# GPIB operation

This section contains information about GPIB standards, bus connections, and primary address selection.

## GPIB standards

The GPIB is the IEEE-488 instrumentation data bus with hardware and programming standards originally adopted by the IEEE (Institute of Electrical and Electronic Engineers) in 1975. The SourceMeter conforms to these standards:

- IEEE-488.1-1987
- IEEE-488.2-1992

The above standards define a syntax for sending data to and from instruments, how an instrument interprets this data, what registers should exist to record the state of the instrument, and a group of common commands. The SourceMeter also conforms to this standard:

- SCPI 1996.0 (Standard Commands for Programmable Instruments)

This standard defines a command language protocol. It goes one step farther than IEEE-488.2-1992 and defines a standard set of commands to control every programmable aspect of an instrument.

## GPIB connections

To connect the SourceMeter to the GPIB bus, use a cable equipped with standard IEEE-488 connectors.

To allow many parallel connections to one instrument, stack the connectors. Two screws are located on each connector to ensure that connections remain secure.

To avoid possible mechanical damage, stack no more than three connectors on any one unit.

*NOTE* *To minimize interference caused by electromagnetic radiation, use only shielded IEEE-488 cables. Available shielded cables from Keithley are Models 7007-1 and 7007-2.*

## Primary address

The SourceMeter ships from the factory with a GPIB primary address of 24. When the unit powers up, it momentarily displays the primary address. You can set the address to a value from 0 to 30, but do not assign the same address to another device or to a controller that is on the same GPIB bus (controller addresses are usually 0 or 21).

The primary address can be checked and/or changed from the COMMUNICATIONS option of the Main Menu (Section 1, "Main menu").

## General bus commands

General commands are those commands, such as DCL, that have the same general meaning regardless of the instrument. Table 14-1 lists the general bus commands.

*Table 14-1*
***General bus commands***

| Command | Effect on SourceMeter |
|---|---|
| REN | Goes into remote when next addressed to listen. |
| IFC | Goes into talker and listener idle states. |
| LLO | LOCAL key locked out. |
| GTL | Cancel remote; restore SourceMeter front panel operation. |
| DCL | Returns all devices to known conditions. |
| SDC | Returns SourceMeter to known conditions. |
| GET | Initiates a trigger. |
| SPE, SPD | Serial polls the SourceMeter. |

## Front panel GPIB operation

This section describes aspects of the front panel that are part of GPIB operation, including messages, status indicators, and the LOCAL key.

### Error and status messages

See Appendix B for a list of error and status messages associated with IEEE-488 programming. The instrument can be programmed to generate an SRQ, and command queries can be performed to check for specific error conditions.

### GPIB status indicators

The REM (remote), TALK (talk), LSTN (listen), and SRQ (service request) annunciators show the GPIB bus status. Each of these indicators is described below.

#### REM

This indicator shows when the instrument is in the remote state. When the instrument is in remote, all front panel keys, except for the LOCAL key, are locked out.

When REM is turned off, the instrument is in the local state, and front panel operation is restored.

*NOTE If LLO is in effect, LOCAL will be locked out. OUTPUT ON/OFF is still operational in remote. If ARM:SOUR is set to manual, the TRIG key will be active in remote.*

#### TALK

This indicator is on when the instrument is in the talker active state.

#### LSTN

This indicator is on when the SourceMeter is in the listener active state.

#### SRQ

When this indicator is on, a service request has been generated. This indicator stays on until the serial poll byte is read or all the conditions that caused SRQ have been cleared. See Section 15, "Status Structure," for more information.

### LOCAL key

The LOCAL key cancels the remote state and restores local operation of the instrument.

Pressing the LOCAL key also turns off the REM indicator and returns the display to normal if a user-defined message was displayed.

If the LLO (Local Lockout) command is in effect, the LOCAL key is also inoperative.

For safety reasons, the OUTPUT key will still be active in LLO.

## Programming syntax

The information in this section covers syntax for both common commands and SCPI commands. For information not covered here, see the IEEE-488.2 and SCPI standards. See Section 16 and Section 18 for more details on common and SCPI commands, respectively.

### Command words

Program messages are made up of one or more command words.

## Commands and command parameters

Common commands and SCPI commands may or may not use a parameter. The following are some examples:

| | |
|---|---|
| *SAV <NRf> | Parameter (NRf) required |
| *RST | No parameter used |
| :CALCulate1:STATe <b> | Parameter <b> required |
| :SYSTem:PRESet | No parameter used |

*NOTE At least one space between the command word and the parameter is required.*

**Brackets [ ]** — Some command words are enclosed in brackets ([ ]). These brackets are used to denote an optional command word that does not need to be included in the program message. For example:

:INITiate[:IMMediate]

These brackets indicate that :IMMediate is implied (optional) and does not have to be used. Thus, the above command can be sent in one of two ways:

:INITiate

or

:INITiate:IMMediate

Notice that the optional command is used without the brackets. When using optional command words in your program, do not include the brackets.

**Parameter types** — The following are some of the more common parameter types:

| | |
|---|---|
| <b> | Boolean — Used to enable or disable an instrument operation. 0 or OFF disables the operation, and 1 or ON enables the operation. Example:<br>:CALCulate1:STATe ONEnable Calc 1 math expression |
| <name> | Name parameter — Select a parameter name from a listed group. Example:<br><name>= NEVer<br>= NEXt<br>:TRACe:FEED:CONTrol NEXt |
| <NRf> | Numeric representation format — This parameter is a number that can be expressed as an integer (e.g., 8), a real number (e.g., 23.6), or an exponent (2.3E6). Example:<br>:SYSTem:KEY 11Press EXIT key from over the bus |

<n> Numeric value — A numeric value parameter can consist of an NRf number or one of the following name parameters: DEFault, MINimum, MAXimum. When the DEFault parameter is used, the instrument is programmed to the *RST default value. When the MINimum parameter is used, the instrument is programmed to the lowest allowable value. When the MAXimum parameter is used, the instrument is programmed to the largest allowable value. Examples:

| | |
|---|---|
| :ARM:TIMer 0.1 | Sets timer to 100 msec. |
| :ARM:TIMer DEFault | Sets timer to 0.1 sec. |
| :ARM:TIMer MINimum | Sets timer to 1 msec. |
| :ARM:TIMer MAXimum | Sets timer to 99999.99 sec. |

<numlist> Numlist — Specify one or more numbers for a list. Example:

| | |
|---|---|
| :STATus:QUEue:ENABle (-110:-222) | Enable errors -110 through -222 |

<NDN> Non-decimal numeric — This parameter is used to send values in the binary, octal, or hexadecimal format. The prefix designates the format type:

| | |
|---|---|
| #Bxx...x | #B specifies the binary format. |
| | xx...x is the binary number (using 0s and 1s). |
| #Qxx...x | #Q specifies the octal format. |
| | xx...x is the octal number (values 0 through 7). |
| #Hxx...x | #H specifies the hexadecimal format. |
| | xx...x is the hexadecimal number (values 0 |
| through 9 | |
| | and A through F). |

Examples to send the decimal value 36 in the non-decimal formats:

| | |
|---|---|
| *ESE #b100100 | Binary format |
| *ESE #q44 | Octal format |
| *ESE #h24 | Hexadecimal format |

Angle brackets < > — Angle brackets (< >) are used to denote a parameter type. Do not include the brackets in the program message. For example:

:OUTPut <b>

The <b> indicates a Boolean-type parameter is required. Therefore, to enable the selected source, you must send the command with the ON or 1 parameter as follows:

:OUTPut ON
:OUTPut 1

## Query commands

This type of command requests (queries) the presently programmed status. It is identified by the question mark (?) at the end of the fundamental form of the command. Most commands have a query form:

| | |
|---|---|
| :ARM:TIMer? | Queries the timer interval. |

Most commands that require a numeric parameter (<n>) can also use the DEFault,
MINimum, and MAXimum parameters for the query form. These query forms are used to determine the *RST default value and the upper and lower limits for the fundamental command. Examples are:

| | |
|---|---|
| :ARM:TIMer? DEFault | Queries the *RST default value. |
| :ARM:TIMer? MINimum | Queries the lowest allowable value. |
| :ARM:TIMer? MAXimum | Queries the largest allowable value. |

## Case sensitivity

Common commands and SCPI commands are not case sensitive. You can use upper or lower case and any case combination. Examples:

| | |
|---|---|
| *RST | = *rst |
| :DATA? | = :data? |
| :SYSTem:PRESet | = :system:preset |

*NOTE Using all upper case will result in slightly faster command response times.*

## Long-form and short-form versions

A SCPI command word can be sent in its long-form or short-form version. The command subsystem tables in Section 18 provide the long-form version. However, the short-form version is indicated by upper case characters. Examples:

| | |
|---|---|
| :SYSTem:PRESet | long-form |
| :SYST:PRES | short-form |
| :SYSTem:PRES | long-form and short-form combination |

Note that each command word must be in long-form or short-form, and not something in between. For example, :SYSTe:PRESe is illegal and will generate an error. The command will not be executed.

## Short-form rules

Use the following rules to determine the short-form version of any SCPI command:

- If the length of the command word is four letters or less, no short form version exists. Example:
- :auto = :auto

These rules apply to command words that exceed four letters:

- If the fourth letter of the command word is a vowel (including "y"), delete it and all the letters after it. Example
- :immediate = :imm
- If the fourth letter of the command word is a consonant, retain it but drop all the letters after it. Example:
- :format = :form
- If the command contains a question mark (?; query) or a non-optional number included in the command word, you must include it in the short-form version. Example:
- :delay? = :del?
- Command words or characters that are enclosed in brackets ([ ]) are optional and need not be included in the program message.

*NOTE For fastest response to commands, always use short forms.Program messages*

A program message is made up of one or more command words sent by the computer to the instrument. Each common command is a three letter acronym preceded by an asterisk (*). SCPI commands are categorized in the :STATus subsystem and are used to explain how command words are structured to formulate program messages.

```
:STATus                              Path (Root)
   :OPERation                           Path
      :ENABle <NRf>                        Command and parameter
      :ENABle?                          Query command
   :PRESet                           Command
```

### Single command messages

The above command structure has three levels. The first level is made up of the root command (:STATus) and serves as a path. The second level is made up of another path (:OPERation) and a command (:PRESet). The third path is made up of one command for the :OPERation path. The three commands in this structure can be executed by sending three separate program messages as follows:

:stat:oper:enab <NRf>

:stat:oper:enab?

:stat:pres

In each of the above program messages, the path pointer starts at the root command (:stat) and moves down the command levels until the command is executed.

## Multiple command messages

You can send multiple command messages in the same program message as long as they are separated by semicolons (;). The following is an example showing two commands in one program message:

:stat:oper; :stat:oper:enab <NRf>

When the above is sent, the first command word is recognized as the root command (:stat). When the next colon is detected, the path pointer moves down to the next command level and executes the command. When the path pointer sees the colon after the semicolon (;), it resets to the root level and starts over.

Commands that are on the same command level can be executed without having to retype the entire command path. Example:

:stat:oper:enab <NRf>; enab?

After the first command (:enab) is executed, the path pointer is at the third command level in the structure. Since :enab? is also on the third level, it can be typed in without repeating the entire path name. Notice that the leading colon for :enab? is not included in the program message. If a colon were included, the path pointer would reset to the root level and expect a root command. Since :enab? is not a root command, an error would occur.

## Command path rules

- Each new program message must begin with the root command, unless it is optional (e.g., [:SENSe]). If the root is optional, simply treat a command word on the next level as the root. For fastest operation, do not send optional data.
- The colon (:) at the beginning of a program message is optional and need not be used. However, eliminating the first colon will result in fastest operation. Example:
- :stat:pres = stat:pres
- When the path pointer detects a colon (:) it moves down to the next command level. An exception is when the path pointer detects a semicolon (;), which is used to separate commands within the program message (see next rule).

- When the path pointer detects a colon (:) that immediately follows a semi-colon (;), it resets to the root level.
- The path pointer can only move down. It cannot be moved up a level. Executing a command at a higher level requires that you start over at the root command.

### Using common and SCPI commands in the same message

Both common commands and SCPI commands can be used in the same message as long as they are separated by semicolons (;). A common command can be executed at any command level and will not affect the path pointer. Example:

```
:stat:oper:enab <NRf>; *ESE <NRf>
```

### Program message terminator (PMT)

Each program message must be terminated with an LF (line feed), EOI (end or identify), or an LF+EOI. The bus will hang if your computer does not provide this termination. The following example shows how a multiple command program message must be terminated:

```
:outp on <PMT>
```

### Command execution rules

- Commands execute in the order that they are presented in the program message.
- An invalid command generates an error and, of course, is not executed.
- Valid commands that precede an invalid command in a multiple command program message are executed.
- Valid commands that follow an invalid command in a multiple command program message are ignored.

## Response messages

A response message is the message sent by the instrument to the computer in response to a query command program message.

### Sending a response message

After sending a query command, the response message is placed in the Output Queue. When the SourceMeter is then addressed to talk, the response message is sent from the Output Queue to the computer.

### Multiple response messages

If you send more than one query command in the same program message (see "Multiple command messages," page 14-11), the multiple response messages for all the queries are sent to the computer when the SourceMeter is addressed to talk. The responses are sent in the order the query commands were sent and are separated by semicolons (;). Items within the same query are separated by commas (,). The following example shows the response message for a program message that contains four single item query commands:

0; 1; 1; 0

### Response message terminator (RMT)

Each response is terminated with an LF (line feed) and EOI (end or identify). The following example shows how a multiple response message is terminated:

0; 1; 1; 0 <RMT>

## Message exchange protocol

Two rules summarize the message exchange protocol:

Rule 1. You must always tell the SourceMeter what to send to the computer.

The following two steps must always be performed to send information from the instrument to the computer:

1. Send the appropriate query command(s) in a program message.
2. Address the SourceMeter to talk.

Rule 2. The complete response message must be received by the computer before another program message can be sent to the SourceMeter.

# RS-232 interface operation

*NOTE The programmable aspects of RS-232 operation (baud rate, data bits, parity, and terminator are configured from the COMMUNICATION option of the Main Menu. (See Section 1, "Main menu.")*

## Sending and receiving data

The RS-232 interface transfers data using 8 data bits, 1 stop bit, and no parity. Make sure the device you connect to the SourceMeter also uses these settings.

You can break data transmissions by sending a ^C (decimal 3) or ^X (decimal 18) character string to the instrument. This clears any pending operation and discards any pending output.

## Baud rate

The baud rate is the rate at which the SourceMeter and the programming terminal communicate. Choose one of these available rates:

- 57600
- 38400
- 19200
- 9600
- 4800
- 2400
- 1200
- 600
- 300

The factory selected baud rate is 9600.

When you choose a baud rate, make sure the programming terminal or printer that you are connecting to the SourceMeter can support the baud rate you selected. Both the SourceMeter and the other device must be configured for the same baud rate.

## Data bits and parity

The RS-232 interface can be configured to send/receive data that is 7 or 8 bits long using even, odd, or no parity. No parity is only valid when using 8 data bits.

## Terminator

The SourceMeter can be configured to terminate each program message that it transmits to the controller with any of the following combinations of <CR> and <LF>:

| | |
|---|---|
| <CR> | Carriage return |
| <CR+LF> | Carriage return and line feed |
| <LF> | Line feed |
| <LF+CR> | Line feed and carriage return |

## Flow control (signal handshaking)

Signal handshaking between the controller and the instrument lets the two devices communicate with each other about readiness to receive data. The SourceMeter does not support hardware handshaking (flow control).

Software flow control is in the form of XON and XOFF characters and is enabled when XON-XOFF is selected from the RS-232 FLOW CONTROL menu. When the input queue of the unit becomes more than Ifull, the instrument issues an XOFF command. The control program should respond to this and stop sending characters until the SourceMeter issues the XON, which it will do once its input buffer has dropped below half-full. The SourceMeter recognizes XON and XOFF sent from the controller. An XOFF will cause the instrument to stop outputting characters until it sees an XON. Incoming commands are processed after the <CR> character is received from the controller.

If NONE is the selected flow control, there will be no signal handshaking between the controller and the SourceMeter. Data will be lost if transmitted before the receiving device is ready.

## RS-232 connections

The RS-232 serial port is connected to the serial port of a computer using a straight-through RS-232 cable terminated with DB-9 connectors. Do not use a null modem cable. The serial port uses the transmit (TXD), receive (RXD), and signal ground (GND) lines of the RS-232 standard. Figure 14-1 shows the rear panel connector for the RS-232 interface, and Table 14-2 shows the pinout for the connector.

If your computer uses a DB-25 connector for the RS-232 interface, you will need a cable or adapter with a DB-25 connector on one end and a DB-9 connector on the other, wired straight through (not null modem).

*Figure 14-1*
***RS-232 interface connector***

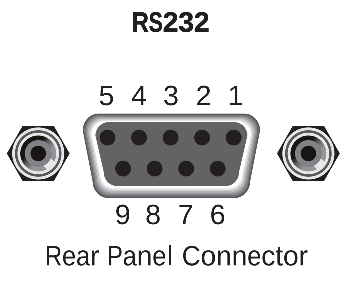

*Table 14-2*
***RS-232 connector pinout***

| Pin numbezr | Description |
|---|---|
| 1 | Not used |
| 2 | TXD, transmit data |
| 3 | RXD, receive data |
| 4 | Not used |
| 5 | GND, signal ground |
| 6 | Not used |
| 7 | RTS, ready to send |
| 8 | CTS, clear to send |
| 9 | Not used |

NOTE: CTA and RTS are tied together.

Table 14-3 provides pinout identification for the 9-pin (DB-9) or 25-pin (DB-25) serial port connector on the computer (PC).

*Table 14-3*
***PC serial port pinout***

| Signal | DB-9 pin number | DB-25 pin number |
|---|---|---|
| DCD, data carrier detect | 1 | 8 |
| RXD, receive data | 2 | 3 |
| TXD, transmit data | 3 | 2 |
| DTR, data terminal ready | 4 | 20 |
| GND, signal ground | 5 | 7 |
| DSR, data set ready | 6 | 6 |
| RTS, request to send | 7 | 4 |
| CTS, clear to send | 8 | 5 |
| RI, ring indicator | 9 | 22 |

## Error messages

See Appendix B for RS-232 error messages.