REMOTE CONTROL AND PROGRAMMING REFERENCE

for all FLUKE 190 family
of
ScopeMeter® test tools

Revision 1a – December 2012

TABLE OF CONTENTS

INTRODUCTION

When using ScopeMeter® testtools, the desire to store any measurement results on the PC, for reference purposes or for documenting the tests made is rather obvious. For that purpose, Fluke offers a dedicated software package: FlukeView ScopeMeter for Windows (or 'SW90W'). It allows the user to store test results, be it voltage readings or complete scope screen images, or long-time recordings of successive measurements over time, or even to make a spectral analysis and see what frequency components are contained within a signal. FlukeView offers all this in a readily available software package, at an affordable price, with user interface in either English, French or German language, to run on a Windows PC.

FlukeView offers the possibility to store complete instrument settings on the PC, as a file, for sharing those with co-workers or for re-using them at a later point in time. And the package even includes a possibility to remotely control the ScopeMeter testtool via a 'virtual instrument' seen on the PC-screen, where the mouse can be used to press keys just like you would on the real instrument; leave your ScopeMeter is a testtroom and control it remotely, while staying in a more safe environment yourself.

FlukeView ScopeMeter software is readily available as a commercial package, for applications on a Windows PC (Win-XP or later). When in doubt about purchasing the package, a trial version is available as a demo package (meaning: no storage nor printing capabilities included). For virtually all ScopeMeter users, FlukeView ScopeMeter is the perfect tool to get all tasks mentioned above covered – quickly and conveniently.

Nevertheless, in some situations end-users of ScopeMeter testtools want to write test protocols themselves, for instance for educational purposes, or want to use ScopeMeters in a larger system-set-up in which FlukeView can't be easily included or can't provide the functionality that the user is looking for. For that reason, we have written the document at hand.

This document contains remote control and programming information for all models of the different series of the Fluke 190-series ScopeMeter® testtools. Previous editions of the document dealt with the 'original 190-series' (instruments that can be recognized by three digit typenumbers without any letters included, e.g. 'Fluke-199'), and later on was extended to also cover the 190B and 190C series (e.g. 'Fluke-196C'). This edition now is created as an update to extend the description and to support the newer 'Fluke 190-series-II' instruments as well (model numbers like 'Fluke-190-204' or 'Fluke-190-502').

The first edition was written around the PM9080 Optically Insulated RS232 Adapter/Cable, which was the only interface cable available at that point in time. At a later stage, the OC4USB was introduced, which includes an RS-232 to USB converter, and which behaves just like the PM9080, yet it uses the electrical interface of the USB-port to establish the physical connection to the (notebook-)PC. From PC-side, however, this interface is seen through a virtual RS-232-port, also known as a COM-port, and command handling is exactly the same as when using the PM9080.

The Fluke-190-series-II has a similar kind of interface/converter built into the instrument mainframe. If you look at the testtool body, you may see a USB-connector on the side of the instrument (in case you can't find it: open the plastic cover on the left-hand side of the testtool, closest to the on/off switch). An optical interface inside the ScopeMeter housing provides the hardware link between that USB-port and the other internal electronics, while ensuring full electrical insulation, needed to use the testtool as a double insulated instrument in CAT III 1000 V applications. As a result of this interface, and seen from the PC-side, also the 190-series-II ScopeMeter testtools are controlled through a virtual RS-232 port (a.k.a. 'a COM-port').

Details about making the interconnection between PC and instrument using PM9080 or the OC4USB may be found in Appendix B of this document.

Drivers for the interface cables for the 190, 190B and 190C series were included on the CD-ROM that is or was found with the instrument.
Details about installing the dedicated USB-drivers for the 190-series-II testtools can be found as Appendix C of this document.

INTRODUCTION TO PROGRAMMING

Basic Programming Information

Before trying to remotely control any ScopeMeter tetstool, a communication link needs to be established between PC and ScopeMeter. Next, the software interface needs to be established. Refer to Appendix B and C for details. Once you have installed the interface and drivers, you can control the ScopeMeter® from the computer with simple communication facilities, such as GWBASIC, QuickBASIC or QBASIC, programming languages from Microsoft® Corporation.

All examples given in this manual are in the QBASIC language but will also run in QuickBASIC.

QuickBASIC, as a compiler, allows you to make executable files of programs, so you can start such programs directly from DOS. For reading and interpreting this manual, knowledge of these programming languages is assumed. QBASIC was supplied with Microsoft MS-DOS 5.0 and higher and with Windows 95, 98, and NT. With newer Windows operating systems, QBasic is no longer included as a standard, but it is available as freeware instead, e.g. through 'quickbasic.sourceforge.net'.
The benefit of presenting the program examples as QBASIC files is in transparency, the easy 'readability' even for programming engineers who are accustomed to work in other, newer programming languages.

Features of the syntax and protocol used with ScopeMeters are as follows:

- Easy input format with a 'forgiving' syntax: all commands consist of two characters that can be either in UPPER or lower case. Parameters, that sometimes follow the command, may be separated from it by one or more separation characters.
- Strict and consistent output format:
    - alpha character responses are always in UPPERCASE.
    - Parameters are always separated by a comma ("," = ASCII 44, see Appendix A).
    - Responses always end with the carriage return code (ASCII 13). Because the carriage return code is a non-visible character (visible neither on screen nor on a print-out), this character is represented as <cr> in the command syntax.
- Synchronization between input and output: after receipt of each command, the ScopeMeter returns an acknowledge character followed by the carriage return code (ASCII 13). This indicates that the command has been successfully received and executed. The computer program must always read this acknowledge response before sending the next command to the ScopeMeter testtool. See below and Appendix D for details.

Commands structure

All remote commands for the ScopeMeter consist of a header, made up of two letters ('alpha characters'), which are sometimes followed by additional parameters. Example:

`RI`	This is the Reset Instrument command. It resets the ScopeMeter.

Some of the commands are followed by one or more parameters to give the ScopeMeter more information. Example:

`SS 8`	This is the Save Setup command. It saves the present acquisition settings in memory. The SS header is followed by a separator (space), then followed by the parameter "8" to indicate where to store the settings. The meaning of this parameter is described in Chapter 3 COMMAND REFERENCE.

Some commands require several parameters. Example:

`WT 9,50,30`	This is the Write Time command. This command requires three parameters. The parameters are separated by a comma, which is called the Program Data Separator. You may use only one comma between the parameters. Also refer to the section 'Data Separators'.

A code at the end of each command tells the ScopeMeter that the command is ended. This is the carriage return code (ASCII 13) and is called the Program Message Terminator. This code is needed to indicate to the ScopeMeter that the command is completed so it can start executing the command.

Also refer to the section 'Command and Response Terminators'.

Responses to receive from the ScopeMeter

After each command sent to the ScopeMeter, there is an automatic response coming back from it, indicated as < acknowledge> which you MUST input (receive and handle), to let the computer know whether or not the previous command has been successfully executed.

Refer to the 'Acknowledge' section below.

There are several commands that ask the ScopeMeter for response data. Such commands are called Queries. Example:

`ID`	This is the IDentification query which asks for the model number and the software version of the ScopeMeter. When the ScopeMeter receives a query it sends the < acknowledge> reply, just as it does after every command received, but now it is followed by the requested response data.

The format of the response data depends upon which query was sent. When a response consists of different response data portions, these are separated by commas (ASCII character 44).

Also refer to the section 'Data Separators', below.

All response data, < acknowledge> as well as following (requested) response data, is terminated by the carriage return code (<cr> = ASCII 13).

Also refer to the section 'Command and Response Terminators'.

Acknowledge

After receiving a command, the ScopeMeter automatically returns the <acknowledge> response to let the computer know whether or not the received command has been successfully executed. This response is a one-digit number followed by <cr> as response terminator.

- If <acknowledge> is 0, it indicates that the ScopeMeter has successfully executed the command.
- If the command was a query, the <acknowledge><cr> response is immediately followed by the requested response data, terminated by <cr>.
- If <acknowledge> is 1 or higher, this indicates that the ScopeMeter testtool has not executed the command successfully. In that case, and if the command was a query, the <acknowledge><cr> response is **NOT** followed by any further response data.

There can be several reasons for a non-zero <acknowledge> response. For more details, refer to Appendix D.

In case of an error you can obtain more detailed status information by using the ST (STATUS) query.

Notes:

1. You MUST ALWAYS input <acknowledge> (that is: make the PC accept this acknowledgement), even when the original command was not a query.
2. Throughout this manual, the <acknowledge> may be written in full (as was just done), or as <ackn>.

Data Separators

Data Separators are used between parameters sent to the ScopeMeter and between values and strings received from the ScopeMeter.
Comma (",") is used as program data separator as well as response data separator:

- Program Data Separator

Name	Character	ASCII value (Decimal)	Comments
Comma	,	44	Single comma allowed

- Response Data Separator

Comma	,	44	

Command and Response Terminators (Message Terminators)

- Command (Program Message) Terminators

A code is needed at the end of each command to tell the ScopeMeter that the command is ended, and that the instrument can start executing the command. This code is called the Program Message Terminator. The code needed for the ScopeMeter is carriage return (ASCII code 13 decimal).

Notes:

1. The carriage return code is a non-visible ASCII character. Therefore this code is represented as <cr> in the Command Syntax and Response Syntax lines given for each command.
2. The QBASIC programming language, which is used for all program examples, automatically adds a carriage return to the end of the command output. (In the QBASIC language, this is the PRINT #.... statement.)

Once <cr> is recognized by the ScopeMeter, the entered command is executed. After EACH command, the ScopeMeter returns <ackn><cr> to the computer to signal the end of the command processing (also see the section 'Acknowledge'.)

- Response (Message) Terminators

The response from the ScopeMeter ends with a carriage return (ASCII 13). This is indicated as <cr> in the Response Syntax for each command.

Typical program sequence - An example

A typical program sequence consists of the following user actions:

1. Set the communication parameters for the RS232 port of the computer to match the ScopeMeter settings (not needed for 190-series II).
2. Send a command or query to the ScopeMeter.
3. Read (input) the acknowledge response from the ScopeMeter.

If the response value is zero, go to step 4.

If the response value is not zero, the ScopeMeter didn't execute the previous command. Read the error message from the following acknowledge subroutine, recover the error, and repeat the command or query (this is not shown in the following program example.)

4. If a query was sent to the ScopeMeter, read the response.
5. The sequence of steps 2, 3, and 4 may be repeated for different commands or queries.
6. Close the communication channel.

The following program example gives this in more detail:

```
'Example of a typical program sequence:
'****************  Beginning of example program  ****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
                    'This QBASIC program line sets the parameters for the
                    'RS232 port (COM1 on the Computer) to match the
                    'ScopeMeter power-on default settings. It also opens a
                    'communication channel (assigned #1) for input or output
                    'through the COM1 port. Your ScopeMeter must be connected
                    'to this port. "RB2048" sets the size of the computer
                    'receive buffer to 2048 bytes to prevent buffer overflow
                    'during communication with the ScopeMeter.
PRINT #1, "ID"      'Outputs the IDENTITY command (query) to the ScopeMeter.
GOSUB Acknowledge   'This subroutine inputs the acknowledge response from
                    'the ScopeMeter and displays an error message if the
                    'acknowledge value is non-zero.
INPUT #1, Response$ 'This inputs the response data from the IDENTITY query.
PRINT Response$     'Displays the queried data.
CLOSE #1            'This closes the communication channel.
END                 'This ends the program.
'******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
Acknowledge:
INPUT #1, ACK                          'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'******************  End of example program  ****************
```

COMMAND REFERENCE

Conventions

Page layout used for each command

XY Header

Each command description starts with a header for quickly finding the command.

This header includes the two-character command indicator (i.c. 'XY') used for the command syntax, and includes the full command name (i.c. 'HEADER'). Note: for the sake of this example, the command indicator differs from the command name, where usually one is an abbreviation of the other.

Purpose:	Explains what the command does or what it is used for.
Command Syntax:	Shows the syntax for the command. Parameters are separated by commas. Commands are terminated by <cr> (carriage return).
Response Syntax:	Shows the format of the response from the ScopeMeter. Responses are terminated by <cr> (carriage return). Each Response Syntax starts with the <ackn> response, followed by the query response if the syntax relates to a query.
Note:	Further notes relating to the command, or giving reference to associated commands.
Example:	This is an example QBASIC program which shows how you can use the command. The example may also include some other commands to show the relation with those commands. The following two comment lines (which start with an ') successively indicate the beginning and the end of an example program.

```
'*****************  Beginning of example program  *****************
'
'
'********************  End of example program  ********************
```

Use an MS-DOS Editor and copy the complete example program between the two lines to a file name with the .BAS extension. Start QBASIC and open this file from the FILE menu. Longer programs may include page breaks. Pay attention to remove those prior to running the program, or make them proceed by the ' (=remark) character to prevent the QBASIC interpreter from interpreting them as an incorrect statement.

Note: some editors may not recognize the quote (') symbol as the start of a REMARK-line. In such case, replace the quote-symbols by a REM-text.

When you have connected the ScopeMeter, you can start the program from the RUN menu.

You may want to refer to the Table of Contents on page 3 for an overview of all available commands, for an easier reference.

Syntax conventions

The Command Syntax and the Response Syntax may contain the following meta symbols and data elements:

UPPERCASE	These characters are part of the syntax. For commands, lower case is also allowed.
<...>	An expression between these brackets is a code, such as <cr> (carriage return) that can not be expressed in a printable character, or it is a parameter that is further specified. Do not actually include these brackets in the command!
[...]	The item between these brackets is optional. This means that you may omit it from the command, or the ScopeMeter may omit this from a response. Do not actually include these brackets in the command!
|	This is a separator between selectable items. This means, that you must choose to use only one of the listed items (exclusive or).
{.....}	Specifies an element that may be repeated zero or more times.
(......)	Grouping of multiple elements.
<binary_character> =	0 to 255
<digit> =	0 to 9
<sign> =	+ or – [this is: positive or negative]
<decimal_number> =	<digit>{<digit>}
<float> =	<mantissa><exponent>
<mantissa> =	<signed_integer>
<exponent> =	<signed_byte>
<signed_integer> =	<binary_character><binary_character> Two bytes representing a signed integer value. The first byte is the most significant and contains the sign bit (bit 7).
<signed_long> =	four <binary_character>'s
<unsigned_integer> =	<binary_character><binary_character> Two bytes representing an unsigned integer value. The first byte is the most significant.
<unsigned_long> =	four <binary_character>'s

AS AUTO SETUP

Purpose:	Switches on the AUTO-ranging mode of the ScopeMeter, which will allow it to automatically adept the actual instrument settings to the input signals applied.
Command Syntax:	`AS<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	• You may select elements of the setup not to modified by the AUTO SET procedure, by going through the USER OPTIONS menu of the ScopeMeter. • The instrument will send the <ackn> as soon as the Autoranging mode has been activated. This doesn't indicate that a stable or the most suitable display of the applied input signal would have been arrived at. Depending on a series of variables, achieving a stable display may take up to 10 seconds. Reaching such stability is not signaled to the PC, though. Once AURTORANGING is active, settings may also change in case the signal changes.
Example:	Example: The following example program sends an AUTO SETUP command to the ScopeMeter. Connect a repetitive signal on INPUT A to see the effect of AUTO SETUP.

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "AS"                          'Sends AUTO SETUP command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  *******************
```

AT ARM TRIGGER

Purpose:	Resets and arms the trigger system for a new acquisition. This command can also be used for single shot acquisitions. When the AT command is given while an acquisition is in progress, this acquisition is aborted and the trigger system is rearmed. The command can also be used to go from other mdoes (e.g. from Replay mode) to 'live' acquisition mode.
Command Syntax:	`AT<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	• Also see the example program for the IS command, which uses the AT command for a single shot application. • The <ackn> is sent only once the ScopeMeter has completed rearming the acquisition system
Example:	The following example program (see next page) arms the trigger system of the ScopeMeter using the AT command. This means that after this command the ScopeMeter starts an acquisition as soon as a trigger is generated by the actual signal (thus: when signal exceeds the trigger level) or from a TA (Trigger Acquisition) command. After the AT (Arm Trigger) command it is assumed that the signal amplitude is sufficient to actually trigger the acquisition. If not, you can use the TA (TRIGGER ACQUISITION) command to force the acquisition to start. This is not useful, however, if you want the acquisition to be actually started by a signal transition, for synchronization purposes.

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "AT"                          'Sends the ARM TRIGGER command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
CLOSE #1
END
 '*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

CM CLEAR MEMORY

Purpose:	Clears all saved setups, waveforms, and screens from internal memory.
Command Syntax:	`CM<cr>`
Response Syntax:	`<ackn><cr>`
Note:	Reciving the <ackn> confirms that the action has been completed. Depending on memory size of the instrument at hand, this may take several seconds.
Example:	See below

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1,"CM"                        'Sends the Clear Memory command.
GOSUB Acknowledge                    'Input acknowledge from ScopeMeter.
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                        'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

DS DEFAULT SETUP

Purpose:	Resets the ScopeMeter to the factory-default settings, just like these were at the moment of original delivery, except for the RS232 communication settings such as baud rate (so as to keep the communications alive).
Command Syntax:	`DS<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	• Wait for at least 2 seconds after the <ackn> reply has been received, so as to let the ScopeMeter settle itself, before you send the next command. • A Master Reset (refer to the Users Manual) performs the same, but also resets the RS232 communication settings to the default values. • Refer to AS (= Auto Setup) for a command that gives an automatic adaptation of settings to match the applied input signals.
Example:	See below

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS
PRINT #1, "DS"                          'Sends DEFAULT SETUP command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
SLEEP 2                                 'Delay (2 sec.) necessary after "DS".
PRINT #1, "ID"                          'Sends the IDENTIFICATION query.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, ID$                           'Inputs identity data from ScopeMeter.
PRINT ID$                               'Displays identity data.
CLOSE #1
END
'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

GD GET DOWN

Purpose:	Switches off the instrument's power system.
Command Syntax:	`GD<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	• If external power (mains adapter) is supplied, you can use the SO command to switch the instrument on again. • If no external power is connected, the instrument can only be switched on manually by pressing the ON/OFF key.
Example:	See below

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS
PRINT #1, "GD"                          'Sends the GET DOWN command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
PRINT "The GET DOWN command switched the ScopeMeter off."
PRINT "Press any key on the PC keyboard to switch "
PRINT "the ScopeMeter on again."
SLEEP
PRINT #1, "SO"                          'Sends the SWITCH ON command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

GL GO TO LOCAL

Purpose:	Switches on the 'local operation mode', this is: enable the keypad for local (manual) control.
Command Syntax:	`GL<cr>`
Response Syntax:	`<ackn><cr>`
Note:	Also refer to the GR (Go to Remote) command.
Example:	The following example (see next page) uses the GR (GO TO REMOTE) command to set the ScopeMeter in the REMOTE state so that the keypad is disabled. Next, the GL (GO TO LOCAL) command is sent to activate the keypad and provide manual (local) operation of the ScopeMeter testtool again.

GR GO TO REMOTE

Purpose:	Sets the ScopeMeter in the remote operation mode which disables local (keypad controlled) operation.
Command Syntax:	`GR<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	You can use the following methods to return to the local operation mode so as to enable manual (local) operation again: 1. Sending the GL (GO TO LOCAL) command. 2. Switching off the instrument (through on/off key on the front panel or through GD command), then switching on again (through SO command or locally, using on/off key on front panel).
Example:	The following example (see next page) uses the GR (GO TO REMOTE) command to set the ScopeMeter in the REMOTE state so that the keypad is disabled. Next, the GL (GO TO LOCAL) command is sent to activate the keypad and provide manual (local) operation of the ScopeMeter testtool again.

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "GR"                          'Sends GO TO REMOTE command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
PRINT "All ScopeMeter keys (except the Power ON/OFF key)
PRINT "are now disabled by the GR (GO TO REMOTE) command."
PRINT "Check this."
PRINT
PRINT "Press any key on the PC keyboard to continue."
SLEEP
PRINT
PRINT #1, "GL"                          'Sends GO TO LOCAL command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
PRINT "The ScopeMeter keys are now enabled again by the "
PRINT "GL (GO TO LOCAL) command."
PRINT "Check this."
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  *******************
```

HO HOLD

Purpose:	Sets the ScopeMeter in the Hold mode. In more detail: the ScopeMeter stops sampling the input channels and stops calculating new measurement results. Bringing the instrument to Hold-mode makes it 'freeze' the display.
Command Syntax:	`HO<cr>`
Response Syntax:	`<ackn><cr>`
Example:	The following example program uses the HO command to stop the signal acquisition and freeze the information on screen; next it uses the AT (ARM TRIGGER) command to enable a new acquisition of applied input signal(s) again.

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS
PRINT #1, "HO"                        'Sends the HOLD command.
GOSUB Acknowledge                     'Input acknowledge from ScopeMeter.
PRINT "The HOLD command has put the ScopeMeter in HOLD."
PRINT "Check on the ScopeMeter screen."
PRINT "Press any key on the PC keyboard to continue and"
PRINT "enable acquisition again."
SLEEP
PRINT #1, "AT"                        'Sends the ARM TRIGGER command to
                                      'enable acquisition again.
GOSUB Acknowledge                     'Input acknowledge from ScopeMeter.
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                         'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

ID IDENTIFICATION

Purpose:	Makes the ScopeMeter send its model identification data to the PC.
Command Syntax:	`ID<cr>`
Response Syntax:	`<ackn><cr>[<identity><cr>]` where, <identity> is an ASCII string containing the following data elements: <model_number>;<software_version>;<creation_date>;<languages>
Example:	The following example program queries the identity data of the ScopeMeter and displays this data on the PC screen.

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "ID"                          'Sends IDENTIFICATION query.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, IDENT$                        'Inputs the queried data.
PRINT IDENT$                            'Displays queried data.
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

IS INSTRUMENT STATUS

Purpose:	Queries the contents of the ScopeMeter's status register. The returned value reflects the present operational status of the testtool itself (see note). Status is indicated by a 16-bit word, presented as an integer value, where each bit signals the logical status of an individual parameter (so: represented as 1 or 0). See table below for details.
Command Syntax:	`IS<cr>`
Note:	Not to be confused with the ST-command (Status Query), which provides status information about the CPL-interface (e.g. 'Illegal Command' or 'Checksum Error').
Response Syntax:	`<ackn><cr>[<status><cr>]` where, <status> = integer value 0 to 65535 (see below)

<status>		
Bit	**Value**	**Status Description**
0	1	Maintenance mode
1	2	Charging
2	4	Recording
3	8	AutoRanging
4	16	Remote
5	32	Battery Connected
6	64	Power Adapter (external power) applied
7	128	Calibration necessary
8	256	Instrument in HOLD mode
9	512	Pre Calibration busy
10	1024	Pre Calibration valid
11	2048	Replay buffer full
12	4096	Triggered
13	8192	Instrument On
14	16384	Instrument Reset occurred
15	32768	Next <status> field available. Not used to date (hence always 0)

Example program – See next page.

```
‘***************** Beginning of example program *****************

CLS                                     ‘Clears the PC screen
OPEN “COM1:1200,N,8,1,CS,DS,RB2048” FOR RANDOM AS #1
PRINT #1, “IS”                          ‘Sends the INSTRUMENT STATUS query
GOSUB Acknowledge                       ‘Input acknowledge from ScopeMeter
INPUT #1, Status$                       ‘Input Instrument Status
StV = VAL(Status$)                      ‘Decimal value of Instrument Status
PRINT “Instrument Status : “; StV
IF (StV AND 1) = 1 THEN PRINT “  ScopeMeter in Maintenance mode.”
IF (StV AND 2) = 2 THEN PRINT “  ScopeMeter charging.”
IF (StV AND 4) = 4 THEN PRINT “  ScopeMeter recording.”
IF (StV AND 8) = 8 THEN PRINT “  AutoRanging active”
IF (StV AND 16) = 16 THEN PRINT “  ScopeMeter remote.”
IF (StV AND 32) = 32 THEN PRINT “  Battery connected.”
IF (StV AND 64) = 64 THEN PRINT “  Power Adapter connected.”
IF (StV AND 128) = 128 THEN PRINT “  Calibration necessary.”
IF (StV AND 256) = 256 THEN PRINT “  ScopeMeter in HOLD.”
IF (StV AND 512) = 512 THEN PRINT “  Pre-calibration busy.”
IF (StV AND 1024) = 1024 THEN PRINT “  Pre-calibration valid.”
IF (StV AND 2048) = 2048 THEN PRINT “  Replay-buffer full.”
IF (StV AND 4096) = 4096 THEN PRINT “  ScopeMeter triggered.”
IF (StV AND 8192) = 8192 THEN PRINT “  ScopeMeter On.”
ELSE
    PRINT “  ScopeMeter Off.”
END IF
IF (StV AND 16384) = 16384 THEN PRINT “  Reset Instrument occurred.”
END
‘******************  Acknowledge subroutine  ********************
    ‘Use this subroutine after each command or query sent to the
    ‘ScopeMeter. This routine inputs the acknowledge response from
    ‘the ScopeMeter. If the response is non-zero, the previous
    ‘command was not correct or was not correctly received by
    ‘the ScopeMeter. Then an error message is displayed and
    ‘the program is aborted.

Acknowledge:
INPUT #1, ACK                           ‘Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT “Error “; ACK; “: “;
    SELECT CASE ACK
        CASE 1
            PRINT “Syntax Error”
        CASE 2
            PRINT “Execution Error”
        CASE 3
            PRINT “Synchronization Error”
        CASE 4
            PRINT “Communication Error”
        CASE IS < 1
            PRINT “Unknown Acknowledge”
        CASE IS > 4
            PRINT “Unknown Acknowledge”
    END SELECT
    PRINT “Program aborted.”
    END
END IF
RETURN
‘*******************  End of example program  *******************
```

PC PROGRAM COMMUNICATIONS *(not used in 190-series-II)*

Purpose:	Programs the baud rate for RS232 communications (not used with 190-series-II).
Command Syntax:	`PC <baudrate>` where, `<baudrate> =` 1200 | 2400 | 4800 | 9600 | 19200 38400 (Fluke 19xC) 57600 (Fluke 19xC, PM9080/101 or OC4USB required) The default baudrate is 1200 . This is set at power-on or after an RI (Reset Instrument) command.
Notes:	• The Fluke 19x/19xC instruments support 1 stopbit, 8 databits and software handshake (a.k.a. "X-on X-off protocol"). Hardware handshaking is not supported. • The 190-series-II has a USB interface. No baudrate setting is required, nor supported. If the PC command is used nevertheless (with proper parameters), this will be accepted and ignored and a positive <ackn> given.
Response Syntax:	`<ackn><cr>`
Example:	See an example for this command under the QP (QUERY PRINT) command description.

PS PROGRAM SETUP

Purpose:	Restores a complete setup, as previously saved internally using the SS (Save Setup) command and queried by the PC using the QS (Query Setup) command, then saved in a string variable or as a file. `<Command 1> -> <Response 1> -> <Command 2> -> <Response 2>`
Command Syntax 1:	`PS [<saved_setup_no>]<cr>` where, <saved_setup_no> = 0 : Actual setup
Response Syntax 1:	`<ackn><cr>`
Notes:	• If no number is given, the sent set-up becomes active immediately, overwriting the present instrument settings. • The 190-series and 190-series-II don't support set-up numbers other than zero (0), so the set-up automatically becomes the actual and active one. Use the SS (save Set-up) command to copy this restored and re-activated set-uop into one of the memory locations for later usage. • Wait for at least two seconds after the <ackn> reply has been received, to allow the ScopeMeter to settle, before sending a next command.
Command Syntax 2:	`<queried_setup><cr>` <queried_setup> = the data returned in reply to the QS command in an earlier stage (omit the <ackn><cr> from the original response, store the received data string only).
Response Syntax 2:	`<ackn><cr>`
Notes:	• Wait for at least two seconds after the <ackn> reply has been received, to allow the ScopeMeter to settle, before sending a next command. • The ScopeMeter sends the <ackn> reply after it has executed the setup from the PS command. You must send the <setup> string as a whole, exactly as returned from the QS (Query Setup) command. • In case the set-up string is recognized not to originate from an instrument of the same model-number and the same firmware revision, the instrument may decide not to act upon receiving the set-up string. If so, no acknowledgement will be sent. • Modification of the set-up string in any way may make the ScopeMeter crash, after which a full Reset may be necessary to recover the instrument. (Refer to the ScopeMeter Users Manual.)
Example:	The example program on the next two pages demonstrates the use of the QS (QUERY SETUP) and the PS (PROGRAM SETUP) commands. The active (or present) setup is queried from the ScopeMeter, copied to the PC and saved to file. The program then invites you to change the ScopeMeter settings, after which the original setup is read from file and sent back to the ScopeMeter.

```
`*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS                                     'Clears the PC screen
GOSUB ClearPort                         'Clears pending data from port.
PRINT #1, "QS"                          'Queries the actual setup data.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
GOSUB Response                          'Writes the setup data to file.
PRINT "Present setup data is stored in the file SETUP0"
PRINT "This setup will now be retrieved from the file and"
PRINT "sent back to the ScopeMeter."
PRINT "To see if this works, change the present settings and"
PRINT "verify if the ScopeMeter returns to the previous"
PRINT "settings when receiving the set-up string."
PRINT
PRINT "Press any key on the PC keyboard to continue."
SLEEP
CLS
PRINT #1, "PS"                          'Program header for programming
                                        'the setup data to the ScopeMeter.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
OPEN "SETUP0" FOR INPUT AS #2           'Opens file SETUP0 for data retrieval.
DO WHILE NOT EOF(2)
    SUCHR$ = INPUT$(1, #2)              'Reads setup data from file
    PRINT #1, SUCHR$;                   'Programs ScopeMeter with the
                                        'setup data stored in SETUP0$.
LOOP
PRINT #1, CHR$(13);                     'Program message terminator
CLOSE #2                                'Close file SETUP0.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
END
`*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
`*******  Clears pending data from the RS232 port ********
ClearPort:
    WHILE LOC(1) > 0
        Dummy$ = INPUT$(1, #1)
    WEND
RETURN
`
`
```

```
'****************** Response subroutine ********************
    'This subroutine reads bytes from the RS232 buffer as long
    'as they enter. When no bytes enter for 1 second, the program
    'assumes that the ScopeMeter has terminated its response.
    'All bytes that enter the buffer are appended to the string
    'Resp$.

Response:
start! = TIMER                          'Wait for bytes (maximum 1 s)
                                        'to enter RS232 buffer
WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
WEND
IF LOC(1) > 0 THEN                      'If RS232 buffer contains bytes
    OPEN "Setup0" FOR OUTPUT AS #2      'File for setup data
    DO
                                        'LOC(1) gives the number of bytes waiting:
        ScopeInput$ = INPUT$(LOC(1), #1)'Input bytes
        PRINT #2, ScopeInput$;
        start! = TIMER
        WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
        WEND
    LOOP WHILE LOC(1) > 0               'Repeat as long as bytes enter
    CLOSE #2
  END IF
RETURN
'*******************  End of example program  *******************
```

QM QUERY MEASUREMENT

Purpose: Queries the ScopeMeter testtool for the availability, validity and characteristics of the various automatic measurements (see Syntax 1) or for the actual results of automatic measurements (see Syntax 2). Readings or measurements must be active on screen in order to be available for transfer to a remote PC.

Command Syntax 1:

```
QM<cr>
```

Response Syntax 1:

```
<ackn><cr>[<reading>{,<reading>}<cr>]
```

where,

<reading> = <no>,<valid>,<source>,<unit>,<type>,<pres>,<resol>

<no> = <decimal_number> (see the following ables)

Measurement numbers are used consistently amongst the 190-, 190B and 190C-series, but different measurement numbers are used with the 190-series-II testtools. Carefully select from either of the two following tables!

Notes: See next page for a not on the usage of Syntax 1 versus Syntax 2 commands.

Table applies to 190, 190B and 190C specifically.

Scope mode	Meter mode	TrendPlot mode	No.	Response
*		*	11	Measurement reading 1
	*		11	Meter absolute reading
	*		19	Meter relative reading (relative to instrument setup reference value)
*		*	21	Measurement reading 2
*			31	Cursor 1 absolute amplitude value
*			41	Cursor 2 absolute amplitude value
*			53	Cursor absolute amplitude value (Maximum)
*			54	Cursor absolute amplitude value (Average)
*			55	Cursor absolute amplitude value (Minimum)
*			61	Cursor relative amplitude value (Delta V)
*			71	Cursor relative time value (delta T)

Table applies to 190-series-II specifically.

Scope mode	Meter mode	TrendPlot mode	No.	Response
*		*	11	Measurement reading 1
	*		11	Meter absolute reading
	*		19	Meter relative reading (relative to instrument setup reference value)
*		*	21	Measurement reading 2
*		*	31	Measurement reading 3
*		*	41	Measurement reading 4
*			61	Cursor 1 absolute amplitude value
*			62	Cursor relative amplitude value (Delta V)
*			71	Cursor 2 absolute amplitude value
*			72	Cursor relative time value (delta T)
*			73	Cursor absolute amplitude value (Maximum)
*			74	Cursor absolute amplitude value (Average)
*			75	Cursor absolute amplitude value (Minimum)
*			76	Cursors frequency

Note: The Syntax 1 command enables the programmer to once ask for the availability, validity and type of measurements that are available given the actual set-up of the instrument. Once this information is known, the use of the command as per Syntax 2 allows for the specific results of individual measurement(s) more rapidly. Those results are then sent as numbers, without any context or unit-of-measure.

<valid> validity of the reading:

1	reading valid
0	reading non-valid

<source> source of the reading, as per the following table:

No.	190-series, 190B-series, 190C-series	190-series-II
1	Voltage channel or Input A (Scope mode)	Input A (Scope mode)
2	Ampere channel or Input B (Scope mode)	Input B (Scope mode)
3	External Input	Input C (Scope mode)
4	---	Input D (Scope mode)
5	---	External Input
12	Input A_over_B (for power and/or phase readings) or M (Mathematics A+B, A-B or AxB)	
21	Input B_over_A (for power and/or phase readings)	

<unit> unit of the reading:

0	None (off)	11	Degrees
1	Volt	12	Celsius
2	Ampere	13	Fahrenheit
3	Ohm	14	percentage (%)
4	Watt	15	dBm 50 Ohm
5	Farad	16	dBm 600 Ohm
6	Kelvin	17	dBVolt
7	seconds	18	dBAmpere
8	hours	19	dBWatt
9	days	20	Volt * Ampere Reactive (VAR)
10	Hertz	21	Volt * Ampere (VA)

<type> reading characteristic of the measurement:

0	None	18	Reactive Power
1	Mean	19	Apparent Power
2	Rms	20	Real Power
3	True rms	21	Harmonic Reactive Power
4	Peak peak	22	Harmonic Apparent Power
5	Peak maximum	23	Harmonic Real Power
6	Peak minimum	24	Harmonic rms
7	Crest factor	25	Displacement Power Factor
8	Period	26	Total Power Factor
9	Duty cycle negative	27	Total Harmonic Distortion
10	Duty cycle positive	28	Total Harmonic Distortion with respect to Fundamental
11	Frequency	29	K Factor (European definition)
12	Pulse width negative	30	K Factor (US definition)
13	Pulse width positive	31	Line Frequency
14	Phase	32	Vac PWM or Vac+dc PWM
15	Diode	33	Rise time
16	Continuity	34	Fall time
17	*(not assigned)*		

	<pres> presentation value of the reading: 0 Absolute value 1 Relative value 2 Logarithmic value 3 Linear value 4 Fahrenheit 5 Celsius
	<resol> resolution of the reading as <float> to determine the least significant digit
Command Syntax 2:	QM <no>{,<no>}<cr> where <no> is as per the table above (carefully select the table for 190, 190B and 190C series, or the table for 190-series-II).
Response Syntax 2:	`<ackn><cr>[<meas_value>{,<meas_value>}<cr>]` where, <meas_value> = [<sign>]<decimal_number>E<sign><decimal_number>
Notes:	• Only measurement results displayed on screen are available for output. • Not all types of readings are available for all models and firmware versions of the ScopeMeter testtools • Maximum number of 10 readings per command. • If one of the readings <no> is non-valid, no readings will be returned. • Only active (valid) readings will be returned.
Example:	See next pages.

‘**************** Beginning of example program ****************

```
CLS                                     ‘Clears the PC screen.
OPEN “COM1:1200,N,8,1,CS,DS,RB2048” FOR RANDOM AS #1
PRINT #1, “QM”                          ‘Queries for active readings
GOSUB Acknowledge                       ‘Input acknowledge from ScopeMeter.
    ‘*** Examines only the 7 inputs of the first reading <no> 11.
INPUT #1, reading.no                    ‘1st <decimal_number>
IF reading.no = 11 THEN
    PRINT “Measurement reading 1”;
ELSEIF reading.no = 21 THEN
    PRINT “Measurement reading 2”;
ELSE
    PRINT “Unknown measurement reading”;
END IF
INPUT #1, validity                      ‘2nd <decimal_number>
IF validity = 1 THEN
    PRINT “ is valid”
ELSE
    PRINT “ is ‘not’ valid”
END IF
INPUT #1, source                        ‘3rd <decimal_number>
PRINT “Source of reading      = “;
IF source = 1 THEN
    PRINT “Voltage channel Input A”
ELSEIF source = 2 THEN
    PRINT “Ampere channel Input B”
ELSEIF source = 3 THEN
    PRINT “Input External”
ELSE PRINT “Unknown source?”
END IF
INPUT #1, unit                          ‘4th <decimal_number>
PRINT “Unit of reading        = “;
IF unit = 1 THEN
    PRINT “Volt”
ELSEIF unit = 2 THEN
    PRINT “Ampere”
ELSEIF unit = 3 THEN
    PRINT “Ohm”
ELSE
    PRINT “Unexpected unit?”
END IF
INPUT #1, types                         ‘5th <decimal_number>
PRINT “Type of reading        = “;
IF types = 1 THEN
    PRINT “Mean value”
ELSEIF types = 2 THEN
    PRINT “Rms value”
ELSEIF types = 3 THEN
    PRINT “True rms value”
ELSE
    PRINT “Unexpected characteristic?”
END IF
INPUT #1, presentation                  ‘6th <decimal_number>
PRINT “Presentation of reading= “;
IF presentation = 0 THEN
    PRINT “Absolute value”
ELSEIF presentation = 1 THEN
    PRINT “Relative value”
ELSEIF presentation = 2 THEN
    PRINT “Logarithmic value”
ELSE
    PRINT “Unexpected value?”
END IF
INPUT #1, resolution                    ‘7th <decimal_number>
PRINT “Resolution of reading  =”; resolution
‘
‘
```

```
GOSUB ClearReadings                     'Clears rest of readings data from port
PRINT #1, "QM 11"                       'Queries Measurement reading 1 or
                                        'Meter absolute reading (Meter mode).
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, result
PRINT "Measurement value ="; result; "V"
CLOSE #1
END
'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******  Clears pending data from the RS232 port *********
ClearReadings:
    WHILE LOC(1) > 0
        LINE INPUT #1, dummy$
    WEND
RETURN
'********************  End of example program  *******************
```

QP QUERY PRINT

Purpose:

Queries a screen dump of the ScopeMeter in a selected printer format.
This allows you to make a copy of the ScopeMeter screen on paper or as an image file held on the PC.
Image aspect ratios:

1 : 1 = width x height = 240 x 240	(as was the screen of e.g. the 190)
4 : 3 = width x height = 320 x 240	(as is, e.g., the screen of the 190-II)

Command Syntax:

```
QP[ <screen_number>,<output_format>[,<block_transfer>]]<cr>
```

where,

<screen_number> = 0 This number represents the screen image number to be printed. Number zero is default, and for some instruments the only possible value, resulting in a copy of the actual screen. If this value is omitted, number zero is assumed as well.

<output_format> = <number>
where

#	190-series	190C-series	190-series-II
0	Epson FX, LQ compatible; 1:1	Epson FX, LQ compatible; 4:3	Epson FX, LQ compatible; 4:3
1	Laser Jet; 4:3	--	--
2	DeskJet; 4:3	--	--
3	PostScript; 4:3	--	--
11	--	PNG format (<block_transfer> mandatory); 4:3	--
12	--	FBRLE2D = Fluke 199C-series color pallet based compressed image format (4:3)	

<block_transfer> =	b	binary format
	B	Binary format

Notes:

- Sending QP without arguments returns the screen image in Epson format (i.e., this command is equivalent to QP 0,0).
- The instrument can only printout the active screen (<screen_number> equal 0) for the “real” printer formats.
  The Binary format is capable to print all (saved screens included) available screens.
  The <block_transfer> options are mandatory for the <Binary> transfer and is not possible for all other formats.
- Two different example programs are given, the first for a ‘real printer’ type of file (QP 0), the second for a block transfer of a PNG-file (QP 0,11).

Response Syntax:

for QP or QP 0,0 or QP 0,1 or QP 0,2 or QP 0,3

(Note: for response to ‘QP 0,11’ (this is: requesting a PNG-file) see below, after the example program).

<ackn><cr>[<printer_data>]
where,
<printer_data> = This data can be sent directly to the printer to get a hard copy of the screen on paper.

Example:

The following program reads the ScopeMeter screen (print) data and copies this data to the file QPfile. This file can be copied to the printer port LPT1, for example. The Read Buffer length for the PC is set to 7500 bytes to prevent buffer overflow during input from the ScopeMeter. The communication speed (baud rate) is set to 19200 and after the data transfer it is reset to 1200 (default baud rate).

```
'*****************  Beginning of example program  *****************
CLS
OPEN "COM1:1200,N,8,1,CS,DS,RB7500" FOR RANDOM AS #1
                                   'Programs COM1 port parameters to
                                   'match the ScopeMeter power-on default
                                   'values.
PRINT #1, "PC 19200"               'Programs ScopeMeter to the maximum
                                   'baud rate.
GOSUB Acknowledge                  'Input acknowledge from ScopeMeter.
CLOSE #1
OPEN "COM1:19200,N,8,1,CS,DS,RB7500" FOR RANDOM AS #1
                                   'Programs COM1 port parameters to
                                   'match the new ScopeMeter settings.
PRINT #1, "QP 0,0"                 'Sends QUERY PRINT data command.
                                   '(actual screen for EPSON print)
GOSUB Acknowledge                  'Input acknowledge from ScopeMeter.
PRINT
PRINT "Busy reading print data !"
PRINT
GOSUB Response
PRINT #1, "PC 1200"                'Programs ScopeMeter back to the
                                   'default baud rate.
GOSUB Acknowledge                  'Input acknowledge from ScopeMeter.

PRINT "Print data copied to file 'QPFILE'."
PRINT "You can copy the file contents to the EPSON Printer."
PRINT "DOS-example:  COPY Qpfile LPT1"
CLOSE                              'Close all files.
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                      'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'
'
```

```
'****************** Response subroutine ********************
    'This subroutine reads bytes from the RS232 buffer as long
    'as they enter. When no bytes enter for 1 second, the program
    'assumes that the ScopeMeter has terminated its response.
    'All bytes that enter the buffer are appended to the string
    'Resp$.
'
Response:
start! = TIMER                          'Wait for bytes (maximum 2 s)
                                        'to enter RS232 buffer
WHILE ((TIMER < (start! + 2)) AND (LOC(1) = 0))
WEND
IF LOC(1) > 0 THEN                      'If RS232 buffer contains bytes
    Resp$ = ""
    OPEN "Qpfile" FOR OUTPUT AS #2      'File for print data
    DO
                                        'LOC(1) gives the number of bytes waiting:
        ScopeInput$ = INPUT$(LOC(1), #1)'Input bytes
        PRINT #2, ScopeInput$;
        start! = TIMER
        WHILE ((TIMER < (start! + 2)) AND (LOC(1) = 0))
        WEND
    LOOP WHILE LOC(1) > 0               'Repeat as long as bytes enter
    CLOSE #2
END IF
RETURN
'********************  End of example program  *******************
```

Response Syntax:	for QP 0,11,b or QP 0,11,B
	
	(Note: for response to other than 'QP 0,11' see above, before the example program).
	
	`<ackn><cr><png_data_length>,<png_data>`
	where,
	<png_data_length> = <digit>{<digit>}
	This field indicates the total number of bytes in the <png_data>.
	
	`<png_data>      = <segment>{<segment>}`
	
	`<segment>       = <ackn><cr>#0<block_header>`
	`                     <block_length><block_data><check_sum><cr>`
	
	`<block_header> = <binary_character>`
	When the most significant bit (bit 7) is set, this block (segment) is the last one in the sequence.
	
	`<block_length> = <unsigned_integer>`
	Specifies the number of <binary_character>'s that follow in the <block_data> field.
	
	`<block_data>    = {<binary_character>}`
	Part of the graphics (PNG) data.
	
	`<check_sum>     = <binary_character>`
	One binary character which represents the sum of all the <binary_character>'s sent after the <block_length> and before the <check_sum>.

Note:	The <png_data> is sent in blocks (segments). When the <block_data> parts of all <segment>'s are concatenated, they form a PNG-format graphics file of length <png_data_length> bytes. The instrument has to be prompted for every block (segment): Command syntax for block transfer: <segment_acknowledge><cr> where, <segment_acknowledge> =

0 Continue: Requests the next segment.
1 Retransmit: Requests retransmission of the just transferred segment.
2 Terminate: Aborts block transfer for this QP command.

The PNG format is specified in: "PNG (Portable Network Graphics) Specification, Version 1.2", by G. Randers-Pehrson et al. (PNG Development Group), July 1999; This document is available from www.libpng.org/pub/png/.

The PNG file consists of the following chunks:

IDHR:	Header chunk describing the image characteristics.
PLTE:	Palette chunk. The first 96 entries form the color palette table, the next 96 entries form the grey-scale palette table for conversion to Black & White.

Notice that the index numbers in the IDAT chunk only refer to the first 96 palette entries. To retrieve the grey-scale values, add 96 to the index numbers.

tEXt:	Text chunk specifying the acquisition date and time of the screen. The Keyword is "Creation Time", the Text field format is "dd-mm-yyyy,hh:mm:ss".
IDAT:	The image data chunk.
IEND:	The image end chunk.

Example: Example for QP 0,11,b or QP 0,11,B:

The following program reads screen (print) data in PNG format from a Fluke 19xC instrument and copies this data to the file SCREEN.PNG. This file can be viewed by loading it into a graphics editor or browser.
The Read Buffer length for the PC is set to 7500 bytes to prevent buffer overflow during input from the ScopeMeter. The communication speed (baud rate) is set to 19200 and after the data transfer it is reset to the default value of 1200 baud.

```
'*****************  Beginning of example program  *****************
CLS
OPEN "COM1:1200,N,8,1,CS,DS,RB7500" FOR RANDOM AS #1
                                        'Programs COM1 port parameters to
                                        'match with the ScopeMeter power-on
                                        'defaults.
PRINT #1, "PC 19200"                    'Programs ScopeMeter to the maximum
                                        'guaranteed baud rate.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
CLOSE #1
OPEN "COM1:19200,N,8,1,CS,DS,RB7500" FOR RANDOM AS #1
                                        'Programs COM1 port parameters to
                                        'match the new ScopeMeter settings.
PRINT #1, "QP 0,11,B"                   'Sends QUERY PRINT data command.
                                        '(actual screen in PNG format)
PRINT
PRINT "Busy reading screen data !"
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
                                        '(This may take 5 to 10 seconds)
ScreenDataLength$ = ""
DO
    C$ = INPUT$(1, #1)
    ScreenDataLength$ = ScreenDataLength$ + C$
LOOP WHILE C$ <> ","
BytesToReceive& = VAL(ScreenDataLength$)

OPEN "SCREEN.PNG" FOR OUTPUT AS #2      'File for PNG data.
BlockNumber% = 1
'
'
```

```
DO
    PRINT “Reading block “; BlockNumber%
    GOSUB ReadBlock                     ‘Read data into BlockData$
    PRINT #2, BlockData$;
    BlockNumber% = BlockNumber% + 1
LOOP WHILE LastBlock% = 0
CLOSE #2
‘
IF BytesToReceive& <> 0 THEN
    PRINT “Block transfer protocol error.”
END IF
PRINT #1, “PC 1200”                     ‘Programs ScopeMeter back to the
                                        ‘default baud rate.
GOSUB Acknowledge                       ‘Input acknowledge from ScopeMeter.
CLOSE #1
PRINT “Print data copied to file ‘SCREEN.PNG’.”
PRINT “You can use a browser program or a graphics editor”
PRINT “to view this file.”
END

‘****************** ReadBlock subroutine ********************
    ‘This subroutine reads one block of data from the RS232 port.
    ‘The actual data bytes received (i.e., excluding the block
    ‘header, checksum and acknowledge bytes) are stored in the
    ‘string BlockData$.
    ‘LastBlock% indicates whether the received block is the last
    ‘one (1) or not (0).

ReadBlock:
PRINT #1, “0”                           ‘Request the next data block.
GOSUB Acknowledge                       ‘Input acknowledge from ScopeMeter.
BlockHeader$ = INPUT$(5, #1)            ‘Read the block header.
IF LEFT$(BlockHeader$, 2) <> “#0” THEN
    PRINT “Block transfer protocol error.”
    CLOSE                               ‘Close all files.
    PRINT “Program aborted.”
    END
END IF

IF (ASC(MID$(BlockHeader$, 3, 1)) AND 128) = 128 THEN
    LastBlock% = 1                      ‘This is the last block.
ELSE
    LastBlock% = 0
END IF
BlockLenHigh% = ASC(MID$(BlockHeader$, 4, 1))
BlockLenLow% = ASC(MID$(BlockHeader$, 5, 1))
BlockLength& = (256 * BlockLenHigh%) + BlockLenLow%
BlockData$ = INPUT$(BlockLength&, #1    ‘Read the block data.
CheckSum$ = INPUT$(2, #1)               ‘Read the checksum
ReceivedCheckSum% = ASC(LEFT$(CheckSum$, 1))
CalculatedCheckSum% = 0
FOR I& = 1 TO BlockLength&
    Byte% = ASC(MID$(BlockData$, I&, 1))
    CalculatedCheckSum% = CalculatedCheckSum% + Byte%
    CalculatedCheckSum% = CalculatedCheckSum% MOD 256
NEXT I&
IF CalculatedCheckSum% <> ReceivedCheckSum% THEN
    PRINT “Checksum error”
    PRINT #1, “2”                       ‘Terminate (abort) QP command.
                                        ‘(We could send “1” instead to request
                                        ‘the block again)
    GOSUB Acknowledge                   ‘Input acknowledge from ScopeMeter.
    CLOSE                               ‘Close all files.
    PRINT “Program aborted.”
    END
END IF
‘
‘
```

```
BytesToReceive& = BytesToReceive& - BlockLength&
RETURN

'******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  *******************
```

QS QUERY SETUP

Purpose:	Queries the ScopeMeter testtool for the present acquisition setup data.
Command Syntax:	`QS [<setup_no>]<cr>` where, <saved_setup_no> = 0 : Actual setup
Response Syntax:	`<ackn><cr>[#0{<node>}<cr>]` where, <node> = <node_header><node_identifier><node_length>[<node_data>] <check_sum> <node_header> = <binary_character> Possible values: 20 hex All nodes except the last (end node) A0 hex End node <node_identifier> = <binary_character>. Unique number for each specific node. <node_length> = <unsigned_integer>. Specifies the number of <binary_character> fields that follow in the <node_data> field. <node_data> = {<binary_character>}. The contents of <node_data> depends on the <node_identifier> and the selected setup. <check_sum> = <binary_character>. Contains the sum of all the binary bytes in the <node_data> field.
Notes:	• Also see the Program Setup (PS) command. • If no set-up number is specified, the data about the active set-up is returned
Example:	An example that includes this QS command can be seen with the section covering the PS (PROGRAM SETUP) command.

QW QUERY WAVEFORM

Purpose: Queries the trace data (administration and/or sample data) of a waveform from the ScopeMeter.
When a waveform is queried that is still being processed (or 'under construction'), that processing is completed before sending the waveform, so as to avoid incomplete traces from being returned.

Command Syntax:

```
QW <trace_no>[,V|S]
```

<trace_no> = <decimal number>, as per the tables below

<trace_no>	**table valid for Fluke 190 models specifically**		
Mode	Channel A	Channel B	
Scope mode	10	20	'normal' and Min/Max trace
Scope Mode, mathematics trace	30		Min/Max of A+B, A-B or AxB
ScopeRecord mode	10	20	Min/Max/Average-trace
TrendPlot mode	11	21	

<trace_no>	**table valid for Fluke 190B and 190C C models specifically**		
Mode	Channel A	Channel B	
Scope mode	10	20	'normal' and Min/Max trace
	12	22	Min/Max trace of channel envelope
	13	23	Min/Max trace of channel reference
Scope Mode, mathematics trace	30		Min/Max of A+B, A-B or AxB
ScopeRecord mode,	10	20	Min/Max/Average-trace
TrendPlot mode	11	21	

<trace_no>	**table valid for Fluke 190-series-II specifically**			
Mode	Channel A	Channel B	Channel C*	Channel D*
Scope mode, scope trace	10	20	50	60
Scope mode, persistence	12	22	52	62
Scope mode, reference trace	13	23	53	63
Scope mode, spectrum	14	24	54	64
Scoperecord mode	10	20	50	60
TrendPlot mode	11	21	31	41
	* In 4-channnel models only			

V | v	Trace values (samples) only
S | s	Setup (administration) data only.

In case V or S is omitted, both trace values and setup data is returned.

Note: The trace data may have different formats, depending on a combination of instrument setttings:
* In scope mode, with waveform options = 'normal' and glitch detect = 'off': waveform data is returned as 16-bit values.
* In all other scope modes: trace is given as min/max trace, using 8-bit values.
* In ScopeRecord mode: trace is given as min/max trace, using 8-bit values.
* In TrendPlot mode: trace is given as min/max/average data in 16-bit values.

Response Syntax:

```
<ackn><cr>[<trace_data><cr>]
```

where,
<trace_data> = <trace_admin> | <trace_samples> | trace_admin>,<trace_samples>

	If the optional parameter (V or S) is omitted: <trace_data> = <trace_admin>,<trace_samples><cr> This includes the complete information about the trace (waveform). For detailed descriptions about the waveform structure, refer to Appendix F. If option V or v (value only) is given: <trace_data> = <trace_samples><cr> For detailed descriptions about the waveform structure, refer to Appendix F. If option S or s (Setup data only) is given: <trace_data> = <trace_admin><cr> where, <trace_admin> = string of hexadecimal characters, representing the setup related to the given <trace_no>.
Example:	The program example for Query Waveofrm is a rather extensive program, and the print-out takes several pages. Given its size, the example has been included as Appendix G, near the end of this document, rather than as part of these descriptive pages.

RD READ DATE

Purpose:	Reads the date setting of the real time clock in the testtool.
Command Syntax:	`RD<cr>`
Response Syntax:	`<ackn><cr>[<date><cr>]` where, <date> = string of the following format: <year>,<month>,<day> e.g. 1999,8,14
Note:	Refer to RT (Read Time), WD (Write Date) and WT (Write Time) commands for associated functionality
Example:	The following example program reads the date setting from the ScopeMeter.

```
'*****************  Beginning of example program  *****************
CLS
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "RD"                          'Sends the READ DATE query.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, SMYear$, SMMonth$, SMDay$     'Inputs the date string.
PRINT "Date "; SMYear$; "-"; SMMonth$; "-"; SMDay$
                                        'Displays the date string.
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  ******************
```

RI RESET INSTRUMENT

Purpose:	Resets the entire instrument, including the CPL interface. The CPL interface is forced into local and operational mode. Only the baudrate setting (for 190, 190B and 190C models) remains unchanged (note: given the USB-interface, the 190-series-II doesn't support any baudrate setting).
Command Syntax:	`RI<cr>`
Response Syntax:	`<ackn><cr>`
Note:	• Wait for at least 2 seconds after the <ackn> reply has been received, so as to allow the testtool to settle itself, before you send any more commands. • For changing the interface settings after a reset, refer to the command PC (Program Communications).
Example:	The following example program (see next page) resets the ScopeMeter and waits for 2 seconds to let the ScopeMeter execute the reset and become ready for next commands. The ScopeMeter is queried for the identification data; this data is input and displayed on the PC screen.

```
'*************  Beginning of example program  **************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "RI"                          'Sends the RESET INSTRUMENT command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
SLEEP 2                                 'Delay (2 s) necessary after reset.
GOSUB ClearPort                         'Clears pending data from port.
PRINT #1, "ID"                          'Sends IDENTIFICATION query.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, IDENT$                        'Inputs the queried data.
PRINT IDENT$                            'Displays queried data.
CLOSE #1
END

'******************  Acknowledge subroutine  *******************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******  Clears pending data from the RS232 port ********
ClearPort:
    WHILE LOC(1) > 0
        Dummy$ = INPUT$(1, #1)
    WEND
RETURN
'*******************  End of example program  ******************
```

RP REPLAY

Purpose:	Queries the instrument for the total number of valid replay screens available (see Syntax 1), or to select and setup the Replay analysis mode and to select and bring on to the instrument's screen a user-selected replay screen-image from memory (see Syntax 2).
Command Syntax 1:	`RP<cr>`
Response Syntax 1:	`<ackn><cr><nr_of_screens><screen_index><cr>` where, <nr_of_screens> = 0 to 100 : number of valid screens (0 = no valid screens available) <screen_index> = 0 to -99 : index of the actual screen
Command Syntax 2:	`RP <screen_index><cr>` where, <screen_index> = 0 to -99 : Replay screen number 0 = most recent (current) screen -99 = oldest screen
Response Syntax 2:	`<ackn><cr>` As a result, the Replay display function is started and the requested replay screen image <screen_index> is activated on the instrument's LCD.
Notes:	• When <screen_index> is omitted, nothing happens. • Replaying screens only works in SCOPE mode. • Use the QP, QS, QM, QW commands for information about the replayed screen and measurements. • Send the AT command to disable the Replay function and to return to normal (running) mode. • Requesting a screen that is not available (screen index out of range) will result in an execution error.
Example:	(See next page)

```
'****************  Beginning of example program  ****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "RP"                          'Queries for number of valid replay
                                        'screens + active screen number
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1, nr.of.screens                 '1st <decimal_number>
IF (nr.of.screens < 0) OR (nr.of.screens > 100) THEN
    PRINT nr.of.screens; " is not a valid number of replay screens"
ELSE
    PRINT "Number of valid replay screens ="; nr.of.screens
END IF
INPUT #1, current.index                 '2nd <decimal_number>
PRINT "Current replay screen number  = "; current.index
PRINT "Previous replay screen number = "; current.index - 1
'
PRINT #1, "RP ";                        'Queries for the current replay screen
PRINT #1, current.index - 1
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
PRINT
PRINT "View the previous Replay screen."
PRINT "Press any key on the PC keyboard to continue."
SLEEP
PRINT #1, "AT"                          'Go back to normal (running) mode
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
CLOSE #1
END
'
'******************  Acknowledge subroutine  ******************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  *******************
```

RS RECALL SETUP

Purpose:	Recalls an internally stored setup. This setup must have been stored in the ScopeMeter manually or with the SS (Save Setup) command. The effect of the RS command is that the instrument setup is recalled and the instrument forced into running mode (190, 190C), or is forced into HOLD state (190-series-II) for further analysis of the waveform and measurements that were stored together with the set-up information/. With instruments of the 190-series-II, the AT command can be used to bring the instrument to ‘running’ state.
Command Syntax:	`RS <setup_reg><cr>` where, <setup_reg> = 1 to 15 : Screen/Setup memories 1001 : Long Record/Replay memory 1 1002 : Long Record/Replay memory 2
Note:	• Some models store setup data and waveform- or recording-data together. Recalling the setup then also recalls the waveform information. • Some instrument configurations of the 190-series-II have more internal memory, which allows for set-up 1 to 30 to be recalled. Such instruments also allow for 10 long recording or replay memories to be used, and recalled. • Also refer to Save Setup command for information about internal storage of setup data.
Response Syntax:	`<ackn><cr>`
Note:	The new setup is active when you have received the <ackn> response from the ScopeMeter.
Example:	The following example program (see next page) saves the present setup in setup memory 8. You are invited to change the actual instrument settings. Then the original settings are recalled from setup memory 8 and made the actual setting again.

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "SS 8"                        'Sends SAVE SETUP command.
                                        'Setup saved in setup memory 8.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter
PRINT " The present setup data is stored in setup memory 8."
PRINT " The remainder of this program will restore these."
PRINT " To test if this works, change the present settings"
PRINT " and verify if the ScopeMeter returns to the original"
PRINT " settings after continuing the program."
PRINT
PRINT " Press any key on the PC keyboard to continue."
SLEEP
PRINT #1, "RS 8"                        'Sends RECALL SETUP command.
                                        'Setup recalled from register 8.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
PRINT
PRINT " Original settings restored"
CLOSE #1
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

RT READ TIME

Purpose:	Reads the real time clock time settings, in a 24-hours format.
Command Syntax:	`RT<cr>`
Response Syntax:	`<ackn><cr>[<time><cr>]` where, <time> = string of the following format: <hours>,<minutes>,<seconds> e.g. 15,4,43
Note:	Refer to RD (Read Date), WD (Write Date) and WT (Write Time) commands for associated functionality
Example:	The following example program reads the time setting from the ScopeMeter.

```
'*****************  Beginning of example program  *****************
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1,"RT"                           'Sends the READ TIME query.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
INPUT #1,Smhour$,Smmin$,Smsec$          'Inputs the time strings.
PRINT "Time "; Smhour$;":";Smmin$;":";Smsec$
                                        'Displays the time string.
END
'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******************  End of example program  ******************
```

SO SWITCH ON

Purpose:	Switches the ScopeMeter on.
Command Syntax:	`SO<cr>`
Response Syntax:	`<ackn><cr>`
Notes:	• This commands requires that the testtool is being powered by the external power adapter (mains adapter). • A time of at least two seconds should be allowed after receiving the <ackn> reply for the instrument to actually start up, before sending any more remote commands.
Example:	An example of this is seen with the command GD (GET DOWN).

SS SAVE SETUP

Purpose:	Saves the present setup in one of the (battery-backup powered) instrument registers.
Command Syntax:	`SS <setup_reg><cr>` where, <setup_reg> = 1 to 15 : Screen/Setup memories. 1001 : Long Record/Replay memory 1 1002 : Long Record/Replay memory 2 When <setup_reg> is omitted, number 1 is assumed.
Notes:	• Some models store setup data and waveform- or recording-data together. Recalling the setup then also recalls the waveform information. • Some instrument configurations of the 190-series-II have more internal memory, which allows for set-up 1 to 30 to be recalled. Such instruments also allow for 10 long recording or replay memories to be used, and recalled. • Refer to Recall Setup command (RS) for information about remotely re-activating a setup previously stored.
Response Syntax:	`<ackn><cr>`
Example:	See an example for this command under RECALL SETUP (RS) command.

ST STATUS QUERY

Purpose:	Queries the error status of the ScopeMeter's CPL interface. The reply is a 16-bit word, presented as an integer value, in which each bit represents the Boolean value of a related error event. After sending the reply to 'ST' or after receiving an RI (Reset Instrument) command, the status word is reset (i.c. set to zero). A complete description of the status word is given in Appendix E.
Command Syntax:	`ST<cr>`
Response Syntax:	`<ackn><cr>[<status>` where, <status> = integer value 0 to 32767
Note:	Not to be confused with IS-command (Instrument Status) which provides information about the operational status of the main-instrument ScopeMeter (e.g. 'battery connected', 'recording mode' or 'instrument on').
Example:	The following example program sends a wrong command to the ScopeMeter to test the Acknowledge subroutine and to check the status returned from the ST query. The acknowledge subroutine contains a GOSUB Status.display to input the status data from the ScopeMeter when the acknowledge response is non-zero (ACK <> 0). (See next page)

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "PC 12345"                    'Sends a baud rate value that is
                                        'out of range for the ScopeMeter.
GOSUB Acknowledge.Status                'Input acknowledge from ScopeMeter
                                        'and the status value if the
                                        'acknowledge value is non-zero.
END
'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    GOSUB Status.display                'Further specifies the error.
    PRINT "Program aborted."
    END
END IF
RETURN
'**************  Displays ScopeMeter status *****************
    'This subroutine gives you further detail in case the
    'acknowledge reply from the ScopeMeter is non-zero.
    '
Status.display:
PRINT #1, "ST"                          'Sends the STATUS query.
GOSUB Acknowledge.Status                'Inputs acknowledge from ScopeMeter.
INPUT #1, STAT                          'Inputs status value.
PRINT "Status " + STR$(STAT) + ": ";
IF STAT = 0 THEN PRINT "No error"
IF (STAT AND 1) = 1 THEN PRINT "Illegal Command"
IF (STAT AND 2) = 2 THEN PRINT "Data format of parameter is wrong"
END IF
IF (STAT AND 4) = 4 THEN PRINT "Parameter out of range"
IF (STAT AND 8) = 8 THEN PRINT "Invalid command in this CPL interface"
END IF
IF (STAT AND 16) = 16 THEN PRINT "Command not implemented"
IF (STAT AND 32) = 32 THEN PRINT "Invalid number of parameters"
END IF
IF (STAT AND 64) = 64 THEN PRINT "Wrong number of data bits"
END IF
IF (STAT AND 512) = 512 THEN PRINT "Conflicting instrument settings"
END IF
IF (STAT AND 16384) = 16384 THEN PRINT "Checksum error"
END IF
RETURN
'*******************  End of example program  ******************
```

TA TRIGGER ACQUISITION

Purpose:	The command will simulate a trigger, meaning it will initiate an acquisition (assuming the ScopeMeter was ready to be started). This command acts as a hardware trigger that starts a new acquisition. In SINGLE shot acquisition mode the trigger system must have been armed with the AT (Arm Trigger) command before the TA command can be used.
Command Syntax:	`TA<cr>`
Response Syntax:	`<ackn><cr>`
Example:	See example below

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "TA"                          'Sends TRIGGER ACQUISITION command.
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

WD WRITE DATE

Purpose:	Writes the real time clock date settings.
Command Syntax:	`WD <date><cr>` where, <date> = string of the following format: <year>,<month>,<date> e.g. 1999,9,21
Response Syntax:	`<ackn><cr>`
Note:	Refer to RD (Read Date), RT (Read Time) and WT (Write Time) commands for associated functionality
Example:	The following example program writes a date into the the ScopeMeters realtime clock system.

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "WD 1999,9,14"                'Sets the real time clock
                                        'to September 14, 1999
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

WT WRITE TIME

Purpose:	Writes the real time clock time settings. The 24-hours time format is adhered to.
Command Syntax:	`WT <time><cr>` where, <time> = string of the following format: <hours>,<minutes>,<seconds> e.g. 15,30,0
Response Syntax:	`<ackn><cr>`
Note:	Refer to RD (Read Date), RT (Read Time) and WD (Write Date) commands for associated functionality
Example:	The following example program writes a time into the the ScopeMeters realtime clock system..

```
'*****************  Beginning of example program  *****************
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "WT 15,28,0"                  'Sets the real time clock to 03:28 p.m..
GOSUB Acknowledge                       'Input acknowledge from ScopeMeter.
END

'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'********************  End of example program  *******************
```

Appendix A ASCII Codes

Part 1 - ASCII codes 00 (00h) through 127 (7Fh).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Part 2 - ASCII codes 128 (80h) through 255 (FFh).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	▪
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

Appendix B Installing the interface cable and drivers

Fluke 120-, Fluke 190-, Fluke 190B- and Fluke 190C-series all have an optical connector for communications with the PC. This connector can be used with either the PM9080 (optical to RS-232 adapter/cable) or with the OC4USB (optical to USB adapter/cable).

The Fluke 190-series-II is itself equipped with a USB-connector for PC communications.

Installing PM9080

In order for the PM9080 to work as interface cable on the computer, do the following:

- Connect the PM9080 to the RS232 port of the computer. If necessary, use a 9-pin to 25-pin adapter and 25-pin gender changer.

- Hook the PM9080 cable up to the ScopeMeter® testtool.

- Turn on the computer and the ScopeMeter.

- Make sure that the communication settings for the RS232 port of the computer (COM-port) match those of the ScopeMeter testool.

After power-on, the default settings of the ScopeMeter are as follows:

```
  1200 baud, No parity, 8 data bits, 1 stop bit
```

You can modify the baud rate with the PC (Program Communication) command. See chapter 3 COMMAND REFERENCE. Other settings are fixed.

You can modify the computer RS232 port settings to match the above ScopeMeter settings with the following DOS command:

```
    MODE COM1:1200,N,8,1
```

This command assumes that COM1 is the RS232 port used on the computer. Replace COM1 in the above command with COM2, COM3, or COM4 if one of these ports is used. You can place this command in the computer startup file AUTOEXEC.BAT so that the default settings for the computer are the same as for the ScopeMeter. If you want to use a higher data transfer speed (baud rate), let your QBASIC program change the settings for both the computer and the ScopeMeter.

See the example under the PC (Program Communication) command in chapter 3 COMMAND REFERENCE.

Installing OC4USB

Use of the OC4USB may require additional drivers. These are supplied together with the instrument, on a CD-ROM; the same CD-ROM that carries the User Manuals. If the PC has internet access, the PC will download and install these drivers automatically from the internet.

Drivers for 190-series-II

Installation of the necessary drivers for the ScopeMeter 190-series-II is described in the User Manual as included with the instrument, as Appendix A. A copy of that description is included as Appendix C within this document at hand.

Appendix C Installing USB-drivers for 190-series-II

Introduction

The Fluke 190-series-II ScopeMeter testtool is equipped s with a USB interface (connector type: USB mini-B) for communications with a computer. In order for communications to be handled by this interface, also dedicated drivers need to be installed onto the computer.

This Appendix describes how to install the drivers for the 190-series-II on a Windows XP computer. Installing on other (newer) Windows platforms will be similar, yet individual screens may look slightly different. Drivers for Windows XP, Vista and Win 7 are available from the Windows Driver Distribution Center, and can be downloaded automatically if your computer is connected to the internet.

The drivers have passed Windows Logo Verification and are signed by Microsoft Windows Hardware Compatibility Publisher. This is a requirement for installation within Windows-7.

Note:

The Fluke 190-series-II instrument requires two drivers to be loaded in sequence:
* first - it requires installation of the Fluke 190-II ScopeMeter USB driver
* next it requires installation of the Fluke USB Serial port driver.

Both of these drivers need to be installed in order to be able to get communications going between the ScopeMeter 190-series-II and the computer.

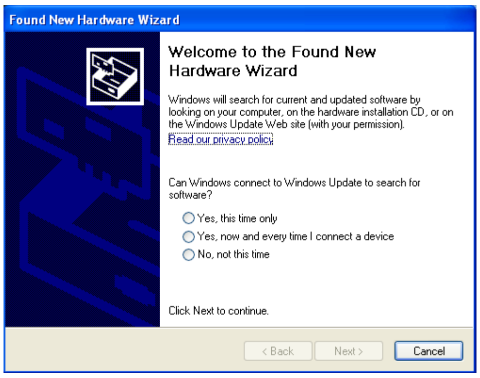
A USB-cable with USB-A connector on one side and a mini-USB-B connector on the other side, is included with each instrument of this family as a standard.

Installing the USB Drivers

To install the USB drivers, do the following:

Connect the Fluke 190-series-II instrument to the PC.
The USB cable can be plugged in and out even when both the computer and the instrument are switched on (‘hot-swapped’). It is not required to power down either of these.

In case the appropriate drivers for the Fluke 190 Series II instrument are not already found on the PC, Windows will show that there is New Hardware detected, and the Wizard for installing new hardware will open.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy

Can Windows connect to Windows Update to search for software?

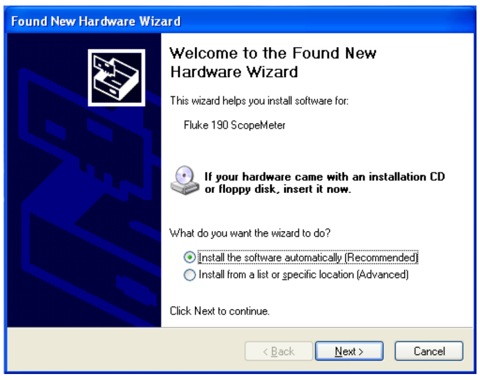
- Yes, this time only
- Yes, now and every time I connect a device
- No, not this time

Click Next to continue.

Depending on your PC settings, Windows may ask for permission to search the Windows Update website on the internet for the latest revision. When you have an internet connection active, it is advised to select ‘*Yes’* and click ‘*Next’*. Alternatively, you may choose to install the drivers from the CD-ROM or from a location on the hard drive, if so preferred, select ’*No, not this time’*.

Downloading from the internet assures you will install the latest version available of the drivers.

In the following window click ‘*Next*’ to install the software automatically. Windows will then download the drivers from the Windows Driver Distribution Center on the internet automatically. If no connection to the internet is available, you may use the CD-ROM, supplied with the ScopeMeter, which also contains a version of these drivers.



Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

Fluke 190 ScopeMeter

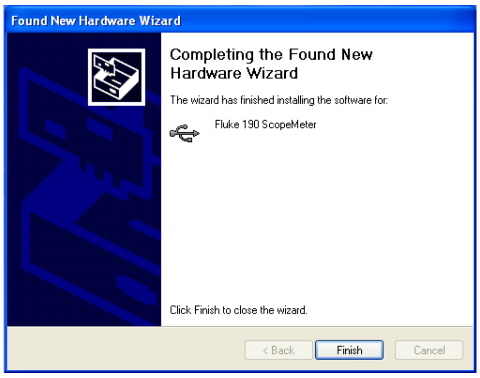
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

[✓ Install the software automatically (Recommended)]
Install from a list or specific location (Advanced)

Click Next to continue.

Follow the instructions on screen. When the driver has finished installation click ‘*Finish*’ to complete the first step of the driver installation.



Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

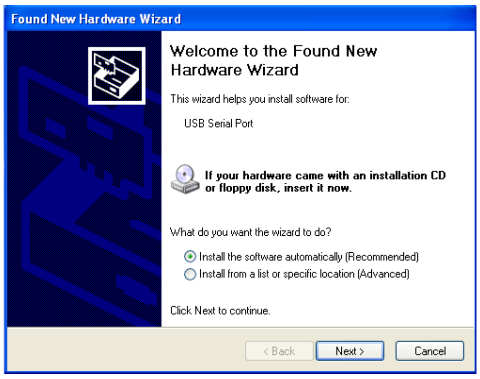
Fluke 190 ScopeMeter

Click Finish to close the wizard.

After completing of the first step, the New Hardware Wizard will start again, this time to install the USB Serial Port Driver.

Click ‘*Next*’ to install the software automatically.

Windows will then download the drivers from the Windows Driver Distribution Center on the internet automatically. If no connection to the internet is available, you may use the CD-ROM, supplied with the ScopeMeter, which also contains a version of these drivers.



Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

USB Serial Port

If your hardware came with an installation CD or floppy disk, insert it now.

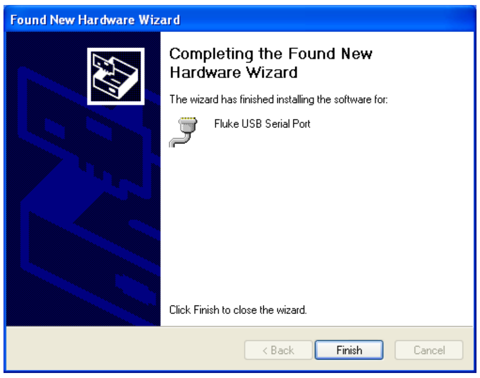
What do you want the wizard to do?

[✓ Install the software automatically (Recommended)]
Install from a list or specific location (Advanced)

Click Next to continue.

Once more, follow the instructions on screen. Once the installation completes, click ‘*Finish*’ to complete the final step of the driver installation.

You are now ready to use the ScopeMeter with FlukeView Software SW90W from version V5.0 onwards.



Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Fluke USB Serial Port

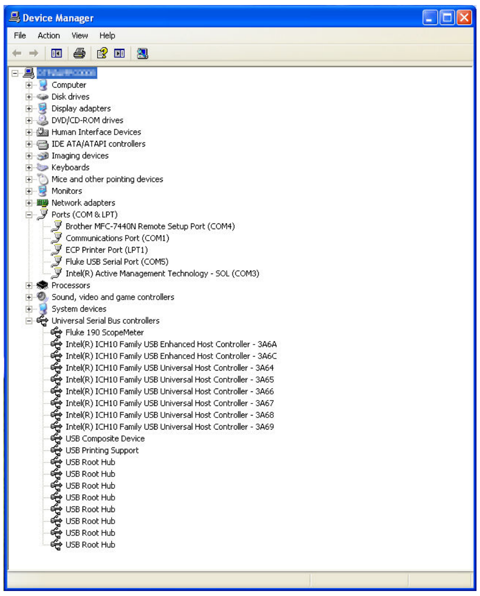
Click Finish to close the wizard.

To check if the drivers were loaded properly, connect the ScopeMeter 190-series-II testtool to your computer, using the USB-cable supplied and open the Device Manager (see the ‘Help’-file of your computer system in case you want to find out how to open the device manager for your particular version of Windows).

From the device manager main menu, click on the ‘+’sign seen with the line on ‘Universal Serial Bus controllers’, to expand this group. The ‘Fluke 190 ScopeMeter’ should be listed in there.

From the device managers main menu, click on the ‘+’sign seen with the line ‘Ports (COM & LPT)’ to expand this group. The ‘Fluke USB Serial Port COM(x)’ should be listed in here.

Note that the COM port number (indicated as ‘x’ here) may differ and is assigned by Windows automatically.



```
[redacted]
    Computer
    Disk drives
    Display adapters
    DVD/CD-ROM drives
    Human Interface Devices
    IDE ATA/ATAPI controllers
    Imaging devices
    Keyboards
    Mice and other pointing devices
    Monitors
    Network adapters
    Ports (COM & LPT)
        Brother MFC-7440N Remote Setup Port (COM4)
        Communications Port (COM1)
        ECP Printer Port (LPT1)
        Fluke USB Serial Port (COM5)
        Intel(R) Active Management Technology - SOL (COM3)
    Processors
    Sound, video and game controllers
    System devices
    Universal Serial Bus controllers
        Fluke 190 ScopeMeter
        Intel(R) ICH10 Family USB Enhanced Host Controller - 3A6A
        Intel(R) ICH10 Family USB Enhanced Host Controller - 3A6C
        Intel(R) ICH10 Family USB Universal Host Controller - 3A64
        Intel(R) ICH10 Family USB Universal Host Controller - 3A65
        Intel(R) ICH10 Family USB Universal Host Controller - 3A66
        Intel(R) ICH10 Family USB Universal Host Controller - 3A67
        Intel(R) ICH10 Family USB Universal Host Controller - 3A68
        Intel(R) ICH10 Family USB Universal Host Controller - 3A69
        USB Composite Device
        USB Printing Support
        USB Root Hub
        USB Root Hub
        USB Root Hub
        USB Root Hub
        USB Root Hub
        USB Root Hub
        USB Root Hub
        USB Root Hub
```

Notes

- Sometimes, other application software may require a dedicated port number (e.g. in the range COM-1.... COM-4). In such case, the COM port number for the ScopeMeter can be changed manually.

  To manually assign a different COM port number, right-click on ‘Fluke USB Serial Port COM-(5)’ and select ‘Properties’. From the Properties-menu, select the tab ‘Port Settings’, and click ‘Advanced…..’ to be allowed to change the port number.

- Sometimes, other PC-application programs installed on the PC, automatically take control over the port we just created for the Fluke 190-series-II ScopeMeter communications. In most of these cases, this can be resolved by unplugging the 190-series-II ScopeMeter USB cable, waiting a few seconds, then reconnecting the cable.

Appendix D Acknowledge Data

The ScopeMeter returns an <acknowledge> reply after each command or query. Throughout this manual, this may be written in full or as <ackn>. The value indicated with this acknowledgement, indicates either correct or incorrect operation. You must read this reply, in order to check for the correct operation and to achieve synchronization between your program and the RS232 interface of the ScopeMeter.

<ackn> VALUE	MEANING
0	No Error
1	Syntax Error (see Note)
2	Execution Error (see Note)
3	Synchronization Error
4	Communication Error
Notes:	The ST query may give you additional information. In case the ScopeMeter detects an error during the execution of a command, it sends the corresponding <ackn> reply, terminates further execution of the command and will be ready to accept a new command.

Syntax Error

Returned when the command is not understood by the ScopeMeter for one of the following reasons :

- Unknown header
- Wrong instructions
- Data format of body is wrong, e.g. alpha characters when decimal data is needed.

Execution Error

Returned when internal processing is not possible because of one of the following reasons:

- Data out of range
- Conflicting instrument settings

Synchronization Error

Returned when the ScopeMeter receives data while it does not expect any data. This can occur as follows:

- The ScopeMeter receives a new command while a previous command or query is not yet completely executed. You can prevent this error by doing the following:
  1. Read the <ackn> reply after each command or query.
  2. If this <ackn> is zero and if a query was sent to the ScopeMeter, read all available response data.

Communication Error

Any framing, parity or overrun error detected on the received data will cause a Communication Error.

Appendix E Status Data

The Status word returned from the ST query gives you additional information in case you have received a non-zero <ackn> reply. The Status word is a 16-bit binary word in which each bit may be set true or false. 'True' represents an error event with a decimal value determined by the bit position (see the table below).

In case more than one bit in the status word is set true, the response from the ST query will be the sum of the decimal values of the individual bits.

Example:
```
<status> = 34,  this equals  32 + 2
  2 = Wrong parameter data format
 32 = Invalid number of parameters
```

Bit	Decimal value	Event description	<ackn> value
0	1	Illegal command	1
1	2	Wrong parameter data format	1
2	4	Parameter out of range	1 or 2
3	8	Command not valid in present state	1
4	16	Command not implemented	2
5	32	Invalid number of parameters	2
6	64	Wrong number of data bits	2
7	128	Flash ROM not present	2
8	256	Invalid flash software	2
9	512	Conflicting instrument settings	2
10	1024	User Request (URQ)	device dependent
11	2048	Flash ROM not programmable	2
12	4096	Wrong programming voltage	2
13	8192	Invalid keystring	1
14	16384	Checksum error	2
15	32768	Next <status> value available (always zero)	

Remarks:

1. A bit in the status word is set when the corresponding error event occurs.
2. Bits do not affect each other.
3. New error events will 'accumulate' in the status word. This means existing bits remain set, until cleared (see below).

The status word is cleared (i.c. all bits reset) as follows:

1. Once the response (the status word) from the ST query has been read.
2. By sending an RI (Reset Instrument) command.

Appendix F Waveform Data

The waveform data that is received in response to the QW (Query Waveform) query, consists of the following data.

<trace_admin>,<trace_samples>

where,

<trace_admin> =	#0<block_header><block_length><trace_result><y_unit><x_unit><y_divisions><x_divisions><y_scale><x_scale><y_step><x_step><y_zero><x_zero><y_resolution><x_resolution><y_at_0><x_at_0><date_stamp><time_stamp><check_sum>

In this,

<block_header> =	`<binary_character>` Possible values: 144 and 0. The value 0 is returned when also the <trace_samples> data block is requested.
<block_length> =	`<unsigned_integer>` This value gives the number of bytes that are transmitted after the <block_length> and before the <check_sum>.
<trace_result> =	`<binary_character>` If bit 0 is set (decimal value 1) the trace is a direct result of a trace acquisition. If bit 1 is set (decimal value 2) the trace is a result of the TrendPlot function (recording numerical results). If bit 2 is set (decimal value 4) either the trace itself is an envelope trace, or an envelope trace is available. If bit 3 is set (decimal value 8) either the trace itself is a reference trace, or a reference trace is available. If bit 4 is set (decimal value 16) either the trace itself is a mathematics trace, or a mathematics trace is available. Note: This <trace_result> information is not available in all instrument types/versions alike.
<y_unit> =	`<unit>`
<x_unit> =	`<unit>` The <unit> is a <binary_character> which value represents the unit: None = 0 <Volt> = 1 <Ampere> = 2 <Ohm> = 3 <Watt> = 4 <Farad> = 5 <Kelvin> = 6 <seconds> = 7 <hours> = 8 <days> = 9 <Hertz> = 10 <Degree> = 11 <degree_Celsius> = 12 <degree_Fahrenheit> = 13 <percentage> = 14 <dBm 50 Ohm> = 15 <dBm 600 Ohm> = 16 <dB Volts> = 17 <dB Ampere> = 18 <dB Watts> = 19 <Volt * Ampere Reactive> = VAR = 20 <Volt * Ampere> = VA = 21
<y_divisions> =	<unsigned_integer> Number of y divisions in which the waveform is displayed on the instrument screen.
<x_divisions> =	<unsigned_integer> Number of x divisions in which the waveform is displayed on the instrument

	screen.
<y_scale> =	<float> Number of units per y division.
<x_scale> =	<float> Number of units per x division.
<y_step> =	<binary_character> Specifies in which scale the <y_scale> is set by the instrument: 1 = 1-2-5 range 2 = 1-2-4 range
<x_step> =	<binary_character> Specifies in which scale the <x_scale> is set by the instrument: 1 = 1-2-5 range 3 = record range 4 = variable range
<y_zero> =	<float> Measurement value for the samples with value zero (0) that you can see as offset value.
<x_zero> =	<float> This field specifies the x-offset of the first sample in <trace_samples>. (is time between trigger moment and first sample.)
<y_resolution> =	<float> This field contains the value that represents the step between two consecutive sample values or in other words the step per least significant bit.
<x_resolution> =	<float> This field contains the value (seconds) that represents the distance between two samples. (is time between two samples.) In the case of an FFT-trace, this value is the frequency of the fundamental (Hz).
<y_at_0> =	<float> This field contains the value corresponding with the lowest horizontal grid line.
<x_at_0> =	<float> This field contains the value corresponding with the most left vertical grid line. Value = 0E0 (not used).
<date_stamp> =	<year><month><day>
<year> =	<digit><digit><digit><digit>; (e.g. 1997)
<month>=	<digit><digit>; (e.g. 12)
<day> =	<digit><digit>; (e.g. 31)
<time_stamp> =	<hours><minutes><seconds>
<hours>=	<digit><digit>; (e.g. 23)
<minutes>=	<digit><digit>; (e.g. 58)
<seconds>=	<digit><digit>; (e.g. 32)
<check_sum> =	<binary_character>

	One binary character which represents the sum of all the <binary_character>‘s sent after the <block_length> and before the <check_sum>.

And where

<trace_samples>=	#0<block_header><block_length><sample_format> <overload><underload><invalid><nbr_of_samples> <samples><check_sum><cr>
<block_header>=	`<binary_character> which is 129.`
<block_length>=	`<unsigned_long>` This (4-bytes) value gives the number of bytes that are transmitted after the <block_length> and before the <check_sum>.
<sample_format>=	<binary_character> This byte specifies the format of the samples. The highest bit (7) defines whether the samples should be interpreted as signed (1) or unsigned values (0). Bit numbers 6, 5, and 4 in <sample_format> define the sample combination (bits 654): 000 = normal trace samples 100 = Min/Max trace samples 110 = Min/Max/Average trace samples 111 = Min=Max trace samples Min=Max=Average trace samples [Average & Display Glitches No] <nbr_of_samples> specifies the number of sample pairs in this case. The bits 0 to 2 in <sample_format> define the number of <binary_character>‘s in which a sample value is represented.
<overload> =	`<sample_value>` This field specifies which value in the trace samples represents the overload value.
<underload> =	`<sample_value>` This field specifies which value in the trace samples represents the underload value.
<invalid> =	`<sample_value>` This field specifies which value in the trace samples represents an invalid sample. Invalid samples can be present at locations in the trace that have not been filled (yet). This can e.g. occur in random sampling.
<nbr_of_samples>=	`<unsigned_integer>` Total number of samples, Min/Max sample pairs, or Min/Average/Max sample triplets that follow.
<samples> =	`{<sample_value>}` In total <nbr_of_samples> will be transmitted.
<sample_value>=	`{<binary_character>}` Depending on the number of <binary_character>‘s in <sample_format>, each <sample_vale> is transmitted in a number of <binary_character>‘s. In case, the <sample_value> contains multiple <binary_character>‘s, the most significant byte is transmitted first.
<check_sum> =	`<binary_character>` One binary character which represents the sum of all the <binary_character>‘s sent after the <block_length> and before the <check_sum>.

Remarks: The instrument will finish any processing on the queried waveform first before sending the data to the remote device (usually: the PC). Therefore, the remote device will not have to do any polling on status bits before the query is sent. For as long as the waveform that was queried for, is still being processed, the processing will continue before making the waveform available to the external device. And so, no incomplete traces will be sent to the remote device.

In case the waveform being processed is being generated in Roll mode, the query will result in an execution error.

The remote device (PC) has the possibility to cancel the query, in case waiting for response takes too long. This can be achieved by sending an <esc> character or hardware break.

Appendix G Program Example for Query Waveform

```
'*****************  Beginning of example program  *****************
    '
    ' If an error occurs in the waveform data, the program stops.
    '
C65536 = 65536                         '2-bytes Maximum constant
C32768 = 32768                         '2-bytes Sign-bit constant
C256   =   256                         '1-byte Maximum constant
C128   =   128                         '1-byte Sign-bit constant
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS
GOSUB ClearPort                        'Clears pending data from port
'
Query$ = "QW 10"                       'Queries normal trace INPUT A when you
                                       'select "Display Glitches No".
                                       'Queries min/max trace INPUT A when
                                       'you select "Persistence" or "Display
                                       'Glitches Yes"; see also Command Syntax.
    '*****
    '* A normal trace is a series of waveform samples consisting
    '* of single waveform points.
    '* A min/max trace is a series of waveform samples consisting
    '* of both minimum and maximum value waveform points.
    '* A min/max/average trace is a series of waveform samples
    '* consisting of minimum, maximum, and average waveform points.
    '*****
PRINT #1, Query$                'Response = <trace_admin>,<trace_samples>
GOSUB Acknowledge               'Inputs acknowledge from ScopeMeter
Resp$ = ""                      'Clears the total Response string
GOSUB Response                  'Writes waveform data to Resp$ & files
GOSUB Interpret.Admin           'Interprets waveform administration data
                                'See also Appendix C
GOSUB Interpret.Samples         'Interprets waveform sample data
GOSUB Create.CSV                'Creates Wave.CSV file from waveform data
                                'as input for Excel, for example.
END
'
'*******************  Acknowledge subroutine  ********************
    'Use this subroutine after each command or query sent to the
    'ScopeMeter. This routine inputs the acknowledge response from
    'the ScopeMeter. If the response is non-zero, the previous
    'command was not correct or was not correctly received by
    'the ScopeMeter. Then an error message is displayed and
    'the program is aborted.
'
'
Acknowledge:
INPUT #1, ACK                          'Reads acknowledge from ScopeMeter.
IF ACK <> 0 THEN
    PRINT "Error "; ACK; ": ";
    SELECT CASE ACK
        CASE 1
            PRINT "Syntax Error"
        CASE 2
            PRINT "Execution Error"
        CASE 3
            PRINT "Synchronization Error"
        CASE 4
            PRINT "Communication Error"
        CASE IS < 1
            PRINT "Unknown Acknowledge"
        CASE IS > 4
            PRINT "Unknown Acknowledge"
'
```

```
    END SELECT
    PRINT "Program aborted."
    END
END IF
RETURN
'*******  Clears pending data from the RS232 port *********
'
'
'
ClearPort:
    WHILE LOC(1) > 0
        Dummy$ = INPUT$(1, #1)
    WEND
RETURN
'
    '****************** Response subroutine ********************
    'This subroutine reads bytes from the RS232 buffer as long
    'as they enter. When no bytes enter for 1 second, the program
    'assumes that the ScopeMeter has terminated its response. All
    'bytes that enter the buffer are appended to the string Resp$
    'and are written to the following files:
    '  File Waveform : the waveform data bytes
    '  File Waveresp : the waveform ASCII values
    '
'
'
'
Response:
start! = TIMER
'Wait for bytes (maximum 1 s) to enter RS232 buffer
WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
WEND
IF LOC(1) > 0 THEN                      'If RS232 buffer contains bytes
    OPEN "WaveForm" FOR OUTPUT AS #2
                                        'File to contain the waveform data bytes
    docount = 1
    total.count& = 0
    DO
        ' LOC(1) gives the number of bytes waiting:
        total.count& = total.count& + LOC(1)
        ScopeInput$ = INPUT$(LOC(1), #1)    'Input bytes
        PRINT #2, ScopeInput$;
        PRINT total.count&;
        Resp$ = Resp$ + ScopeInput$
        start! = TIMER
        WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
        WEND
        docount = docount + 1
    LOOP WHILE LOC(1) > 0               'Repeat as long as bytes enter
    CLOSE #2
    PRINT
END IF
    '
    '****** Write the total Response string to file WaveResp
    '
OPEN "WaveResp" FOR OUTPUT AS #3
PRINT "Response data length = "; LEN(Resp$)
PRINT #3, "Response data length = "; LEN(Resp$)
FOR i = 1 TO LEN(Resp$)
    PRINT #3, ASC(MID$(Resp$, i, 1));
NEXT i
CLOSE #3: RETURN
'
'
'
'
'
'
```

```
Interpret.Admin:
Resp.Count = 1                          'Byte counter for Resp$
SumCheck1% = 0                          'Sumcheck byte for Resp$
    '
    '***** Interpret the <trace_admin> waveform data bytes
    '***** in the Resp$ string (see Appendix C).
    '
    '***** 2 bytes <trace_admin> block trailing : #0
IF MID$(Resp$, Resp.Count, 2) <> "#0" GOTO Wave.Error
Resp.Count = Resp.Count + 2
    '
    '***** 1 byte <block_header>
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb <> 128 AND nb <> 0 GOTO Wave.Error
Resp.Count = Resp.Count + 1
    '
    '***** 2 bytes <block_length>
Block1.Length = ASC(MID$(Resp$, Resp.Count, 1)) * 256
Block1.Length = Block1.Length + ASC(MID$(Resp$, Resp.Count + 1, 1))
Resp.Count = Resp.Count + 2
    '
    '***** 1 byte <trace_result> : 0, 1, or 2
Trace.Result = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Trace.Result
IF Trace.Result < 0 OR Trace.Result > 2 GOTO Wave.Error
Resp.Count = Resp.Count + 1
    '
    '***** 1 byte <y_unit>
Y.Unit = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Y.Unit
Resp.Count = Resp.Count + 1
PRINT "<y_unit>          ="; Y.Unit,
    '
    '***** 1 byte <x_unit>
X.Unit = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + X.Unit
Resp.Count = Resp.Count + 1
PRINT "      <x_unit>          ="; X.Unit
    '
    '***** 2 bytes <y_divisions>
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Sample.Byte
Y.Divisions = Sample.Byte * 256
Sample.Byte = ASC(MID$(Resp$, Resp.Count + 1, 1))
SumCheck1% = SumCheck1% + Sample.Byte
Y.Divisions = Y.Divisions + Sample.Byte
Resp.Count = Resp.Count + 2
PRINT "<y_divisions>     ="; Y.Divisions,
    '
    '***** 2 bytes <x_divisions>
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Sample.Byte
X.Divisions = Sample.Byte * 256
Sample.Byte = ASC(MID$(Resp$, Resp.Count + 1, 1))
SumCheck1% = SumCheck1% + Sample.Byte
X.Divisions = X.Divisions + Sample.Byte
Resp.Count = Resp.Count + 2
PRINT "      <x_divisions>     ="; X.Divisions
'
DIM expscale(2)                         'Exponents for Y/X.Scale
DIM YXscale#(2)                         'Values for Y/X.Scale
'
    '***** 3 bytes <y_scale> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <y_scale> = <sign><mantissa> E <sign><exponent>
    '*****            Example: +123E-4 = 123 / 10000 = 0.0123
'
'
```

```
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
    IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb +ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
expscale(1) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXscale#(1) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <x_scale> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <x_scale> = <sign><mantissa> E <sign><exponent>
    '*****            Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
expscale(2) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXscale#(2) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 1 byte <y_step>
Y.Step = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Y.Step
Resp.Count = Resp.Count + 1
PRINT "<y_step>          ="; Y.Step,
    '
    '***** 1 byte <x_step>
X.Step = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + X.Step
Resp.Count = Resp.Count + 1
PRINT "      <x_step>          ="; X.Step
'
'
'
DIM exponent(6)                  'Exponents for Y/X.Zero & Y/X.Resol & Y/X.At.0
DIM YXvalue#(6)                  'Values for Y/X.Zero & Y/X.Resol & Y/X.At.0
    '
    '***** 3 bytes <y_zero> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <y_zero> = <sign><mantissa> E <sign><exponent>
    '*****            Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
'
```

```
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(1) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(1) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <x_zero> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <x_zero> = <sign><mantissa> E <sign><exponent>
    '*****            Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(2) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(2) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <y_resolution> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <y_resolution> = <sign><mantissa> E <sign><exponent>
    '*****                  Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(3) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(3) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <x_resolution> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <x_resolution> = <sign><mantissa> E <sign><exponent>
    '*****                  Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(4) = ASC(MID$(Resp$, Resp.Count + 2, 1))
'
```

```
YXvalue#(4) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <y_at_0> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <y_at_0> = <sign><mantissa> E <sign><exponent>
    '*****                  Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(5) = ASC(MID$(Resp$, Resp.Count + 2, 1)) YXvalue#(5) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 3 bytes <x_at_0> = <mantissa_high><mantissa_low><exponent>
    '***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
    '***** <x_at_0> = <sign><mantissa> E <sign><exponent>
    '*****                  Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
    nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256             'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                       'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(6) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(6) = nb
Resp.Count = Resp.Count + 3
    '*****
    '* Further calculation after 'Signed.Samples' determination
    '*****
    '***** 8 bytes <year><month><date>
FOR i = 0 TO 7
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
Year$ = MID$(Resp$, Resp.Count, 1)
Year$ = Year$ + MID$(Resp$, Resp.Count + 1, 1)
Year$ = Year$ + MID$(Resp$, Resp.Count + 2, 1)
Year$ = Year$ + MID$(Resp$, Resp.Count + 3, 1)
Month$ = MID$(Resp$, Resp.Count + 4, 1)
Month$ = Month$ + MID$(Resp$, Resp.Count + 5, 1)
Day$ = MID$(Resp$, Resp.Count + 6, 1)
Day$ = Day$ + MID$(Resp$, Resp.Count + 7, 1)
Resp.Count = Resp.Count + 8
PRINT "<date_stamp>      = "; Year$ + "-" + Month$ + "-" + Day$;
    '
    '***** 6 bytes <hours><minutes><seconds>
FOR i = 0 TO 5
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
Hours$ = MID$(Resp$, Resp.Count, 1)
Hours$ = Hours$ + MID$(Resp$, Resp.Count + 1, 1)
'
```

```
Minutes$ = MID$(Resp$, Resp.Count + 2, 1)
Minutes$ = Minutes$ + MID$(Resp$, Resp.Count + 3, 1)
Seconds$ = MID$(Resp$, Resp.Count + 4, 1)
Seconds$ = Seconds$ + MID$(Resp$, Resp.Count + 5, 1)
Resp.Count = Resp.Count + 6
PRINT "    <time_stamp>      = "; Hours$+":"+Minutes$+":"+Seconds$
    `
    `***** 1 byte <check_sum>
Check.Sum% = ASC(MID$(Resp$, Resp.Count, 1))
IF Check.Sum% <> (SumCheck1% MOD 256) GOTO Wave.Error
Resp.Count = Resp.Count + 1
PRINT "<check_sum> ="; Check.Sum%; " & ";
PRINT "SumCheck1 MOD 256 ="; SumCheck1% MOD 256
RETURN
Wave.Error:
PRINT "Waveform admin error at byte  :"; Resp.Count
PRINT "Waveform decimal byte value   ="; ASC(MID$(Resp$,Resp.Count,1)
PRINT "SumCheck so far (MOD 256)     ="; SumCheck1% MOD 256
CLOSE: END
`
Interpret.Samples:
    `
    `***** Interpret the <trace_samples> waveform data bytes
    `***** in the Resp$ string (see Appendix C).
    `*****
    `***** 1 byte separator admin/samples : ,
    `***** 2 bytes <trace_samples> block trailing : #0
    `
SumCheck2% = 0
IF MID$(Resp$, Resp.Count, 3) <> ",#0" GOTO Wave2.Error
Resp.Count = Resp.Count + 3
    `
    `***** 1 byte <block_header>
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb <> 144 GOTO Wave2.Error
Resp.Count = Resp.Count + 1
    `
    `***** 4 bytes <block_length>
Block2.Length& = ASC(MID$(Resp$, Resp.Count, 1))
FOR i = 1 TO 3
    Block2.Length& = Block2.Length& * 256
    Block2.Length& = Block2.Length& + ASC(MID$(Resp$,Resp.Count+i,1))
NEXT i
Resp.Count = Resp.Count + 4
PRINT "Number of sample chars ="; Block2.Length&
OPEN "Samples" FOR OUTPUT AS #4
PRINT #4, "Number of sample chars ="; Block2.Length&
    `
    `***** 1 byte <sample_format>
Sample.Format = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = SumCheck2% + Sample.Format
IF (Sample.Format AND 128) = 128 THEN
    Signed.Samples = 1
ELSE
    Signed.Samples = 0
END IF
IF (Sample.Format AND 112) = 64 THEN    `bits 6, 5, 4
    MinMax.Samples = 1                 `Min/Max=100
ELSEIF (Sample.Format AND 112) = 96 THEN
    MinMax.Samples = 2                 `Min/Max/Ave=110
ELSEIF (Sample.Format AND 112) = 0 THEN
    MinMax.Samples = 0                 `Normal=000
ELSEIF (Sample.Format AND 112) = 112 THEN
    IF MID$(Query$, 5, 1) = "1" THEN    `TrendPlot
        MinMax.Samples = 2              `Min=Max=Ave=111
    ELSE                                `Average Min/Max
        MinMax.Samples = 1              `Min=Max=111
`
```

```
END IF
ELSE
MinMax.Samples = 7                     'Unknown format!
END IF
Sample.Bytes = Sample.Format AND 7
IF Sample.Bytes = 1 THEN               'Single-byte samples
    CLimit = C128 : CMaxim = C256
ELSE                                   'Double-byte samples
    CLimit = C32768 : CMaxim = C65536
END IF
'
'
'
Resp.Count = Resp.Count + 1
PRINT "Signed.Samples        = ";
PRINT #4, "Signed.Samples        = ";
IF Signed.Samples = 1 THEN
    PRINT "TRUE     "; : PRINT #4, "TRUE"
ELSE
    PRINT "FALSE    "; : PRINT #4, "FALSE"
END IF
PRINT "Sample.Format     = ";
PRINT #4, "Sample.Format     = ";
IF MinMax.Samples = 0 THEN
    PRINT "Single"
    PRINT #4, "Single"
ELSEIF MinMax.Samples = 1 THEN
    PRINT "Min/Max"
    PRINT #4, "Min/Max"
ELSEIF MinMax.Samples = 2 THEN
    PRINT "Min/Max/Ave"
    PRINT #4, "Min/Max/Ave"
ELSE
    PRINT "Unknown: "; OCT$(Sample.Format); " octal"
    PRINT #4, "Unknown: "; OCT$(Sample.Format); " octal"
END IF
PRINT "Number of Sample.Bytes ="; Sample.Bytes
PRINT #4, "Number of Sample.Bytes ="; Sample.Bytes
    '*****
    '* Further calculation now that 'Signed.Samples' is determined
    '*****
FOR j = 1 TO 2
    IF expscale(j) > 127 THEN          'Negative exponent
        expscale(j) = 256 - expscale(j)
        FOR i = 1 TO expscale(j)
            YXscale#(j) = YXscale#(j) / 10
        NEXT i
    ELSE                               'Positive exponent
        FOR i = 1 TO expscale(j)
            YXscale#(j) = YXscale#(j) * 10
        NEXT i
    END IF
NEXT j
Y.Scale = YXscale#(1)
X.Scale = YXscale#(2)
PRINT "<y_scale>         ="; Y.Scale,
PRINT "      <x_scale>         ="; X.Scale
'
FOR j = 1 TO 6
    IF exponent(j) > 127 THEN          'Negative exponent
        exponent(j) = 256 - exponent(j)
        FOR i = 1 TO exponent(j)
            YXvalue#(j) = YXvalue#(j) / 10
        NEXT i
    ELSE                               'Positive exponent
        FOR i = 1 TO exponent(j)
            YXvalue#(j) = YXvalue#(j) * 10
'
```

```
    NEXT i
    END IF
NEXT j
`
`
Y.Zero  = YXvalue#(1)
X.Zero  = YXvalue#(2)
Y.Resol = YXvalue#(3)
X.Resol = YXvalue#(4)
Y.At.0  = YXvalue#(5)
X.At.0  = YXvalue#(6)
PRINT "<y_zero>          ="; Y.Zero,
PRINT "      <x_zero>          ="; X.Zero
PRINT "<y_resolution>    ="; Y.Resol,
PRINT "      <x_resolution>    ="; X.Resol
PRINT "<y_at_0>          ="; Y.At.0,
PRINT "      <x_at_0>          ="; X.At.0
    `
    `***** <Sample.Bytes> bytes <overload> value
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = SumCheck2% + Sample.Byte
IF (Signed.Samples = 1) AND (Sample.Byte >= 128) THEN
    Sample.Byte = - (256 - Sample.Byte)
END IF
Overload& = Sample.Byte
FOR i = 2 TO Sample.Bytes
    Sample.Byte = ASC(MID$(Resp$, Resp.Count + i - 1, 1))
    SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
    Overload& = Overload& * 256 + Sample.Byte
NEXT i
IF (Signed.Samples = 0) OR (Overload& < CLimit) THEN
    Overload.Value = Overload& * Y.Resol       'Positive value
ELSE                                           'Negative value
    Overload.Value = - ((CMaxim - Overload&) * Y.Resol)
END IF
Resp.Count = Resp.Count + Sample.Bytes
PRINT "Overload sample value  ="; Overload&; Overload.Value
PRINT #4, "Overload sample value  ="; Overload&; Overload.Value
    `
    `***** <Sample.Bytes> bytes <underload> value
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = SumCheck2% + Sample.Byte
IF (Signed.Samples = 1) AND (Sample.Byte >= 128) THEN
    Sample.Byte = - (256 - Sample.Byte)
END IF
Underload& = Sample.Byte
FOR i = 2 TO Sample.Bytes
    Sample.Byte = ASC(MID$(Resp$, Resp.Count + i - 1, 1))
    SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
    Underload& = Underload& * 256 + Sample.Byte
NEXT i
IF (Signed.Samples = 0) OR (Underload& < CLimit) THEN
    Underload.Value = Underload& * Y.Resol     'Positive value
ELSE                                           'Negative value
    Underload.Value = - ((CMaxim - Underload&) * Y.Resol)
END IF
Resp.Count = Resp.Count + Sample.Bytes
PRINT "Underload sample value ="; Underload&; Underload.Value
PRINT #4, "Underload sample value ="; Underload&; Underload.Value
    `
    `***** <Sample.Bytes> bytes <invalid> value
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = SumCheck2% + Sample.Byte
IF (Signed.Samples = 1) AND (Sample.Byte >= 128) THEN
    Sample.Byte = - (256 - Sample.Byte)
END IF
Invalid& = Sample.Byte
`
```

```
FOR i = 2 TO Sample.Bytes
    Sample.Byte = ASC(MID$(Resp$, Resp.Count + i - 1, 1))
    SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
    Invalid& = Invalid& * 256 + Sample.Byte
NEXT i
IF (Signed.Samples = 0) OR (Invalid& < CLimit) THEN
    Invalid.Value = Invalid& * Y.Resol  'Positive value
ELSE                                    'Negative value
    Invalid.Value = - ((CMaxim - Invalid&) * Y.Resol)
END IF
Resp.Count = Resp.Count + Sample.Bytes
PRINT "Invalid sample value   ="; Invalid&; Invalid.Value
PRINT #4, "Invalid sample value   ="; Invalid&; Invalid.Value
    '
    '***** 2 bytes <nbr_of_samples>
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
Nbr.Of.Samples = Sample.Byte
Sample.Byte = ASC(MID$(Resp$, Resp.Count + 1, 1))
SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
Nbr.Of.Samples = Nbr.Of.Samples * 256 + Sample.Byte
IF MinMax.Samples = 1 THEN              'Min/Max pair of samples
    Nbr.Of.Samples = Nbr.Of.Samples * 2
END IF
IF MinMax.Samples = 2 THEN              'Min/Max/Ave samples
    Nbr.Of.Samples = Nbr.Of.Samples * 3
END IF
Resp.Count = Resp.Count + 2
PRINT "Number of samples      ="; Nbr.Of.Samples
PRINT #4, "Number of samples      ="; Nbr.Of.Samples
    '
    '***** <Sample.Bytes> bytes <sample_value>'s
    '
DIM Sample.Value(Nbr.Of.Samples) AS LONG
    FOR i = 1 TO Nbr.Of.Samples         'Sample loop
        Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
        SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
    IF (Signed.Samples = 1) AND (Sample.Byte >= 128) THEN
        Sample.Byte = - (256 - Sample.Byte)
    END IF
        Sample.Value&(i) = Sample.Byte
    IF Sample.Bytes > 1 THEN            'More sample bytes
        FOR j = 2 TO Sample.Bytes
            Sample.Byte = ASC(MID$(Resp$, Resp.Count + j - 1, 1))
            SumCheck2% = (SumCheck2% + Sample.Byte) MOD 256
            Sample.Value&(i) = Sample.Value&(i) * 256 + Sample.Byte
        NEXT j
    END IF
Resp.Count = Resp.Count + Sample.Bytes
    IF i=1 OR i=2 OR i = Nbr.Of.Samples-1 OR i = Nbr.Of.Samples THEN
        IF (Signed.Samples = 0) OR (Sample.Value&(i) < CLimit) THEN
            Ampl.Value = Sample.Value&(i) * Y.Resol   'Positive value
        ELSE                            'Negative value
            Ampl.Value = - ((CMaxim - Sample.Value&(i)) * Y.Resol)
        END IF
        PRINT "Sample"; i; "="; Sample.Value&(i); Ampl.Value
    END IF
    PRINT #4, "Sample"; i; "="; Sample.Value&(i); Ampl.Value
NEXT i
    '
    '***** 1 byte <check_sum>
Check.Sum% = ASC(MID$(Resp$, Resp.Count, 1))
IF Check.Sum% <> (SumCheck2% MOD 256) GOTO Wave2.Error
Resp.Count = Resp.Count + 1
PRINT "<check_sum> ="; Check.Sum%; " & ";
PRINT "SumCheck2 MOD 256 ="; SumCheck2% MOD 256
PRINT #4, "<check_sum> ="; Check.Sum%; " & ";
'
```

```
PRINT #4, "SumCheck2 MOD 256 ="; SumCheck2% MOD 256
    `
    `***** 1 byte CR
C.R = ASC(MID$(Resp$, Resp.Count, 1))
IF C.R <> 13 GOTO Wave2.Error
Resp.Count = Resp.Count + 1
CLOSE #4: RETURN
Wave2.Error:
PRINT "Waveform sample error at byte :"; Resp.Count
PRINT "Waveform decimal byte value   ="; ASC(MID$(Resp$,Resp.Count,1)
PRINT "SumCheck so far (MOD 256)     ="; SumCheck2% MOD 256
CLOSE: END
`
Create.CSV:
    `
    `*****
    `***** Convert the total Response string to file Wave.CSV
    `***** as input file for Excel (spreadsheet), for example.
    `*****
    `
OPEN "Wave.CSV" FOR OUTPUT AS #4
    PRINT #4, "Title        , ";
    IF MID$(Query$, 4, 2) = "10" THEN
        PRINT #4, "Input A"
    ELSEIF MID$(Query$, 4, 2) = "11" THEN
        PRINT #4, "TrendPlot Reading 1"
    END IF
    IF Trace.Result = 0 OR Trace.Result = 1 THEN
        PRINT #4, "ID           ,"; Trace.Result  `Acquisition trace
        PRINT #4, "Type         , "; "Acquisition trace"
    ELSEIF Trace.Result = 2 THEN
        PRINT #4, "ID           ,"; 2          `TrendPlot trace
        PRINT #4, "Type         , "; "TrendPlot trace"
    END IF
    PRINT #4, "Date         , "; Month$+"/"+Day$+"/"+MID$(Year$,3,2)
    PRINT #4, "Time         , "; Hours$+":"Minutes$+":"+Seconds$
    `
    `***** X.Scale = time per division (over 10 divisions)
    PRINT #4, "X Scale      ,"; X.Scale
    PRINT #4, "X At 0%      ,"; X.Zero
    PRINT #4, "X Resolution ,"; X.Resol
    PRINT #4, "X Size       ,"; Nbr.Of.Samples
    PRINT #4, "X Unit       , ";
    IF X.Unit =  7 THEN PRINT #4, "s"
    IF X.Unit = 10 THEN PRINT #4, "Hz"
    PRINT #4, "X Label      ,";
    IF X.Unit =  7 THEN PRINT #4, X.Scale; "s/Div"
    IF X.Unit = 10 THEN PRINT #4, X.Scale; "Hz/Div"
    `
    PRINT #4, "Y Scale      ,"; Y.Scale
    PRINT #4, "Y At 50%     ,"; Y.Zero
    PRINT #4, "Y Resolution ,"; Y.Resol
    PRINT #4, "Y Size       ,";
    IF Sample.Bytes = 1 THEN            `1-byte samples
        PRINT #4, 256
    END IF                              `Range = 256
    IF Sample.Bytes = 2 THEN            `2-byte samples
        PRINT #4, 65536
    END IF                              `Range = 256*256
    PRINT #4, "Y Unit       , ";
    IF Y.Unit = 1 THEN PRINT #4, "V"
    IF Y.Unit = 2 THEN PRINT #4, "A"
    IF Y.Unit = 3 THEN PRINT #4, "Ohm"
    PRINT #4, "Y Label      ,";
    IF Y.Unit = 1 THEN PRINT #4, Y.Scale; "V/Div"
    IF Y.Unit = 2 THEN PRINT #4, Y.Scale; "A/Div"
    IF Y.Unit = 3 THEN PRINT #4, Y.Scale; "Ohm/Div"
`
```

```
    PRINT #4,
'
'
    '***** Sample values x,y (time,amplitude)
    Time.Value = X.Zero                 'Start at x-offset
    MinMax.Flag = MinMax.Samples        'Switch flag (2, 1, 0)
        FOR i = 1 TO Nbr.Of.Samples
            IF (Signed.Samples = 0) OR (Sample.Value&(i) < CLimit) THEN
                                        'Positive value
                Amplit.Value = Sample.Value&(i) * Y.Resol
            ELSE
                                        'Negative value
                Amplit.Value = - ((CMaxim - Sample.Value&(i)) * Y.Resol)
            END IF
            IF MinMax.Samples = 2 THEN  'Min/Max/Ave waveform
                IF MinMax.Flag = 2 THEN
                    MinMax.Flag = MinMax.Flag - 1
                    PRINT #4, Time.Value; ","; Amplit.Value; ",";
                ELSEIF MinMax.Flag = 1 THEN MinMax.Flag = MinMax.Flag - 1
                    PRINT #4, Amplit.Value; ",";
                ELSE
                    MinMax.Flag = 2
                    PRINT #4, Amplit.Value
                    Time.Value = Time.Value + X.Resol
                END IF
            END IF
            IF MinMax.Samples = 1 THEN  'Min/Max waveform
                IF MinMax.Flag = 1 THEN
                    MinMax.Flag = 0
                    PRINT #4, Time.Value; ","; Amplit.Value; ",";
                ELSE
                    MinMax.Flag = 1
                    PRINT #4, Amplit.Value
                    Time.Value = Time.Value + X.Resol
                END IF
            END IF
        IF MinMax.Samples = 0 THEN      'Single waveform
            PRINT #4, Time.Value; ","; Amplit.Value
            Time.Value = Time.Value + X.Resol
        END IF
    NEXT i
CLOSE #4: RETURN
'
'********************  End of example program  *******************
```