the response message is output with <info> as a blank.

```
SENSE1:MEMORY:DATA:INFO V1.0,""
```

9.2.16 SENSe[1|2]:POWer:INTerval

(1) Function

Sets the measurement interval

(2) Program message

```
SENSe[1|2]:POWer:INTerval <time>
SENSe[1|2]:POWer:INTerval?
```

(3) Response message

```
SENSE1|2:POWER:INTERVAL <time>
```

(4) Parameter

`<time>:= {f|0.001 ≤ f ≤ 359999}`

(5) Explanation

This command sets the measurement interval.
The parameter <time> is set in seconds and rounded off to the resolution.

9.2.17 SENSe[1|2]:POWer:RANGe:AUTO

(1) Function

Sets the auto range

(2) Program message

```
SENSe[1|2]:POWer:RANGe:AUTO <sw>
SENSe[1|2]:POWer:RANGe:AUTO?
```

(3) Response message

```
<status>
```

(4) Parameter

`<sw>:= {ON,OFF,1,0}`
`<status>:= 1,0`
1 .... ON
0 .... OFF

(5) Explanation

This command specifies whether to set the measurement range to auto.
If "ON" or "1" is set, automatic setting is valid.
If "OFF" or "0" is set, automatic setting is invalid.

9.2.18 SENSe[1|2]:POWer:RANGe[:UPPer]

(1) Function

Sets the manual range.

(2) Program message

```
SENSe[1|2]:POWer:RANGe[:UPPer] <levle>[DBM]
SENSe[1|2]:POWer:RANGe[:UPPer]?
```

(3) Response message

```
<level>
```

(4) Parameter

```
<level>:= {30,20,10,0,-10,-20,-30,-40,-50,-60,-70,-80,-90,-100,-110}
```

(5) Explanation

This command makes measurement with the measurement range fixed to <level>.
Since <level> is dependent on the unit, some values may not be set in some units.
For the unit, only "DBM" is accepted. The unit can be omitted.

9.2.19 SENSe[1|2]:POWer:REFerence

(1) Function

Sets the reference value.

(2) Program message

```
SENSe[1|2]:POWer:REFerence <type>,<level>[<unit>]
SENSe[1|2]:POWer:REFerence? <type>
```

(3) Response message

```
<level>
```

(4) Parameter

<type>:= {TOA,TOB,TOREF,0,1,2}
When "TOREF" or "2":
<level>:= {f(W)|$1 \times 10^{-16} \le f \le 99.999$}
<level>:= {f(dBm)| $-199.999 \le f \le +199.999$}
<unit>:= {PW,NW,UW,MW,W,DBM}
When "TOA" or "0," and "TOB" or "1"
<level>:= {f(dB)| $-199.999 \le f \le 199.999$}
<unit>:= {DB}

(5) Explanation

This command sets the reference value at the reference measurement time.
"TOREF" sets the specified level value as the reference value of the channel. It can be used in both SENSe1 and SENSe2.
The reference value can be set in W or dBm. If the unit is omitted, dBm is assumed. If the unit is specified as "W," the values from 0.0001 pW to 99.999 W can be set; if the unit is specified as "DBM," the values from –199.999 dBm to +199.999 dBm can be set.
"TOA" and "TOB" are effective only when two optical sensor units are inserted, and the reference value for the difference in levels between two channels is set. In this case difference in level – standard value is displayed. (Where, standard value = reference value + relative value)
The reference value can be set only in dB. Therefore, when adding a unit, only "DB" is effective. "TOA" and "TOB" are effective only for "SENSe2" and "SENSe1," respectively.
When "TOA" or "TOB" is specified, the response message is always returned in dB.
When "TOREF" is specified, the response message is returned in W or dBm in accordance with the current unit.

9.2.20 SENSe[1|2]:POWer:REFerence:DISPlay

(1) Function

Displays the relative value.

(2) Program message

```
SENSe[1|2]:POWer:REFerence:DISPlay
```

(3) Explanation

Relative values that make the display value set to 0 dB is set and values are displayed in relative value display.
Since this command displays relative measurements with the current display value as 0 dB, it can be set even in the absolute value display and reference value display. The display value can be obtained by the following expression.
display value = measurement value – reference value – relative value
If the absolute value display is changed to the relative value display, the reference value is treated as 0.

9.2.21 SENSe[1|2]:POWer:REFerence:STATe

(1) Function

Turns ON/OFF the reference measurement

(2) Program message

```
SENSe[1|2]:POWer:REFerence:STATe <sw>
SENSe[1|2]:POWer:REFerence:STATe?
```

(3) Response message

```
<status>
```

(4) Parameter

`<sw>:= {ON,OFF,1,0}`
`<status>:= {1,0}`

1	ON
0	OFF

(5) Explanation

The command sets ON/OFF of the reference measurement.

9.2.22 SENSe[1|2]:POWer:REFerence:STATe:RATio

(1) Function

Reference selection

(2) Program message

```
SENSe[1|2]:POWer:REFerence:STATe:RATio <sel>
SENSe[1|2]:POWer:REFerence:STATe:RATio?
```

(3) Response message

```
<status>
```

(4) Parameter

`<sel>:= {TOA,TOB,TOREF,0,1,2}`
`<status>:= {0,1,2}`

0	TOA
1	TOB
2	TOREF

(5) Explanation

This command sets the method of the reference measurement.
With <sel>, specify the reference measurement method as shown below.

<sel>	Measurement method
TOA (0)	(measurement value of Channel 2) – (measurement value of Channel 1)
TOB (1)	(measurement value of Channel 1) – (measurement value of Channel 2)
TOREF (2)	(measurement value of specified channel) – (standard value)

Where, the standard value is (reference value) + (relative value).
"TOA" and "TOB" are effective only for "SENSe2" and "SENSe1," respectively.
"TOREF" is effective for both SENSe1 and SENSe2.

9.2.23 SENSe[1|2]:POWer:UNIT

(1) Function

Switches the unit system

(2) Program message

```
SENSe[1|2]:POWer:UNIT <unit>
SENSe[1|2]:POWer:UNIT?
```

(3) Response message

```
<unit>
```

(4) Parameter

```
<unit>:= {DBM,W}
```

(5) Explanation

This command switches the display unit system of the optical power measurement.

9.2.24 SENSe[1|2]:POWer:WAVelength

(1) Function

Specifies the wavelength

(2) Program message

```
SENSe[1|2]:POWer:WAVelength <wavelength>[<unit>]
SENSe[1|2]:POWer:WAVelength?
```

(3) Response message

```
<wavelength>
```

(4) Parameter

<wavelength>:= $\{f(m) | 380 \times 10^{-9} \le f \le 1800 \times 10^{-9}\}$
<wavelength>:= $\{f(Hz) | 166.551 \times 10^{12} \le f \le 788.927 \times 10^{12}\}$
<unit>:= {NM,UM,M,HZ}

(5) Explanation

This command sets the wavelength compensation to the wavelength of <wavelength>.
The setting range and resolution of the wavelength are dependent on the optical sensor unit.
If the unit is omitted in the program message, m is assumed.
The response message is output in accordance with current unit system (m or HZ).

9.2.25 SENSe[1|2]:POWer:WAVelength:UNIT

(1) Function

The display unit of the wavelength

(2) Program message

```
SENSe[1|2]:POWer:WAVelength:UNIT <unit>
SENSe[1|2]:POWer:WAVelength:UNIT?
```

(3) Response message

```
<unit>
```

(4) Parameter

<unit>:= {M,HZ}

(5) Explanation

This command switches the display unit of the wavelength.

9.2.26 SENSe[1|2]:TRIGger:COUNt

(1) Function

Sets the number of times of measurement

(2) Program message

```
SENSe[1|2]:TRIGger:COUNt <count>
```

(3) Response message

```
SENSE1|2:TRIGGER:COUNT <count>
```

(4) Parameter

```
<count>:= {n|1 ≤ n ≤ 1000}
```

(5) Explanation

Sets the number of logging data items.

9.2.27 SENSe[1|2]:TRIGger[:SEQuence][:IMMediate]

(1) Function

Re-start the statistical measurement.

(2) Program message

```
SENSe[1|2]:TRIGger[:SEQuence][:IMMediate]
```

(3) Explanation

This command measures the minimum, maximum, and peak-to-peak values of the measurement data.

9.2.28 READ[1|2]

(1) Function

Starts the high-speed transfer mode

(2) Program message

```
READ[1|2]?
```

(3) Response message

```
<level>
```

(4) Parameter

```
<level>:= <NR3>
```

(5) Explanation

This command returns the current measurement data.
The transfer rate used this command is higher than one used the FETCh[1|2][:SCALar]:POWer[:DC] command.
The data is absolute one, and the unit of data is dBm.
The high-speed transfer mode stops when the READ[1|2]:ABORt command (refer to 9.2.29) is used.

9.2.29 READ[1|2]:ABORt

(1) Function

Stops the high-speed transfer mode

(2) Program message

```
READ[1|2]:ABORt
```

(3) Explanation

This command stop the high-speed transfer mode.

9.3 Light Source

[1|2] indicates the channel number into which the light source to be controlled is inserted. If the optical sensor is inserted into Channel 1, it can be omitted. The brackets ([ ]) are not required.

Example: `SOURCE1:POWER:STATE ON` `SOURCE2:POWER:STATE?` `SOURCE:POWER:STATE 0` etc.

9.3.1 SOURce[1|2]:AM[:INTerval]:FREQuency

(1) Function

Sets the modulation frequency

(2) Program message

```
SOURce[1|2]:AM[:INTerval]:FREQuency CW|<freq>[<unit>]
SOURce[1|2]:AM[:INTerval]:FREQuency?
```

(3) Response message

```
<freq>
```

(4) Parameter

`<freq>:= {0,270,1000,2000}` (Unit: Hz)
`<unit>:= {HZ,KHZ}`

(5) Explanation

This command sets the optical output to CW or the modulation frequency specified in <freq>.
If the unit of <freq> is omitted, Hz is assumed. If the unit is specified in <unit>, it is set in the specified unit. For 0 Hz, CW is set.
The response message is always output in Hz.

9.3.2 SOURce[1|2]:MEMory:COPY[:NAME]

(1) Function

Stores/reads the measurement conditions

(2) Program message

```
SOURce[1|2]:MEMory:COPY[:NAME] MC,<no> | <no>,MC
```

(3) Parameter

`<no>:= {0,1,2,3,4,5,6,7,8,9}`

(4) Explanation

This command stores or reads the measurement conditions using the memory number specified with <no>.
"MC, <no>" stores the measurement condition and "<no>, MC" reads the measurement condition.
If "0" is specified for <no>, only reading is effective because it is the initial condition setting.

9.3.3 SOURce[1|2]:POWer:ATTenuation

(1) Function

Sets the attenuation

(2) Program message

```
SOURce[1|2]:POWer:ATTenuation <level>[DB]
SOURce[1|2]:POWer:ATTenuation?
```

(3) Response message

```
<level>
```

(4) Parameter

`<level>:= {f(dB)|0.00 ≤ f ≤ 6.00}`

(5) Explanation

This command reduces the optical output from the maximum output level by the value specified in <level>.
The setting range and the setting resolution of <level> are dependent on the light source unit.
<level> is rounded off to the setting resolution.
The attenuation is always output in dB. The unit may be omitted.

9.3.4 SOURce[1|2]:POWer:STATe

(1) Function

Sets the optical output

(2) Program message

```
SOURce[1|2]:POWer:STATe <sw>
SOURce[1|2]:POWer:STATe?
```

(3) Response message

```
<status>
```

(4) Parameter

`<sw>:= {ON,OFF,1,0}`
`<status>:= {1,0}`

1	ON
0	OFF

(5) Explanation

This command sets ON/OFF of the optical output.

9.3.5 SOURce[1|2]:POWer:WAVelength

(1) Function:

Sets the wavelength

(2) Program message

```
SOURce[1|2]:POWer:WAVelength UPPer|LOWer|CENTer|<wavelength>[<unit>]
SOURce[1|2]:POWer:WAVelength?
```

(3) Response message

```
<wavelength>
```

(4) Parameter

<wavelength>:= $\{f(m) | 380 \times 10^{-9} \leq f \leq 1800 \times 10^{-9}\}$
<wavelength>:= $\{f(HZ) | 166.551 \times 10^{12} \leq f \leq 788.927 \times 10^{12}\}$
<unit>:= {NM,UM,M,HZ}

(5) Explanation

This command sets the wavelength to <wavelength>.
The range and the resolution of the wavelength are dependent on the light source unit. The actual setting is rounded off to the resolution.
If the unit is omitted in the program message, m is assumed.
If a unit is attached, set in the unit.
The response message is output in accordance with current unit system (m or HZ).
"UPPer" or "LOWer" can be specified as a parameter only for a two-wavelength light source. To "UPPer" and "LOWer," the wavelengths of the long wave and short wave are set, respectively.
Even if the wavelength setting is "UPPer" or "LOWer," the response message returns the wavelengths of the long wave and short wave.
"CENTer" can be specified as a parameter only for a DFB-LD light source. To "CENTer", the center wavelength (default condition) is set.

9.3.6 SOURce[1|2]:POWer:WAVelength:UNIT

(1) Function

The display unit of the wavelength

(2) Program message

```
SOURce[1|2]:POWer:WAVelength:UNIT <unit>
SOURce[1|2]:POWer:WAVelength:UNIT?
```

(3) Response message

```
<unit>
```

(4) Parameter

```
<unit>:= {M,HZ}
```

(5) Explanation

This command switches the display unit of the wavelength.

9.4 Error Messages

(1) Command errors [-100 to -199]

The error codes [-100 to -199] indicate the occurrence of syntax errors in IEEE 488.2. At this time, bit 5 in the event status register is set.
These errors are issued if any of the following events occur.

(a) The device received a message against the IEEE 488.2 standard.
(b) The device received a header that does not conform to the regulation of the device specific commands or the common commands.
(c) GET (Group Execute Trigger) was sent to a program message.

Code	Message	Error detecting condition
-101	Invalid character	Invalid characters are included in the header or parameter.
-104	Data type error	The parameter type is different from that of the specified type.
-105	Get not allowed	GET (Group Execute Trigger) was sent to a program message.
-108	Parameter not allowed	The number of parameters is larger than the specified number.
-112	Program mnemonic too long	The program mnemonic consists of more than 12 characters.
-113	Undefined header	Though the syntax of the header is correct, it is not defined in the device.
-120	Numeric data error	There is an error in the numeric data.
-121	Invalid character in number	An invalid character is included in the numeric data.
-130	Suffix error	There is an error in the suffix.
-144	Character data too long	The character data consists of more than 12 characters.

(2) Execution time error [-200 to -299]

The error codes [-200 to -299] indicate the occurrence of errors in the execution control unit of the device. If an error of this type occurs, bit 4 in the event status register is set.
These errors are issued if any of the following events occur.

(a) <PROGRAM DATA> following the header is out of the regulation of the device.
(b) The program message cannot be executed due to the state of the device.

Code	Message	Error detecting condition
-220	Parameter error	There is an error in the parameter.
-221	Setting conflict	Though the parameter is correct, it cannot be executed due to the state of the device.
-222	Data out of range	The numeric data is out of the regulation of the device.
-224	Illegal parameter value	The received parameters is illegal.
-240	Hardware error	The command cannot be executed due to the hardware failure.

(3) Device specific error [-300 to -399]

The error codes [-300 to -399] indicate the occurrence of errors other than command, query, and execution errors. These errors include the failure of hardware/firmware and self-diagnosis errors.
If an error of this type occurs, bit 3 in the event status register is set.

Code	Message	Error detecting condition
-310	System error	An error occurred in the system.
-315	Configuration memory error	Resume memory is lost.
-350	Queue overflow	There was an abnormality in self-diagnosis.

(4) Query error [-400 to -499]

The error codes [-400 to -499] indicate the occurrence of errors concerning the message exchange control protocol in the output queue control. If an error of this type occurs, bit 2 in the event status register is set.
These errors are issued if any of the following events occur.

(a) Reading is executed from the output queue when there is no output.
(b) The data in the output queue is lost.

Code	Message	Error detecting condition
-410	Query interrupted	Before the device completes the transmission of the response message, an interrupt by a new command occurred.
-420	Query unterminated	No query corresponding to the response message to be read is sent.
-430	Query deadlocked	An attempt is made to buffer the data exceeding the free area in the storage.

Section 10 Program Example

This section describes the creation of the remote control program.
This chapter shows an example of program created using Visual BASIC. For GPIB, the use of National Instrument's hardware and the NI-488.2M software is assumed. For the handling of Visual BASIC and NI-488.2M, see the individual operation manuals.

10.1 Precaution on Programming

On the creating a remote control program, precaution the points in the Table 10-1.

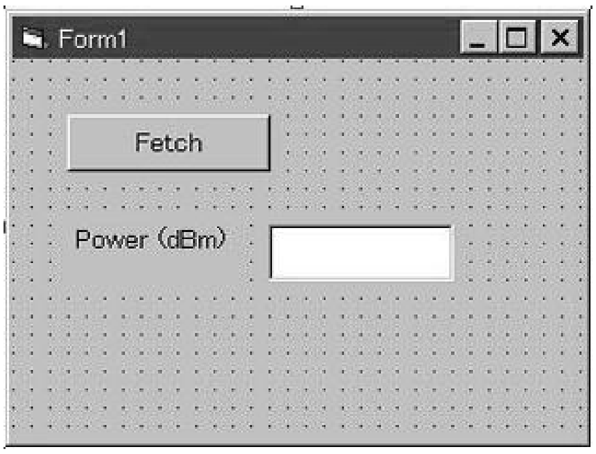
Table 10-1

No.	Precaution	Description
1	Be sure to initialize device.	Devices may be in various states after the device has been operated by its own operating panels and other programs. In many cases, its states may not be proper at the start of use. Therefore, these devices must be initialized to be able to use under certain conditions.
2	Immediately after sending a query, do not send any command other than result reading.	If MLA is received when a command other than a reading command is sent to the controller before reading the query result, the output buffer is cleared, resulting in the loss of the response message. Therefore, be sure to describe the result reading command immediately after reading.
3	Avoid exception handling in the protocol.	Expected exceptions must be handled in the exception handling section in the program so that execution does not stop due to errors.
4	Check interface functions (subset) of individual devices (GPIB).	If a created program is executed for a device that does not have a subset, processing will not proceed. Be sure to check subsets of devices. Also check that the device conforms to IEEE 488.2.
5	Prevent buffer overflow (RS-232C).	The RS-232C interface has a 256 byte data area as an internal receive buffer. However, overflow may occur depending on the processing type. To prevent errors form occurring due to overflow, do not send a large volume data (control commands) at a time when performing remote control using an RS-232C interface. After sending a sequence of commands, send the "OPC?" command, wait for a response to be received, then send the next command for synchronization.

10.2 Program Examples

(1) Reading the measurement data of the optical sensor unit.

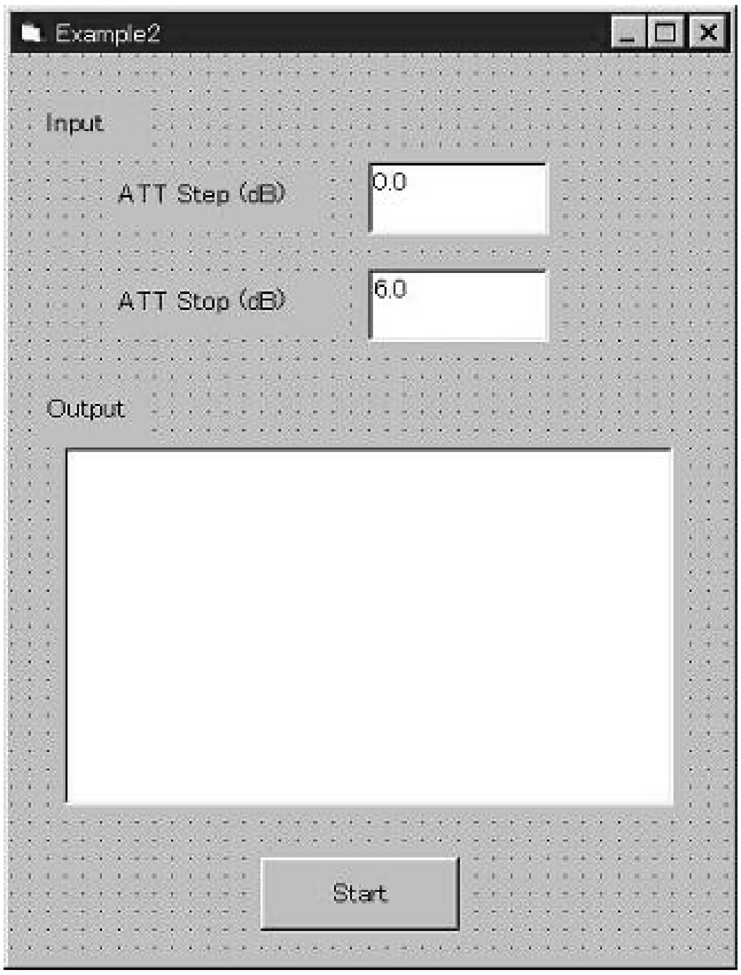
Insert a optical sensor unit into Channel 1 of the MT9810A to measure the optical power of the external light source.
Read the measurement data with GPIB and display the result.
The GPIB address of the MT9810A is 15.



```
Sub cmdfetch_Click()                                                    *1
    Dim buf1 As String*20                                               *2
    Call Send(0,15,"SYSTEM:COMMUNICATE:GPIB:HEAD 0",NLend)              *3
    Call Send(0,15,"SENSE1:POWER:UNIT DBM",NLend)                       *4
    Call Send(0,15,"FETCH1:SCALAR:POWER:DC?",NLend)                     *5
    Call Receive(0,15,buf1,STOPend)                                     *6
    lblPwr.Caption=buf1                                                 *7
End Sub                                                                 *8
```

*3-*4	OTS initial setting
*5-*6	Data reading
*7	Result output

(2) Measure the attenuator value of the light source unit using a optical sensor.
Insert the light source unit and the optical sensor unit into Channel 1 and Channel 2, respectively, of the MT9810A and connect these units using optical fibers. Measure the relative value of attenuation while changing the attenuator value of the light source unit one after another using the optical sensor unit and display the result.
The GPIB address of the MT9810A is 15.



```
Sub cmdstart_Click()                                                   *1
    Dim buf1 As String*15                                              *2
    Dim strAttStep As String*5                                         *3
    Dim strAttStop As String*5                                         *4
    Dim strAtt As String*5                                             *5
    Dim sglAttStep As Single                                           *6
    Dim sglAttStop As Single                                           *7
    Dim sglAtt As Single                                               *8
'
    chrAttStep=txtStep.Text                                            *9
    chrAttStop=txtStop.Text                                           *10
    sglAttStep=val(chrAttStep)                                        *11
    sglAttStop=val(chrAttStop)                                        *12
'
```

```
    Call Send(0,15,"SOURCE1:POWER:STATE 1",NLend)                         *13
    Call Send(0,15,"SOURCE1:POWER:ATTENUATION 0",NLend)                   *14
    Call Send(0,15,"FETCH2:SCALAR:POWER:DC?",NLend)                       *15
    Call Receive(0,15,buf1,STOPend)                                       *16
    lblResult.Caption="ATT=0.0 dB   P0="+buf1                             *17
'
    Call Send(0,15,"SENSE2:POWER:REFERENCE:DISPLAY",NLend)                *18
    sglAtt=sglAttStep                                                     *19
    Do                                                                    *20
        chrAtt=str(sglAtt)                                                *21
        Call Send(0,15,"SOURCE1:POWER:ATTENUATION"+chrAtt,NLend)          *22
        Call Send(0,15,"FETCH2:SCALAR:POWER:DC?",NLend)                   *23
        Call Receive(0,15,buf1,STOPend)                                   *24
        lblResult.Caption=lblResult.Caption+chr(13)                       *25
        lblResult.Caption=lblResult.Caption+"ATT="+chrAtt+"dB  Pr="+buf1  *26
        sglAtt=sglAtt + sglAttStep                                        *27
        If sglAtt > sglAttStop Then                                       *28
             Exit do                                                      *29
        End If                                                            *30
    Loop                                                                  *31
End Sub                                                                   *32
```

*13	Turns ON the optical output of the light source unit.
*14	Sets the attenuation of the light source unit to 0 dB.
*15-*16	Measures the power using the optical sensor unit.
*17	Displays the measurement result.
*18	Sets the optical sensor unit to the relative value measurement mode.
*22	Sets the attenuation of the light source unit.
*23-*24	Measures the power using the optical sensor unit.
*25-*26	Displays the measurement result.
*28-*30	Judges the repetition condition.

NOTE:

In the actual measurement, insert a waiting time of around five seconds between *14 and *15 and between *22 and *23 in order to stabilize the output of the light source unit.

Section 11 LabVIEW Drivers

This section explains the measuring instrument drivers (MX981001A) used to control the MT9810A remotely under LabVIEW.

LabVIEW drivers are modules in which command send and receive functions are incorporated, allowing measuring instruments to be controlled under the U.S. National Instruments Graphic Programming System "LabVIEW." Using these drivers, the MT9810A can be remotely controled without remembering control commands.
To use this drivers, a controller in which National Instruments LabVIEW software (Windows version) is installed is required.

The drivers have been created using LabVIEW Ver. 4.0 (Windows version).
Refer to the LabVIEW User's Guide for how to use LabVIEW.

LabVIEW is a trademark of U.S. National Instruments Corporation.
Windows is a trademark of U.S. Microsoft Corporation.

About LabVIEW

LabVIEW is a graphical program language suitable for controlling measuring instruments and saving and analyzing data.

LabVIEW to creates a program like drawing a circuit diagram, so it is easier to get used to compared with text-based program languages. The execution speed is almost the same as the C language.

LabVIEW supports various libraries related to measuring instrument control and data saving, analysis, and display. Using LabVIEW and measuring instrument drivers, the graphical user interface (GUI) program can be created easily.

11.1 Installation

The following file is stored in the attached floppy disk MX981001A.

```
MT9810A.LLB
```

Installation example

(1) On X:LABVIEW ("X" is the drive name on which LabVIEW is installed), create a directory "MT9810A.LIB."

(2) Copy the file MT9810A.LLB to this directory.

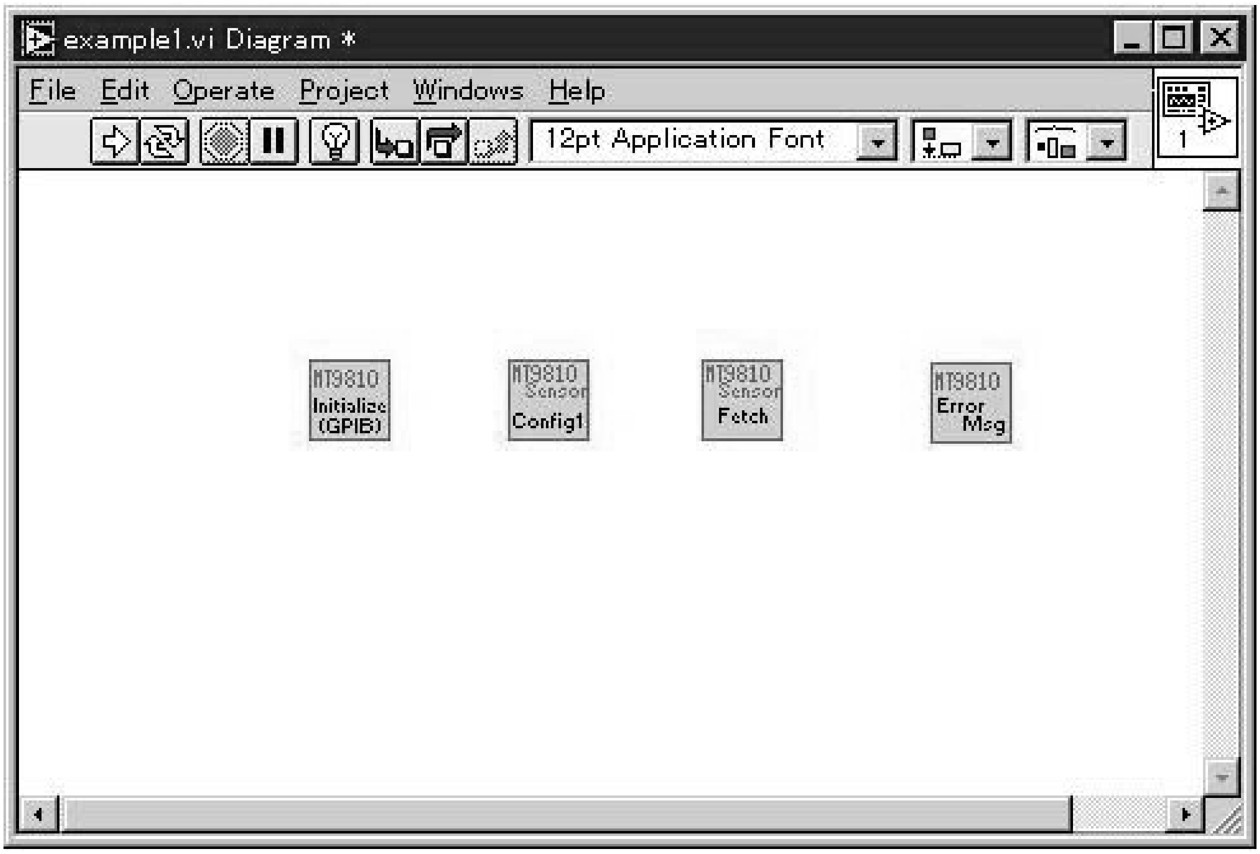
11.2 Program Example

This section gives examples of programs created using the LabVIEW driver.
This section creates a program of optical power measurement using GPIB control in the same manner as the Section "10.2 Program example 1." In this program example, the GPIB address of the MT9810A is 15.

This section uses the following four drivers.

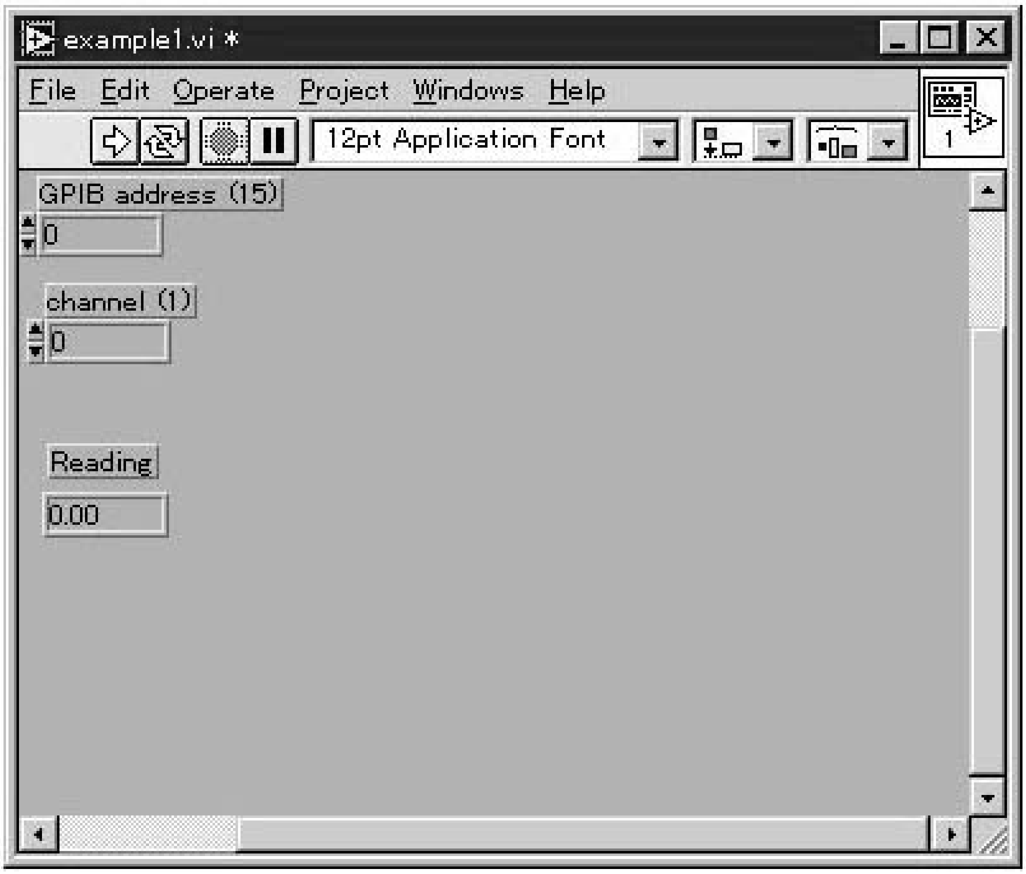
MT9810 Initialize(GPIB).vi	Preparation for communication using GPIB
MT9810 Config.Sensor.vi	Setting of the optical sensor unit
MT9810 Sensor.Fetch.vi	Reading of the measurement data from the optical sensor unit
MT9810 Error.message.vi	Displaying of the error message

(1) Arranging the drivers in the block diagram
Arrange the above drivers in order.



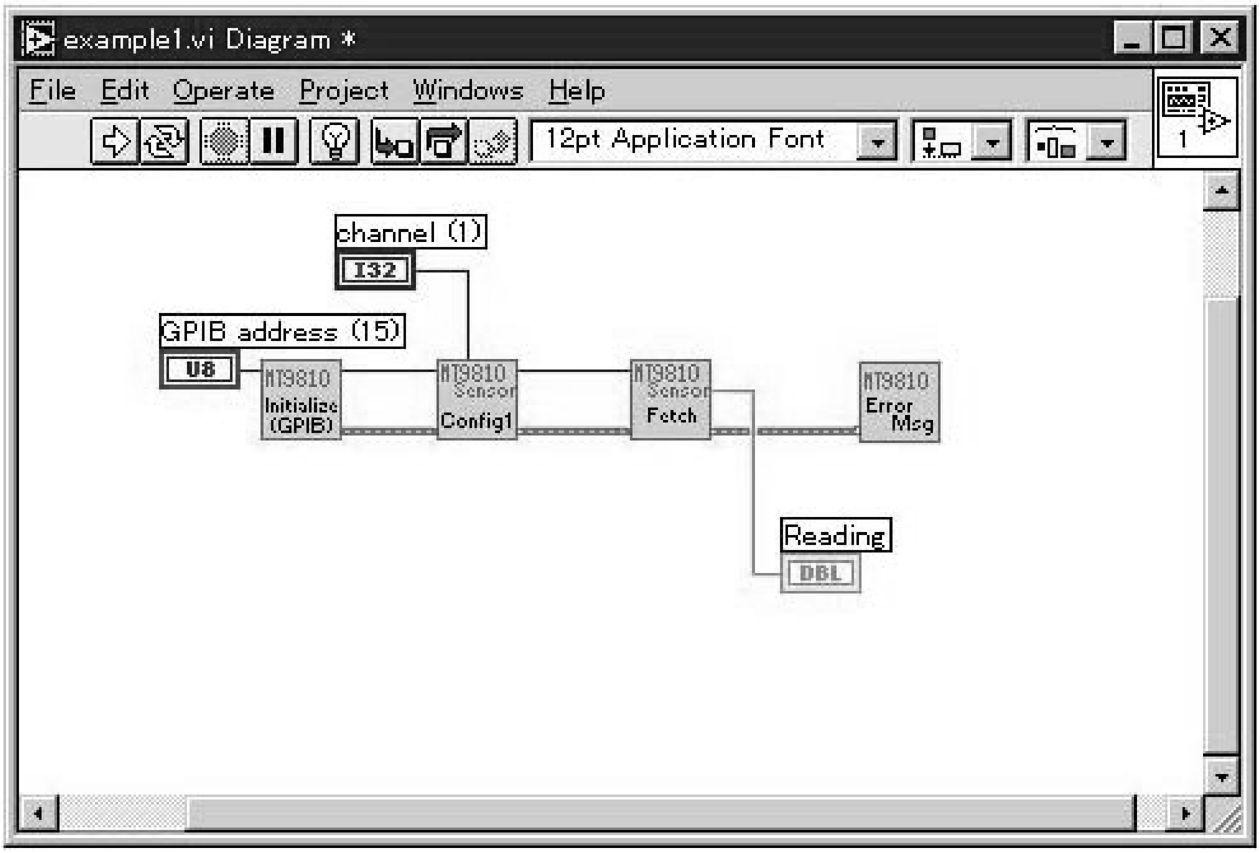
(2) Arranging controllers and displays on the front panel window.
Double-clicking on the icon of MT9810 Initialize(GPIB).vi on the diagram window will open the LabVIEW driver window. Copy the controllers subject to GPIB address input from this window onto the front panel window.

In the same manner, copy the displays for displaying measurement data from the icon of MT9810 Sensor.Fetch.vi.



GPIB address (15)
0
channel (1)
0
Reading
0.00

(3) Connecting displays, controllers, and icons.
Connect driver terminals with wires as shown below.



11.3 List of LabVIEW Drivers

The file name of the LabVIEW driver VI is MT9810 (function name).vi.
The common drivers are used for GPIB and RS-232C, excluding (Initialize).

Table 11-1 Sample/utility

File name	Function
MT9810 VI tree.vi	Loading all drivers
MT9810 Example1.vi	Simple program example
MT9810 Example2.vi	Simple program example
MT9810 Example3.vi	Simple program example
MT9810 Interactive.vi	Communication in device message level
MT9810 Error Message.vi	Error code and detailed information

Table 11-2 Main frame

File name	Function
MT9810 Initialize(GPIB).vi	GPIB preparation
MT9810 Initialize(RS232C).vi	RS-232C preparation
MT9810 Reset.vi	Main frame resetting
MT9810 Self-Test.vi	Internal self-test
MT9810 Config Instrument.vi	Main frame parameter setting/query

Table 11-3 Optical sensor unit

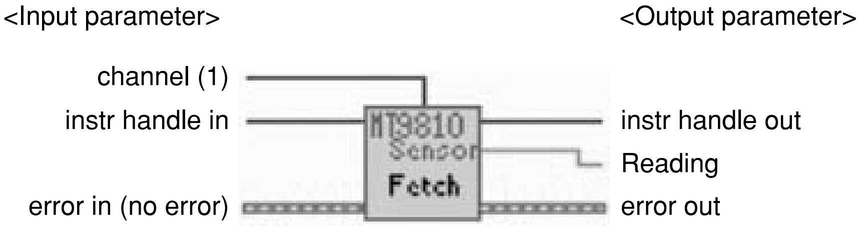
File name	Function
MT9810 Config Sensor Zeroing.vi	Zero-set
MT9810 Config Sensor_1.vi	Parameter setting/query
MT9810 Config Sensor_2.vi	Parameter setting/query
MT9810 Config Sensor Wavelength.vi	Measurement wavelength setting/query
MT9810 Config Sensor Ranging.vi	Measurement range setting/query
MT9810 Config Sensor Reference.vi	Reference measurement setting/query and execution/stop
MT9810 Sensor Fetch.vi	Measurement data query
MT9810 Config Logging Parameter.vi	Logging execution/stop
MT9810 Read Logging Values.vi	Outputting logging data
MT9810 MinMax Values.vi	Resetting or output of measurement data of maximum/minimum values

Table 11-4 Light source unit

File name	Function
MT9810 Config Source.vi	Parameter setting/query
MT9810 Config Source Output.vi	ON/OFF of optical output

11.4 Description of LabVIEW Driver Functions

This section explains functions and input/output parameters of LabVIEW drivers.
The LabVIEW driver receives data and setting values through the terminals on the left of the icon, performs the specified processing according to the input parameters, and outputs the processing results through the terminals on the right side of the icon.



MT9810 Sensor Fetch.vi

In the explanation of parameters in this chapter, the words in the brackets ([ ]) following the variable name indicate variable types.

11.4.1 Common Parameters

This section explains the input/output parameters used in most of the LabVIEW drivers.

instr handle in [I32]

A designator of MT9810 Gloval (global variable, one-dimensional array of cluster) that stores the GPIB address, serial port number, and communication parameter setting. Initialize.vi does not contain this parameter.

instr handle out [I32]

Outputs the value of Instr handle in. Close.vi does not contain this parameter.

error in [clust]

Outputs the error occurrence state before executing VI.

status [bool]	Indicates presence/absence of error. "True" indicates the occurrence of an error.
code [I32]	Indicates the error code at the time of error occurrence (when the status is set to True).
source [str]	Indicates VI in which the error occurred.

error out [clust]

Outputs the error occurrence status after executing VI. The contents of the cluster are the same as those of error in.

Channel [I32]

Indicates the unit channel number (VI for unit only).

11.4.2 Description of functions

(1) Sample/utility VI

MT9810 VI tree.vi

MT9810
VI Tree

All LabVIEW drivers are loaded on VI diagram. (Note that SubVI is not included) It can be used as a list.

MT9810 Example1.vi

MT9810
EXAMPLE
1

A simple program example using the LabVIEW driver.
Takes in the display value of the optical sensor unit in the interval set in Measurement Interval and displays it in the Reading display and in the chart. It sets and executes the optical sensor channel and the GPIB/RS-232 controller. (The GPIB, RS-232C parameter is placed at the right end of the window in a hidden manner) The chart is cleared by pressing the clear button. To end the program, press the Exit button.

MT9810 Example2.vi

MT9810
EXAMPLE
2

A simple program example using the LabVIEW driver.
Displays the state of reference measurement. (Two optical sensor units are required) It sets and executes the GPIB/RS-232 controller. (The GPIB, RS-232C parameter is placed at the left end of the window in a hidden manner) If the fetch button is pressed after changing the optical sensor channel, Absolute Unit, Reference State, and Level Value arbitrarily, the measurement values and the reference measurement values in Channels 1 and 2 are displayed. To end the program, press the Exit button.

MT9810 Example3.vi

MT9810
EXAMPLE
3

A simple program example using the LabVIEW driver.
Displays the maximum and minimum measurement values. It sets and executes the optical sensor channel and the GPIB/RS-232 controller. (The GPIB, RS-232C parameter is placed at the right end of the window in a hidden manner) The maximum value, minimum value, the difference between the maximum and minimum values, and elapsed time are displayed. The maximum and minimum values are reset by pressing the Reset button. To end the program, press the Exit button.

MT9810 Interactive.vi

MT9810
↕
LabVIEW

This driver makes communication with the MT9810A in the device message level. Set the GPIB/RS-232C and enter a device message of transmitting to the MT9810A in either Write Buffer 1, 2, 3, or 4. Specify the Write Buffer number of the device message to be actually sent with the switch and execute it. If a query command is sent, a response message is displayed in the Read Buffer.

MT9810 Error Message.vi

MT9810
Error
Msg

This driver reports the error code and its detailed information. After executing some LabVIEW driver VIs, execute this driver to check the error information

Parameter explanation

- type of dialog [int] .... Select the style of the dialog to be displayed when an error occurs.
  0: The dialog is not displayed.
  1: OK button dialog
  2: Continuance and stop button dialog
- status [bool] .... True if an error occurs.
- code [int] .... The corresponding error code is output.
  0 indicates that there is no error
  A negative code indicates that an error occurred.
  A positive code indicate a warning.
- error message [str] .... Outputs the explanation of the detected error.

(2) Main frame related VI

MT9810 Initialize(GPIB).vi

MT9810
Initialize
(GPIB)

This driver makes preparation for starting communication with the measuring instrument using GPIB.

Actual preparations are as follow:
1. Send device clear.
2. Check the ID of the main frame. (there are choices)
3. Execute reset (level 3). (there are choices)
4. Set the header of the response message to OFF.

Parameter explanation

• GPIB address [V8]	GPIB address
• Reset [bool]	Switching of reset operation.
• ID Query [bool]	Switching of ID check

MT9810 Initialize(RS232C).vi

MT9810
Initialize
(RS232)

This driver makes preparation for starting communication with the measuring instrument using RS-232C.

Actual preparations are as follow:
1. Set the serial port parameters.
2. Check the ID of the main frame. (selectable)
3. Executes reset (level 3). (selectable)
4. Set the header of the response message to OFF.

Parameter explanation

• RS-232C Parameter[clust]	Serial port setting value
Port No. (0:COM1) [V8]	Serial port number
baud rate (bps) [V16]	Baud rate
stop bit [V16]	Stop bit
parity bit [V16]	Parity bit
character (bit) [V16]	Character length
• Reset [bool]	Switching of reset operation.
• ID Query [bool]	Switching of ID check

MT9810 Reset.vi

MT9810
Reset

This driver resets the main frame.

Parameter explanation

(None)

MT9810 Self-Test.vi

MT9810
Self-Test

This driver makes internal self-test and returns the presence/absence of an error. The test result is not output to the "error out" cluster.

Parameter explanation

- Self-test Error [bool] .... True if the test result is error.

MT9810 Config Instrment.vi

MT9810
config
instr

This driver sets/inquires the parameters (display ON/OFF, brightness, date, time, buzzer level) of the main frame.

Parameter explanation

- Display Brightness [I32] .... Display brightness setting
- Set Date [clust] .... Date setting. Enter all values of Year, Month, and Day.
  Year [I32]
  Month [I32]
  Day [I32]
- Set Time [clust] .... Time setting. Enter all values of Hours, Minutes, and Seconds.
  Hours [I32]
  Minutes [I32]
  Seconds [I32]
- Beep Level [I32] .... Buzzer sound level setting

(3) Optical sensor unit related VI

MT9810 Config Sensor Zeroing.vi

MT9810
Sensor
Zero

This driver executes zero-set and outputs either normal end or error. After zero-set operation is ended or after an error occurs, it ends VI. The error is output to the "error out" cluster.

Parameter explanation

(None)

MT9810 Config Sensor_1.vi

MT9810
Sensor
Config1

This driver sets/inquires the parameters (display unit system, calibration factor, and modulation frequency).

Parameter explanation

• Absolute Units [I32]	Switching of display unit system.
• Calibration Factor [double]	Calibration factor
• Modulation Frequency [I32]	Modulation frequency setting value

MT9810 Config Sensor_2.vi

MT9810
Sensor
Config2

This driver sets/inquires the parameters (measurement interval, bandwidth, number of times of averaging).

Parameter explanation

• Measurement Interval [double]	Measurement interval setting value. It is rounded off to the resolution.
• Bandwidth [double]	Bandwidth setting value
• Averaging Time [I32]	Setting value of number of times of averaging

MT9810 Config Sensor Wavelength.vi

MT9810
Sensor
Wave-
length

This driver sets/inquires the parameters (measurement wavelength).

Parameter explanation

• Wavelength Value [double]	Wavelength setting value. The setting range and resolution of the wavelength are dependent on the optical sensor unit. The unit specified in Wavelength Unit is used.
• Wavelength Units [bool]	Switching of the unit of wavelength

MT9810
Sensor
Range

MT9810 Config Sensor Ranging.vi

This driver sets/inquires the measurement range.

Parameter explanation

- Power Range [I32] .... Power range setting value. Depending on the optical sensor unit, some values cannot be set.

MT9810
Sensor
Refer-
ence

MT9810 Config Sensor Reference.vi

This driver sets/inquires the reference measurements (reference measurement method, reference value) and executes/ stops the reference measurement.

Parameter explanation

- Reference State [I32] .... Switching of the reference measurement method. "Reference to the other" is (measurement value) – (measurement value of other channel) – (reference value)."Reference to Value" is (measurement value) – (reference value).
- Level Value [double] .... Reference value. The setting range differs depending on the reference measurement method. The unit of the setting value is dependent on the display unit system. For "Reference to the other," specify in –199.999 to 199.999 dB. For "Reference to Value," specify in 1E-16 to 99.999 W or in –199.999 to 199.999 dBm.

MT9810
Sensor
Fetch

MT9810 Sensor Fetch.vi

This driver inquires the measurement data. The data unit is dBm, W, dB, or % depending on the display unit system or the reference setting.

Parameter explanation

- Reading [double] .... Measurement value

MT9810
Sensor
Logging
Read

MT9810 Config Logging Parameters.vi

This driver executes/stop the logging. To execute the logging, set the number of samples (the number of measurement data items).

Parameter explanation

- Start/Stop [bool] .... Switching of logging execution/stop
- Number of Samples [I32] .... Setting value of number of measurement data items

MT9810
Sensor
Logging
Read

MT9810 Read Logging Values.vi

This driver outputs the logging data.

Parameter explanation

- Number of Samples [I32] .... Setting value of number of measurement data items
- Number of Samples taken [I32] .... Number of data items read
- Result Array [double array] .... Measurement data (log value)
- Data [str] .... Date identifier, unit model name, date and time of measurement, number of times of averaging, interval time, number of data items measured, data statistics (maximum value (dBm), minimum value (dBm), peak-to-peak value (dB), average value (dBm)) are output as shown below.

```
V1.0,"XXXXXXXXX;YY/MM/DD,hh:mm:ss;XXX;XXX;XXX;XXX,XXX,XXX,XXX"
```

MT9810
Sensor
Min
Max

MT9810 MinMax Values.vi

This driver resets the measurement data of the maximum/minimum values or outputs the measurement data of the maximum/minimum value.

Parameter explanation

- Reset [bool] .... Switching of whether to reset the maximum/minimum value
- Minimum [double] .... Minimum value measurement data (dBm, W). No query is made if reset operation is performed.
- Maximam [double] .... Maximum value measurement data (dBm, W). No query is made if reset operation is performed.
- Change in Power Level [double] .... The difference between the minimum and maximum values. No query is made if reset operation is performed.

(4) Light source unit related VI

MT9810
Source
Config

MT9810 Config Source.vi

This driver sets/inquires the parameters (modulation frequency, attenuation, selection and setting of wavelength)

Parameter explanation

- Frequency [I32] .... Modulation frequency setting value
- Attenuation Level [double] .... Attenuation setting value. The setting range and setting resolution are dependent on the light source unit.
- Wavelength Level [double] .... Wavelength setting value. For two wavelength light source, selection of long wave and short wave is made. The range and resolution of wavelength are dependent on the light source unit.

MT9810
Source
Output

MT9810 Config Source Output.vi

This driver turns ON/OFF the optical output.

Parameter explanation

- Source Output Signal State [bool] ..... Switching of ON/OFF of optical output

Anritsu

MT9810A

Optical Test Set

Remote Control

Operation Manual

■ Read this manual before using the equipment. Keep this manual with the equipment.

Anritsu

MT9810A Optical Test Set Remote Control   Operation Manual

ANRITSU CORPORATION 5-10-27, Minamiazabu, Minato-ku, Tokyo 106-8570 Japan / Phone: 81-3-3446-1111

Document No.: M-W1483AE

Printed in Japan