REMOTE CONTROL AND PROGRAMMING REFERENCE

for the FLUKE 190 family

of ScopeMeter test tools

===

This file contains remote control and programming information
for the above-mentioned models with use of the PM9080
Optically Isolated RS232 Adapter/Cable.

It consists of the following chapters:

1. INSTALLING THE PM9080

2. INTRODUCTION TO PROGRAMMING

3. COMMAND REFERENCE

APPENDIXES

1.   INSTALLING THE PM9080

- 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 to the ScopeMeter.

- Turn on the computer and the ScopeMeter.

- Make sure that the communication settings match for the
  RS232 port of the computer and the ScopeMeter.

  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.

==

2.   INTRODUCTION TO PROGRAMMING

        ** Basic Programming Information **

When you have installed the PM9080 as described in the
previous chapter, you can control the ScopeMeter
from the computer with simple communication facilities, such
as GWBASIC, QuickBASIC and QBASIC (programming languages from
Microsoft Corporation).

All examples given in this manual are in the QBASIC language
but will also run in QuickBASIC. QuickBASIC allows you to
make executable files from programs so you can start such
programs directly from DOS.
It is assumed that you have knowledge of these programming
languages. QBASIC is supplied with Microsoft MS-DOS 5.0 and
higher and Windows 95, 98, and NT, including 'on-line' Help.

Features of the syntax and protocol for the ScopeMeter
are as follows:

- Easy input format with a 'forgiving' syntax:
     All commands consist of two characters that can be
     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 D).
     Responses always end with the carriage return code
     (ASCII 13). Because the carriage return code is a
     non-visible character (not visible on the screen or on
     paper), this character is represented as <cr> in the
     command syntax.

- Synchronization between input and output:
     After receipt of every command, the ScopeMeter
     returns an acknowledge character (digit) 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.

** Commands sent to the ScopeMeter **

All commands for the ScopeMeter consist of a header
made up of two alpha characters sometimes followed by
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 received from the ScopeMeter **

After each command sent to the ScopeMeter there
is an automatic response from it, indicated as <acknowledge>
(which you MUST input), to let the computer know whether or
not the received 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 has received a query, it sends
the <acknowledge> reply as it does after any command, but
now it is followed by the queried response data.

The format of the response data depends upon which query is
sent. When a response consists of different response data
portions, these are separated with commas (ASCII code 44).
Also refer to the section 'Data Separators'.

All response data, <acknowledge> as well as following
(queried) response data are terminated with the carriage
return code (<cr> = ASCII 13). Also refer to the section
'Command and Response Terminators'.

** Acknowledge **

After receiving of 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 queried response data
terminated with <cr>.
If <acknowledge> is 1 or higher, it indicates that the
ScopeMeter has not executed the command
successfully. In that case, 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 information see Appendix A.
In case of an error you can obtain more detailed status
information by using the ST (STATUS) query.

Note: YOU MUST ALWAYS INPUT <acknowledge>, EVEN WHEN
         THE COMMAND WAS NOT A QUERY.

** 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

Name	Character	ASCII Value Decimal	Comments
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 it
  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.)

  After <cr> is recognized by the ScopeMeter, the
  entered command is executed. After EACH command the
  ScopeMeter returns <acknowledge><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.

2. Output a command or query to the ScopeMeter.

3. Input the acknowledge response from the ScopeMeter.

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

   If the response value is non-zero, the ScopeMeter did not
   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 output to the ScopeMeter, input its
   response.

5. The sequence of points 2, 3, and 4 may be repeated for
   different commands or queries.

6. Close the communication channel.

Refer to the program example on the next page.

```
'Example of a typical program sequence:

'*****************  Begin 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 example program *****************
```

===

3.   COMMAND REFERENCE

CONVENTIONS

          ** Page layout used for each command **

- Header

  Each command description starts on a new page with a header
  for quickly finding the command. This header indicates the
  command name and the two-character header used for the
  command syntax. Example:

  ===
              AUTO SETUP   AS

  Where   AUTO SETUP is a descriptive name for the command
  (this is no syntax!),

  and   AS   are the first two characters used for
                      the command syntax (not the complete
                      syntax).

- 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 <acknowledge>
  response, followed by the query response if the syntax
  relates to a query.

- 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 these commands.
  The following two comment lines (start with ') successively
  indicate the beginning and the end of an example program.

  '***************** Begin example program *****************

  '***************** End example program *****************

Use an MS-DOS Editor and copy the complete program between
these two lines to a file name with the .BAS extension.
Start QBASIC and open this file from the FILE menu.
Long programs (longer than 55 lines) include page breaks.
Such page breaks are preceded by the ' (remark) character
to prevent the QBASIC interpreter from interpreting them as
an incorrect statement.
When you have connected the ScopeMeter, you can start
the program from the RUN menu.

** 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 insert the brackets in the command!

[...]   The item between these brackets is optional.
                 This means that you may omit it for the
                 command, or for a response it may not appear.
                 Do not insert the brackets in the command!

  | This is a separator between selectable items.
                 This means that you must choose only one of
                 the items (exclusive or).

{...}   Specifies an element that may be repeated 0 or
                 more instances.

(...)   Grouping of multiple elements.

<binary_character>= 0 to 255

<digit> =   0 to 9

<sign> =   + | -

<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

==

** Overview of commands for the ScopeMeter **

```
                          COMMAND      PAGE
COMMAND NAME              HEADER       NUMBER
----------------------------------------------------------
AUTO SETUP               AS           3.5
ARM TRIGGER              AT           3.7
CLEAR MEMORY             CM           3.9
CPL VERSION QUERY        CV           3.11
DEFAULT SETUP            DS           3.13
GET DOWN                 GD           3.15
GO TO LOCAL              GL           3.17
GO TO REMOTE             GR           3.20
HOLD                     HO           3.21
IDENTIFICATION           ID           3.23
INSTRUMENT STATUS        IS           3.25
PROGRAM COMMUNICATION    PC           3.28
PROGRAM SETUP            PS           3.30
QUERY MEASUREMENT        QM           3.34
QUERY PRINT              QP           3.38
QUERY SETUP              QS           3.42
QUERY WAVEFORM           QW           3.43
READ DATE                RD           3.59
RESET INSTRUMENT         RI           3.61
REPLAY                   RP           3.63
RECALL SETUP             RS           3.65
READ TIME                RT           3.68
SWITCH ON                SO           3.70
SAVE SETUP               SS           3.71
STATUS QUERY             ST           3.72
TRIGGER ACQUISITION      TA           3.75
WRITE DATE               WD           3.77
WRITE TIME               WT           3.79
```

==
                    AUTO SETUP   AS

Purpose:

    Invokes an automatic setup for the active mode. The result
    of this command is the same as pressing the AUTO key
    on the ScopeMeter.

    Note: You can select the items that are affected by the
            AUTO SET procedure via the USER OPTIONS key on
            the ScopeMeter.

Command Syntax:

    AS<cr>

Response Syntax:

    <acknowledge><cr>

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.

```
'*****************  Begin 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 example program  *****************
```

===
                    ARM TRIGGER   AT

    Purpose:

       Resets and arms the trigger system for a new acquisition.
       This command is used for single shot measurements.
       When the AT command is given while an acquisition is in
       progress, this acquisition is aborted and the trigger
       system is rearmed.

    Command Syntax:

       AT<cr>

    Response Syntax:

       <acknowledge><cr>

    Example:

    The following example program arms the trigger system of
    the ScopeMeter with the AT command.
    This means that after this command the ScopeMeter starts an
    acquisition when a trigger occurs from the signal (when
    exceeding the trigger level) or from a TA (Trigger
    Acquisition) command.
    After the AT command it is assumed that the signal amplitude
    is sufficient to trigger the acquisition. If it is not, you
    can use the TA (TRIGGER ACQUISITION) command to force the
    acquisition to be triggered. But this is not useful if you
    want the acquisition to be started on a signal edge for
    synchronization purposes.

    Also see the example program for the IS command, which also
    uses the AT command for a single shot application.

    '**************** Begin 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 example program *****************
```

===
                    CLEAR MEMORY   CM

Purpose:

    Clears all saved setups, waveforms, and screens from
    memory.

Command Syntax:

    CM<cr>

Response Syntax:

    <acknowledge><cr>

Example:

```
'****************  Begin 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 example program  *****************
```

```
===============================================================
                  CPL VERSION QUERY                 CV
---------------------------------------------------------------
```

Purpose:

   Queries the CPL interface version.

Command Syntax:

   CV<cr>

Response Syntax:

   <acknowledge><cr>[<version><cr>]

   where,

   <version>   is an ASCII string representing the year this
                version has been created.

Example:

```
'*****************  Begin example program  *****************

OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1,"CV"              'Sends CPL VERSION query.
GOSUB Acknowledge         'Input acknowledge from ScopeMeter.
INPUT #1,VERSION$         'Inputs queried data.
PRINT "CPL Version "; VERSION$    'Displays version data.
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 example program  *****************
```

==
                    DEFAULT SETUP   DS

Purpose:

    Resets the ScopeMeter to the factory settings at
    delivery, except for the RS232 communication settings such
    as baud rate, to keep the communication alive.
    A Master Reset (refer to the Users Manual) performs the
    same, but also resets the RS232 communication settings to
    the default values.

Command Syntax:

    DS<cr>

Response Syntax:

    <acknowledge><cr>

    Note: Wait for at least 2 seconds after the
               <acknowledge> reply has been received, to let
               the ScopeMeter settle itself before you send the
               next command.

Example:

```
'                         Page 3.14

'*****************  Begin 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 s) 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 example program  *****************
```

===
                    GET DOWN   GD

Purpose:

   Switches the instrument's power off. If a power adapter
   is connected, you can use the SO command to switch power
   on again. If there is no power adapter connected, the
   instrument can only be switched on manually by pressing
   the Power ON/OFF key.

Command Syntax:

   GD<cr>

Response Syntax:

   <acknowledge><cr>

Example:

```
'*****************  Begin 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 example program  *****************
```

```
================================================================
                GO TO LOCAL                      GL
----------------------------------------------------------------
```

Purpose:

    Sets the ScopeMeter in the local operation mode
    so the keypad is enabled.
    Also refer to the GR (Go to Remote) command.

Command Syntax:

    GL<cr>

Response Syntax:

    <acknowledge><cr>

Example:

The following example uses the GR (GO TO REMOTE) command
(refer to the description for this command) to set the
ScopeMeter in the REMOTE state so that the keypad
is disabled. After that, the GL (GO TO LOCAL) command
is sent so that the keypad is enabled again.

```
'                          Page 3.18

'*****************  Begin 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 example program  ******************
```

```
===============================================================
                GO TO REMOTE                    GR
---------------------------------------------------------------
```

Purpose:

    Sets the ScopeMeter in the remote operation mode
    so that the keypad is disabled.
    You can use the following methods to return to the
    local operation mode so that the keypad is enabled:
    1. Sending the GL (Go to Local) command.

Command Syntax:

    GR<cr>

Response Syntax:

    <acknowledge><cr>

See an example for this command under GO TO LOCAL (GL).

```
===============================================================
                HOLD                              HO
---------------------------------------------------------------
```

Purpose:

   Sets the ScopeMeter in the Hold mode. In other words, the
   ScopeMeter stops sampling the input channels and
   calculating measurement results.

Command Syntax:

   HO<cr>

Response Syntax:

   <acknowledge><cr>

Example:

```
'****************  Begin 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 example program  ******************
```

===
                    IDENTIFICATION   ID

Purpose:

    Returns the ScopeMeter model identification information.

Command Syntax:

    ID<cr>

Response Syntax:

    <acknowledge><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.

```
'*****************  Begin 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 example program  *****************
```

===
                    INSTRUMENT STATUS   IS

Purpose:

   Queries the contents of the ScopeMeter's status register.
   The returned value reflects the present operational status
   of the ScopeMeter. This is a 16-bit word, presented as an
   integer value, where each bit represents the Boolean value
   of a related event.

Command Syntax:

   IS<cr>

Response Syntax:

   <acknowledge><cr>[<status><cr>]

   where,

   <status> =   integer value 0 to 65535

   <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 (Net) Adapter connected
    7   128   Calibration necessary
    8   256   Instrument Held (HOLD status)
    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> value available

Example:

```
'****************  Begin 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 example program  ******************
```

```
==============================================================
                PROGRAM COMMUNICATION          PC
--------------------------------------------------------------
```

Purpose:

   Programs the baud rate for RS232 communication:

Command Syntax:

   PC <baudrate>

   where,

   <baudrate>   =   1200|2400|4800|9600|19200
                    38400 (Fluke 19xC)
                    57600 (Fluke 19xC, PM9080/101 required)

   The default baudrate is 1200. This is set at power-on or
   after a Reset Instrument command (command "RI")

   Notes:
        The Fluke 19x/19xC instruments support 1 stopbit,
        8 databits and software handshake (X-on X-off
        protocol).
        Hardware handshaking is not supported.

Response Syntax:

    <acknowledge><cr>

See an example for this command under QUERY PRINT (QP).

==
                    PROGRAM SETUP   PS

Purpose:

    Restores a complete setup, previously saved with the SS
    (Save Setup) command and queried with the QS (Query Setup)
    command and saved in a string variable or to 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:

    <acknowledge><cr>

Command Syntax 2:

    <queried_setup><cr>

    <queried_setup> = The data returned with the QS command.
                       (omit the <acknowledge><cr> response).

Response Syntax 2:

    <acknowledge><cr>

    Note: Wait for at least two seconds after the
              <acknowledge> reply has been received, to let
              the ScopeMeter settle itself before you send the
              next command.

Remarks:

    The ScopeMeter sends the <acknowledge> 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.
    If you do not follow this rule, the ScopeMeter
    may crash. A Reset may then be necessary to recover
    the ScopeMeter. (Refer to the ScopeMeter Users Manual.)

Example:

The following example program demonstrates the use of the
QS (QUERY SETUP) and the PS (PROGRAM SETUP) commands.
The present setup is queried from ScopeMeter and saved to
file. The program asks you to change the ScopeMeter settings.
Then the original setup is read from file and sent back

to the ScopeMeter.

```
'*****************  Begin example program  *****************

OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
CLS
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 are 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."
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 example program ******************
```

```
================================================================
                QUERY MEASUREMENT              QM
----------------------------------------------------------------
```

Purpose:

   Queries for active readings (see Syntax 1) or measurement
   results from the ScopeMeter (see Syntax 2).

Command Syntax 1:

   QM<cr>

Command Syntax 2:

   QM <no>{,<no>}<cr>

```
      where in TrendPlot mode, *************************
      <no> = 11 | 21                                  *
                                                      *
      where in Meter mode, ************************** *
      <no> = 11 | 19                               * *
                                                  * *
      where in Scope mode, ************************ * *
      <no> = 11 | 21 | 31 | 41 |                 * * *
             53 | 54 | 55 | 61 | 71              * * *
                                                 * * *
   <no>  MEASUREMENT TYPE / DESCRIPTION          * * *
                                                 * * *
    11   Measurement reading 1                    |*|  |*|
         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)   |*| | |
    71   Cursor relative time value (delta T)      |*| | |
```

Notes: - Maximum 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.

Response Syntax 1:

    <acknowledge><cr>[<reading>{,<reading>}<cr>]

    where,

    <reading> = <no>,<valid>,<source>,<unit>,<type>,<pres>,<resol>
       <no>   see Command Syntax 2
       <valid> validity of the reading:
                   1 reading valid
                   0 reading non-valid
       <source> source of the reading:
                   1 Voltage channel: Input A (Scope mode)
                   2 Ampere channel: Input B (Scope mode)
                   3 Input external: COM & V/Ohm/Diode (Meter mode)
                  12 Input_AB (Phase A over B): A>B (Scope mode),
                      or M (Mathematics A+B, A-B or AxB)
                  21 Input_BA (Phase B over A): B>A (Scope mode)
       <unit>   unit of the reading:
                   0 None (off)
                   1 Volt
                   2 Ampere
                   3 Ohm
                   4 Watt
                   5 Farad
                   6 Kelvin
                   7 seconds
                   8 hours
                   9 days
                  10 Hertz
                  11 Degrees
                  12 Celsius
                  13 Fahrenheit
                  14 percentage (%)
                  15 dBm 50 Ohm
                  16 dBm 600 Ohm
                  17 dBVolt
                  18 dBAmpere
                  19 dBWatt
                  20 Volt * Ampere Reactive (VAR)
                  21 Volt * Ampere (VA)
       <type>   reading characteristic of the measurement:
                   0 None
                   1 Mean
                   2 Rms
                   3 True rms
                   4 Peak peak
                   5 Peak maximum
                   6 Peak minimum
                   7 Crest factor
                   8 Period
                   9 Duty cycle negative
                  10 Duty cycle positive
                  11 Frequency
                  12 Pulse width negative

```
                 13   Pulse width positive
                 14   Phase
                 15   Diode
                 16   Continuity
                 18   Reactive Power
                 19   Apparent Power
                 20   Real Power
                 21   Harmonic Reactive Power
                 22   Harmonic Apparent Power
                 23   Harmonic Real Power
                 24   Harmonic rms
                 25   Displacement Power Factor
                 26   Total Power Factor
                 27   Total Harmonic Distortion
                 28   Total Harmonic Distortion with respect
                      to Fundamental
                 29   K Factor (European definition)
                 30   K Factor (US definition)
                 31   Line Frequency
                 32   Vac PWM or Vac+dc PWM
                 33   Rise time
                 34   Fall time
        <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
```

Response Syntax 2:

    <acknowledge><cr>[<meas_value>{,<meas_value>}<cr>]

    where,

    <meas_value> = [<sign>]<decimal_number>E<sign><decimal_number>

Notes: Only displayed results are available for output.
       Not all readings are available in all Fluke 19x/19xC
       models/versions.

    Example:

    '***************** Begin 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
'
```

'   Page 3.37
```

```
'****************  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 example program  ******************
```

```
===============================================================
                QUERY PRINT                      QP
---------------------------------------------------------------
```

Purpose:

   Queries a screen dump of the ScopeMeter in different
   printer formats. This allows you to make a copy of the
   ScopeMeter screen on paper. Format ratios:
     1 : 1 = width x height = 240 x 240
     4 : 3 = width x height = 320 x 240

Command Syntax:

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

   where,

   <screen_number> = 0   Always zero

   <output_format> = 0   Epson FX, LQ compatible
                        Returns screen image 1:1 (Fluke 19x)
                        Returns screen image 4:3 (Fluke 19xC)
              1   Laser Jet
                        Returns screen image 4:3 (Fluke 19x)
              2   Desk Jet
                        Returns screen image 4:3 (Fluke 19x)
              3   PostScript
                        Returns screen image 4:3 (Fluke 19x)
             11   PNG format (<block_transfer> mandatory)
                        Returns screen image 4:3 (Fluke 19xC)

   <block_transfer>= b   binary format
                   B   Binary format

   Note: Sending QP without arguments returns the screen image
         in Epson format (i.e., this command is equivalent to
         QP 0,0).

Response Syntax for QP or QP 0,0 or QP 0,1 or QP 0,2 or QP 0,3:

   <acknowledge><cr>[<printer_data>]

   <printer_data>   This data can be sent directly to the printer
                   to get a hard copy of the screen on paper.

Example for QP 0,0 (or QP or QP 0,1 or QP 0,2 or QP 0,3):

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).

```
'*****************  Begin 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
                          '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 with 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
      CLOSE                    'Close all files.
      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 example program ******************
```

Response Syntax for QP 0,11,b or QP 0,11,B:

    <acknowledge><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>   = <acknowledge><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>.

The <png_data> is sent in blocks (segments). When the <block_data>
parts of all '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: Request the next segment.
                            1   Retransmit: Request retransmission
                                 of the just transferred segment.
                            2   Terminate: Abort block transfer for
                                 this QP command.

The PNG format is specified in: "PNG (Portable Network Graphics)
Specification, Version 1.2", 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 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 1200 (default baud rate).


```
'***************** Begin 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 with 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
      CLOSE                 'Close all files.
      PRINT "Program aborted."
      END
   END IF
   RETURN

   '***************** End example program *****************
```

```
 ============================================================
                QUERY SETUP                       QS
 ------------------------------------------------------------
```

 Purpose:

    Queries the present acquisition setup data from the
    ScopeMeter.

Command Syntax:

    QS [<setup_no>]<cr>

    where,

    <saved_setup_no> = 0   : Actual setup

Response Syntax:

    <acknowledge><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.

    Note: Also see the Program Setup (PS) command.

See an example for this command under PROGRAM SETUP (PS).

```
================================================================
                QUERY WAVEFORM                   QW
----------------------------------------------------------------
```

Purpose:

   Queries the trace data (administration and/or sample data)
   related to the waveform from the ScopeMeter.

   When a waveform is queried that is still under processing,
   the processing is finished first (no half traces returned).

Command Syntax:

   QW <trace_no>[,V|S]

   <trace_no> = <decimal number>

```
              <trace_no> Trace Source: (only for Fluke 19x)
              ----------------------------------------------
                 10      Scope mode:
                            Normal trace INPUT A
                            Min/Max trace INPUT A
                         Scope Record:
                            Min/Max trace INPUT A
                 11      TrendPlot 1:
                            Min/Max/Average trace
                 20      Scope mode:
                            Normal trace INPUT B
                            Min/Max trace INPUT B
                         Scope Record:
                            Min/Max trace INPUT B
                 21      TrendPlot 2:
                            Min/Max/Average trace
                 30      Scope mode, Mathematics
                         (not available in all versions):
                            Min/Max trace A+B, A-B or AxB

             <trace_no> Trace Source: (only for Fluke 19xC)
             -----------------------------------------------
                 10      Scope mode:
                            Normal trace INPUT A
                            Min/Max trace INPUT A
                         Scope Record:
                            Min/Max trace INPUT A
                 11      TrendPlot 1:
                            Min/Max/Average trace
                 12      Scope mode:
                            Min/Max trace INPUT A ENVELOPE
                 13      Scope mode:
                            Min/Max trace INPUT A REFERENCE
                 20      Scope mode:
                            Normal trace INPUT B
                            Min/Max trace INPUT B
                         Scope Record:
                            Min/Max trace INPUT B
```

21	TrendPlot 2: Min/Max/Average trace
22	Scope mode: Min/Max trace INPUT B ENVELOPE
23	Scope mode: Min/Max trace INPUT B REFERENCE
30	Scope mode, Mathematics: Min/Max trace A+B, A-B or AxB

```
V | v       Trace values (samples) only
S | s       Setup (administration) data only. When V or S is
            omitted, trace values and setup data are returned.
```

Response Syntax:

    <acknowledge><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 C.

```
    If option V or v (value only) is given:

    <trace_data> = <trace_samples><cr>

    For detailed descriptions about the waveform structure,
    refer to Appendix C.

    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:

'*****************  Begin 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 minimum and maximum 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
```

```
     '                      Page 3.45

     '**************** 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
'
```

```
'                               Page 3.47
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
```
,

```
'                           Page 3.48
'
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
,
```

```
'                         Page 3.49
'
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><exponen
    '***** <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
,
```

```
'                            Page 3.50
'
   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><exponen
'***** <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
```

```
'                         Page 3.51
'
     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
,
```

```
'                              Page 3.52
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
,
```

```
'                              Page 3.53
'
     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
,
```

```
'                           Page 3.54
'
     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
     '
```

```
'                              Page 3.55
'
     '***** <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
'
```

```
'                            Page 3.57
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,
'
```

```
'                         Page 3.58
'
     '***** 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 example program  *****************
```

==
                  READ DATE   RD

Purpose:

   Reads the real time clock date settings.

Command Syntax:

   RD<cr>

Response Syntax:

   <acknowledge><cr>[<date><cr>]

   where,

   <date> =   string of the following format:
                       <year>,<month>,<day>
                       e.g. 1999,8,14

Example:

The following example program reads the date setting from
the ScopeMeter.

```
'                        Page 3.60

'****************  Begin 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 example program  ******************
```

```
================================================================
                    RESET INSTRUMENT                 RI
----------------------------------------------------------------
```

Purpose:

   Resets the entire instrument, including the CPL interface.
   The baud rate remains unchanged.

Command Syntax:

   RI<cr>

Response Syntax:

   <acknowledge><cr>

   Note: Wait for at least 2 seconds after the
              <acknowledge> reply has been received, to let
              the ScopeMeter settle itself before you send the
              next command.

Example:

The following example 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.

```
'****************** Begin 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 example program ******************
```

```
===============================================================
                REPLAY                          RP
---------------------------------------------------------------
```

Purpose:

   To select and setup the Replay analysis mode and to select
   a replay screen (see Syntax 2) or to query the total
   number of valid replay screens (see Syntax 1).

   Note: applicable for the Fluke 199 and 196 families

Command Syntax 1:

   RP<cr>

Command Syntax 2:

   RP <screen_index><cr>

   where,

   <screen_index> = 0 to -99   : Replay screen number
                           0 = newest (current) screen
                      -99 = oldest screen

Response Syntax 1:

   <acknowledge><cr><nr_of_screens><screen_index><cr>

   where,

   <nr_of_screens> = 0 to 100   : number of valid screens
                        0 = no valid screens

   <screen_screen> = 0 to -99   : index of the actual screen

Response Syntax 2:

   <acknowledge><cr>

   As a result, the Replay function is started and the replay
   screen <screen_index> is shown on the instrument.

   Notes: - When <screen_index> is omitted, nothing happens.
         - Replaying screens works only in the SCOPE mode.

   Tips: - Use the QP, QS, QM, QW commands for information
          about the replayed screen and measurements.
         - Send the AT command to return to disable the Replay
          function and return to normal (running).

Example:

```
'                             Page 3.64

'***************** Begin 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 mode (running)
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; SCOPE mode selected?"
     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 example program  ******************
```

```
=============================================================
                  RECALL SETUP                     RS
-------------------------------------------------------------
```

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 to running state.

Command Syntax:

   RS <setup_reg><cr>

   where,

   <setup_reg> = 1 to 15   : Screen/Setup memories
               1001   : Long Record/Replay memory Input A
               1002   : Long Record/Replay memory Input B

Response Syntax:

   <acknowledge><cr>

   Note: The new setup is active when you have received the
        <acknowledge> response from the ScopeMeter.

Example:

The following example program saves the present setup in
setup memory 8. You are requested to change the present
settings. Then the original settings are recalled from
setup memory 8 and made the actual setting.

```
'                          Page 3.66

'*****************  Begin 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 are 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 example program  ******************
```

==
                    READ TIME   RT

Purpose:

   Reads the real time clock time settings.

Command Syntax:

   RT<cr>

Response Syntax:

   <acknowledge><cr>[<time><cr>]

   where,

   <time> =   string of the following format:
                     <hours>,<minutes>,<seconds>
                     e.g. 15,4,43

Example:

The following example program reads the time setting from
the ScopeMeter.

```
'*****************  Begin 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 example program  *****************
```

==
                   SWITCH ON   SO

Purpose:

   Switches the ScopeMeter on.
   This only works when the ScopeMeter is powered via the
   power adapter.

Command Syntax:

   SO<cr>

Response Syntax:

   <acknowledge><cr>

See an example for this command under GET DOWN (GD).

===
                  SAVE SETUP   SS

Purpose:

   Saves the present setup in one of the battery-backup
   instrument registers.

Command Syntax:

   SS <setup_reg><cr>

   where,

   <setup_reg> = 1 to 15   : Screen/Setup memories
                                When <setup_reg> is omitted,
                                number 1 is assumed.
                   1001   : Long Record/Replay memory Input A
                   1002   : Long Record/Replay memory Input B

Response Syntax:

   <acknowledge><cr>

See an example for this command under RECALL SETUP (RS).

==
                    STATUS QUERY   ST

Purpose:

   Queries the error status of the ScopeMeter.
   This is a 16-bit word, presented as an integer value,
   where each bit represents the Boolean value of a related
   error event. After the reply or after a RI (Reset
   Instrument) command, the value is reset to zero.
   A complete description of the status word is given in
   Appendix B.

Command Syntax:

   ST<cr>

Response Syntax:

   <acknowledge><cr>[<status>

   where,

   <status> =   integer value 0 to 32767

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).

```
'                           Page 3.73

'****************  Begin 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 + Status subroutine  ***********
'This subroutine inputs the acknowledge value from the
'ScopeMeter. If the acknowledge value is non-zero,
'the ST query is used to get further status information from
'the ScopeMeter with respect to the error.
'In case of an error the program is aborted.

Acknowledge.Status:
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 information if 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 example program *****************

==
                    TRIGGER ACQUISITION   TA

Purpose:

    Triggers an acquisition. This command acts as a
    hardware trigger to start a new acquisition.
    In SINGLE shot acquisition mode the trigger system
    must have been armed with the AT (Arm Trigger)
    command.

Command Syntax:

    TA<cr>

Response Syntax:

    <acknowledge><cr>

Example:

```
'                          Page 3.76

'***************** Begin 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 example program *****************
```

==
                    WRITE DATE   WD

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. 1999,9,14

Response Syntax:

    <acknowledge><cr>

Example:

The following example program programs the ScopeMeter
with a new date setting.

```
'                          Page 3.78

'****************  Begin 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 example program  *****************
```

===
                  WRITE TIME   WT

Purpose:

   Writes the real time clock time settings.

Command Syntax:

   WT <time><cr>

   where,

   <time> =   string of the following format:
                    <hours>,<minutes>,<seconds>
                    e.g. 15,30,0

Response Syntax:

   <acknowledge><cr>

Example:

The following example program programs the ScopeMeter
with a new time setting.

```
'***************** Begin 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 example program *****************
```

```
================================================================
                APPENDIX A          ACKNOWLEDGE DATA
----------------------------------------------------------------
```

The ScopeMeter returns an <acknowledge> reply after
each command or query. The value indicates correct or
incorrect operation. You always must read this reply to check
for the correct operation and to achieve synchronization
between your program and the RS232 interface of the
ScopeMeter.

```
        <acknowledge>
           VALUE            MEANING

            0               No Error
            1               Syntax Error (see Note)
            2               Execution Error (see Note)
            3               Synchronization Error
            4               Communication Error
```

Note: The ST query may give you additional information.

When the ScopeMeter detects an error during the
execution of a command, it sends the corresponding
<acknowledge> 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 <acknowledge> reply after each command or
     query.
  2. If this <acknowledge> 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 Communication Error.

```
===============================================================
                APPENDIX B          STATUS DATA
---------------------------------------------------------------
```

The Status word returned from the ST query gives you extra
information when you have received a non-zero <acknowledge>
reply.
The Status word is a 16-bit binary word where each bit set
true represents an error event with a decimal value
determined by the bit position. (See the following table.)

When more than one bit is set true in the status word, 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	<acknowledge> 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	

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.

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

   1. After the response (the status word) from the ST query
      has been read.
   2. After the RI (Reset Instrument) command.

===
                  APPENDIX C   WAVEFORM DATA

The waveform data that is received from 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>
  where,

  <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.

  <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>
      <month>=    <digit><digit>
      <day>  =    <digit><digit>

<time_stamp> =    <hours><minutes><seconds>
      <hours>=    <digit><digit>
      <minutes>= <digit><digit>
      <seconds>= <digit><digit>

<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. This means that the remote device will
         not have to do any polling on status bits before the
         query is sent. When the waveform that was queried for,
         is still under processing, the processing is finished
         first. So no "half traces" will be returned. When the
         waveform under processing is in roll mode, the query
         will give an execution error.
         The remote device has the possibility to cancel the
         query, when waiting for response takes too long. This
         can be achieved by sending an <esc> or hardware break.

```
=================================================================
                  APPENDIX D        ASCII CODES
-----------------------------------------------------------------


  Hexadecimal value
   | ASCII character
   |  |  Decimal value
   |  |  |
  00 NUL  0      20 SP 32      40 @  64      60 `   96
  01 SOH  1      21 !  33      41 A  65      61 a   97
  02 STX  2      22 "  34      42 B  66      62 b   98
  03 ETX  3      23 #  35      43 C  67      63 c   99
  04 EOT  4      24 $  36      44 D  68      64 d  100
  05 ENQ  5      25 %  37      45 E  69      65 e  101
  06 ACK  6      26 &  38      46 F  70      66 f  102
  07 BEL  7      27 '  39      47 G  71      67 g  103
  08  BS  8      28 (  40      48 H  72      68 h  104
  09  HT  9      29 )  41      49 I  73      69 i  105
  0A  LF 10      2A *  42      4A J  74      6A j  106
  0B  VT 11      2B +  43      4B K  75      6B k  107
  0C  FF 12      2C ,  44      4C L  76      6C l  108
  0D  CR 13      2D -  45      4D M  77      6D m  109
  0E  SO 14      2E .  46      4E N  78      6E n  110
  0F  SI 15      2F /  47      4F O  79      6F o  111

  10 DLE 16      30 0  48      50 P  80      70 p  112
  11 XON 17      31 1  49      51 Q  81      71 q  113
  12 DC2 18      32 2  50      52 R  82      72 r  114
  13 XOF 19      33 3  51      53 S  83      73 s  115
  14 DC4 20      34 4  52      54 T  84      74 t  116
  15 NAK 21      35 5  53      55 U  85      75 u  117
  16 SYN 22      36 6  54      56 V  86      76 v  118
  17 ETB 23      37 7  55      57 W  87      77 w  119
  18 CAN 24      38 8  56      58 X  88      78 x  120
  19  EM 25      39 9  57      59 Y  89      79 y  121
  1A SUB 26      3A :  58      5A Z  90      7A z  122
  1B ESC 27      3B ;  59      5B [  91      7B {  123
  1C  FS 28      3C <  60      5C \  92      7C |  124
  1D  GS 29      3D =  61      5D ]  93      7D }  125
  1E  RS 30      3E >  62      5E ^  94      7E ~  126
  1F  US 31      3F ?  63      5F _  95      7F    127
```

```
Hexadecimal value
 | ASCII character
 |  | Decimal value
 |  |  |
80  ? 128        A0    160      C0  À 192      E0  à 224
81    129        A1  ¡ 161      C1  Á 193      E1  á 225
82  , 130        A2  ¢ 162      C2  Â 194      E2  â 226
83  ƒ 131        A3  £ 163      C3  Ã 195      E3  ã 227
84  „ 132        A4  ¤ 164      C4  Ä 196      E4  ä 228
85  … 133        A5  ¥ 165      C5  Å 197      E5  å 229
86  † 134        A6  ¦ 166      C6  Æ 198      E6  æ 230
87  ‡ 135        A7  § 167      C7  Ç 199      E7  ç 231
88  ^ 136        A8  ¨ 168      C8  È 200      E8  è 232
89  ‰ 137        A9  © 169      C9  É 201      E9  é 233
8A  Š 138        AA  ª 170      CA  Ê 202      EA  ê 234
8B  ‹ 139        AB  « 171      CB  Ë 203      EB  ë 235
8C  Œ 140        AC  ¬ 172      CC  Ì 204      EC  ì 236
8D    141        AD  - 173      CD  Í 205      ED  í 237
8E  ? 142        AE  ® 174      CE  Î 206      EE  î 238
8F    143        AF  ¯ 175      CF  Ï 207      EF  ï 239
90    144        B0  ° 176      D0  Ð 208      F0  ð 240
91  ' 145        B1  ± 177      D1  Ñ 209      F1  ñ 241
92  ' 146        B2  ² 178      D2  Ò 210      F2  ò 242
93  " 147        B3  ³ 179      D3  Ó 211      F3  ó 243
94  " 148        B4  ´ 180      D4  Ô 212      F4  ô 244
95  • 149        B5  µ 181      D5  Õ 213      F5  õ 245
96  – 150        B6  ¶ 182      D6  Ö 214      F6  ö 246
97  — 151        B7  · 183      D7  × 215      F7  ÷ 247
98  ~ 152        B8  , 184      D8  Ø 216      F8  ø 248
99  ™ 153        B9  ¹ 185      D9  Ù 217      F9  ù 249
9A  š 154        BA  ° 186      DA  Ú 218      FA  ú 250
9B  › 155        BB  » 187      DB  Û 219      FB  û 251
9C  œ 156        BC  ¼ 188      DC  Ü 220      FC  ü 252
9D    157        BD  ½ 189      DD  Ý 221      FD  ý 253
9E  ? 158        BE  ¾ 190      DE  Þ 222      FE  þ 254
9F  Ÿ 159        BF  ¿ 191      DF  ß 223      FF    255
```