

REMOTE CONTROL AND PROGRAMMING REFERENCE

for the FLUKE 43 family

Power Quality Analyzers

=====

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

APPENDIX A	ACKNOWLEDGE DATA
APPENDIX B	STATUS DATA
APPENDIX C	WAVEFORM DATA
APPENDIX D	ASCII CODES

=====

1. INSTALLATION OF 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 Power Quality Analyzer.
- Turn on the computer and the Power Quality Analyzer.
- Make sure that the communication settings match for the RS232 port of the computer and the Power Quality Analyzer.

After power-on, the default settings of the Power Quality Analyzer 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 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 Power Quality Analyzer. 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 Power Quality Analyzer. 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 Power Quality Analyzer 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 Power Quality Analyzer 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, a Power Quality Analyzer 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 Power Quality Analyzer.

**** Commands sent to the Power Quality Analyzer ****

All commands for the Power Quality Analyzer 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 Power Quality Analyzer.

Some of the commands are followed by one or more parameters to give the Power Quality Analyzer 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 Power Quality Analyzer 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 Power Quality Analyzer 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 Power Quality Analyzer ****

After each command sent to the Power Quality Analyzer 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 Power Quality Analyzer 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 Power Quality Analyzer.

When the Power Quality Analyzer 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 Power Quality Analyzer 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 Power Quality Analyzer 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 Power Quality Analyzer 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 instrument and between values and strings received from the instrument. 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 instrument 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 instrument 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 instrument, the entered command is executed. After EACH command the instrument 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 instrument 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 instrument settings.
2. Output a command or query to the instrument.
3. Input the acknowledge response from the instrument.

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

If the response value is non-zero, the instrument 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 instrument, 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
'instrument power-on default settings. It also opens a
'communication channel (assigned #1) for input or output
'through the COM1 port. Your instrument 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 instrument.

PRINT #1, "ID"

'Outputs the IDENTITY command (query) to the instrument.

GOSUB Acknowledge

'This subroutine inputs the acknowledge response from
'the instrument 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  
'instrument. This routine inputs the acknowledge  
'response from the instrument. If the response is non-zero,  
'the previous command was not correct or was not correctly  
'received by the instrument. Then an error message is  
'displayed and the program is aborted.
```

Acknowledge:

```
INPUT #1, ACK          'Reads acknowledge from instrument.  
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 instrument. 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 Power Quality Analyzer as indicated in the PM9080 Instruction Manual, 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 Power Quality Analyzer **

COMMAND NAME	COMMAND HEADER	PAGE 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 HARMONICS	QH	3.34
QUERY MEASUREMENT	QM	3.36
QUERY PRINT	QP	3.40
QUERY SETUP	QS	3.44
QUERY WAVEFORM	QW	3.45
READ DATE	RD	3.62
RESET INSTRUMENT	RI	3.64
RECALL SETUP	RS	3.66
READ TIME	RT	3.69
SWITCH ON	SO	3.71
SAVE SETUP	SS	3.72
STATUS QUERY	ST	3.73
TRIGGER ACQUISITION	TA	3.76
WRITE DATE	WD	3.78
WRITE TIME	WT	3.80

=====

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 instrument.

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

Command Syntax:

AS<cr>

Response Syntax:

<acknowledge><cr>

Example:

The following example program sends an AUTO SETUP command to the instrument. Connect a repetitive signal on INPUT 1 to see the effect of AUTO SETUP.


```
'
                                     Page 3.6

'***** 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 instrument.
CLOSE #1
END

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

Acknowledge:
INPUT #1, ACK                     'Reads acknowledge from instrument.
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 instrument with the AT command. This means that after this command the instrument 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 instrument.
CLOSE #1
END
```

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

Acknowledge:

```
INPUT #1, ACK          'Reads acknowledge from instrument.  
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 instrument.
CLOSE #1
END

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

Acknowledge:
INPUT #1, ACK          'Reads acknowledge from instrument.
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:

```
'
                                     Page 3.12

'***** 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 instrument.
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
'instrument. This routine inputs the acknowledge
'response from the instrument. If the response is non-zero,
'the previous command was not correct or was not correctly
'received by the instrument. Then an error message is
'displayed and the program is aborted.

Acknowledge:
INPUT #1, ACK          'Reads acknowledge from instrument.
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 instrument 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 instrument settle itself before you send the next command.

Example:


```

'
'***** 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 instrument.
SLEEP 2                'Delay (2 s) necessary after "DS".
PRINT #1, "ID"         'Sends the IDENTIFICATION query.
GOSUB Acknowledge      'Input acknowledge from instrument.
INPUT #1, ID$          'Inputs identity data from instrument.
PRINT ID$              'Displays identity data.
CLOSE #1
END

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

Acknowledge:
INPUT #1, ACK          'Reads acknowledge from instrument.
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:

```
'
                                     Page 3.16

'***** 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 instrument.
PRINT "The GET DOWN command switched the instrument off."
PRINT "Press any key on the PC keyboard to switch "
PRINT "the instrument on again."
SLEEP
PRINT #1, "SO"          'Sends the SWITCH ON command.
GOSUB Acknowledge       'Input acknowledge from instrument.
CLOSE #1
END

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

Acknowledge:
INPUT #1, ACK           'Reads acknowledge from instrument.
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 instrument 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 instrument 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.

'***** 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 instrument.
PRINT "All instrument 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 instrument.
PRINT "The instrument 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  
'instrument. This routine inputs the acknowledge  
'response from the instrument. If the response is non-zero,  
'the previous command was not correct or was not correctly  
'received by the instrument. Then an error message is  
'displayed and the program is aborted.
```

Acknowledge:

```
INPUT #1, ACK          'Reads acknowledge from instrument.  
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 instrument 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 instrument in the Hold mode. In other words, the instrument 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 instrument.
PRINT "The HOLD command has put the instrument in HOLD."
PRINT "Check on the instrument 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 instrument.
CLOSE #1
END

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

Acknowledge:
INPUT #1, ACK           'Reads acknowledge from instrument.
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 instrument 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 instrument and displays this data on the PC screen.

```
'
                                     Page 3.24

'***** 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 instrument.
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
'instrument. This routine inputs the acknowledge
'response from the instrument. If the response is non-zero,
'the previous command was not correct or was not correctly
'received by the instrument. Then an error message is
'displayed and the program is aborted.

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from instrument.
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 instrument's status register. The returned value reflects the present operational status of the instrument. 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
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 instrument  
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 " instrument in Maintenance mode."  
IF (StV AND 2) = 2 THEN PRINT " instrument charging."  
IF (StV AND 4) = 4 THEN PRINT " instrument recording."  
IF (StV AND 8) = 8 THEN PRINT " AutoRanging active"  
IF (StV AND 16) = 16 THEN PRINT " instrument 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 " instrument 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 4096) = 4096 THEN PRINT " instrument triggered."  
IF (StV AND 8192) = 8192 THEN  
PRINT " instrument On."  
ELSE  
PRINT " instrument 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  
'instrument. This routine inputs the acknowledge  
'response from the instrument. If the response is non-zero,  
'the previous command was not correct or was not correctly  
'received by the instrument. Then an error message is  
'displayed and the program is aborted.
```

Acknowledge:

```
INPUT #1, ACK          'Reads acknowledge from instrument.  
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 *****
```

Purpose:

Programs the baud rate for RS232 communication:

Command Syntax:

PC <baudrate>

where,

<baudrate> = 1200|2400|4800|9600|19200 (guaranteed)

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

Notes:

The Fluke 43(B) supports 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 Syntax 1:

```
PS [<saved_setup_no>]<cr>
```

where,

```
<saved_setup_no> = 0 to 10
```

This is the register number where a setup is stored. Also see the description of the Save Setup (SS) command (0 is 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 instrument settle itself before you send the next command.

Remarks:

The instrument 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 instrument may crash. A Reset may then be necessary to recover the instrument. (Refer to the instrument 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 instrument and saved to file. The program asks you to change the instrument settings. Then the original setup is read from file and sent back to the instrument.

```
'
***** 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 instrument.
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 instrument."
PRINT "To see if this works, change the present settings and"
PRINT "verify if the instrument 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 instrument.
GOSUB Acknowledge    'Input acknowledge from instrument.
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 instrument 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 instrument.
END
```

```

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

```

Acknowledge:

```

INPUT #1, ACK           'Reads acknowledge from instrument.
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 instrument 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 HARMONICS
```

```
QH
-----
```

Purpose: (for the Fluke 43B only)

Queries the Volt/Amp/Watt harmonics data (administration and sample data) from the instrument.

Command Syntax:

```
QH [<harmonics_no>]
```

where,

```
<harmonics_no> = <byte 1><byte 2>
```

```
<byte 1> = ASCII "1" or "2" for channel 1 or 2
          ASCII "3" for channel 1 and 2
```

```
<byte 2> = ASCII "0"/"1"/"2" for Volt/Ampere/Watt harmonics
```

Possible values are:

```
QH 10 : Volt harmonics using channel 1
QH 21 : Ampere harmonics using channel 2
QH 32 : Watt harmonics using channel 1 and 2
```

If QH command is sent without arguments, the active <harmonics_no> is returned.

If QH command is sent with <harmonics_no>, the Response Syntax is:

```
<acknowledge><cr><harm_admin>,<harm_samples><cr>
```

where,

```
<harm_admin> = #0<block_header><block_length><status>
               <amplitude_unit><phase_unit>
               <amplitude_resolution><phase_resolution>
               <fundamental_frequency><date_stamp>
               <time_stamp><checksum>
```

```
<block_header> = <binary_character>
                 Value is 128.
```

```
<block_length> = <unsigned_integer>
                 This value gives the number of bytes that
                 are transmitted after the <block_length>
                 and before the <check_sum>.
```

```
<status> =      <unsigned_integer>
                 0 = Invalid, 1 = Valid
```

```
<amplitude_unit>= <unit>
                  <binary_character> representing the unit:
```

	None	=	0
	<Volt>	=	1
	<Ampere>	=	2
	<Watt>	=	4
<phase_unit> =	<unit>		
	<binary_character>	representing the unit:	
	None	=	0
	<Degree>	=	11

```

<amplitude_resolution> = <float>
                          This field contains the scale value
                          for all amplitude samples.

<phase_resolution> =    <float>
                          This field contains the scale value
                          for all phase samples (always 1).

<fundamental_frequency>= <float>
                          Frequency of the first harmonic, which
                          is the first non-DC spectrum component.

<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>.

<harm_samples> = #0<block_header><block_length><nbr_of_samples>
                 <samples><check_sum><cr>

<block_header>= <binary_character>
                 Value is 129.

<block_length>= <unsigned_integer>
                 This (2-bytes) value gives the number of bytes
                 that are transmitted after the <block_length>
                 and before the <check_sum>.

<nbr_of_samples>= <unsigned_integer>
                 Total number of harmonics samples that follow.

<samples> =      {<harm_amplitude><harm_phase>}

<harm_amplitude>= <float>

<harm_phase> = <signed integer>

<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>.

```

Example: The method for reading and analyzing harmonics data (QH) is similar for reading and analyzing waveforms (QW).

=====

QUERY MEASUREMENT

QM

Purpose:

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

Command Syntax 1:

QM<cr>

Command Syntax 2:

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

where in VOLTS/AMPS/HERTZ mode:

<no> = 11 : Voltage rms (ac + dc)
 21 : Ampere rms (ac + dc)
 31 : Line Frequency on trigger channel
 41 : Crest on channel 1 (Fluke 43B only)
 51 : Crest on channel 2 (Fluke 43B only)

where in POWER mode: (relative to %r or %f)

<no> = 11 : Real power
 21 : Apparent power
 31 : Reactive power
 41 : Total Power Factor (TPF) (Fluke 43B only)
 51 : Displacement Power Factor (DPF or cos phi)
 61 : Line Frequency

where in HARMONICS mode: (Volt/Ampere/Watt mode)

<no> = 11 : Total Harmonic Distortion (THD, not Watts mode)
 21 : Total rms reading
 31 : K-factor (not in Volts mode)
 41 : Frequency of selected harmonic component
 51 : Absolute rms of selected harmonic component
 61 : Relative rms (% of Total rms) of selected harmonic component
 71 : Phase of selected harmonic component

where in SAGS & SWELLS mode:

<no> = 11 : Voltage rms (ac + dc)
 21 : Ampere rms (ac + dc)

where in TRANSIENTS mode: (at cursor position of the selected transient)

<no> = 11 : Maximum Voltage peak
 21 : Minimum Voltage peak
 31 : Maximum Ampere peak
 41 : Minimum Ampere peak

where in TRANSIENTS Learn mode:

<no> = 11 : Voltage rms (ac + dc)
 21 : Line Frequency on Voltage channel

where in Motor INRUSH CURRENT mode:

<no> = 11 : Ampere amplitude at first cursor position
21 : Ampere amplitude at second cursor position

where in Meter OHMS/DIODE/CONTINUITY/CAPACITANCE
and TEMPERATURE mode:

<no> = 11 : Meter reading

where in SCOPE mode:

<no> = 11 : Reading of channel 1 (can be switched off)
21 : Reading of channel 2 (can be switched off)

- 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)
```

```
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 dB Volt
```

```
18 dB Ampere
```

```
19 dB Watt
```

```
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 AC average
<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>

Note: Only displayed results are available for output.

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 instrument.
'*** 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 instrument.
INPUT #1, result
PRINT "Measurement value   ="; result; "V"
CLOSE #1
END
```

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

Acknowledge:

```
INPUT #1, ACK           'Reads acknowledge from instrument.
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:
  LINE INPUT #1, dummy$
  WHILE LOC(1) > 0
    LINE INPUT #1, dummy$
  WEND
RETURN
```

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

=====

QUERY PRINT

QP

Purpose:

Queries a screen dump of the instrument in different printer formats. This allows you to make a copy of the instrument screen on paper.

Command Syntax:

QP <screen_number>,<output_format><cr>

where,

<screen_number> = 0 Always zero

<output_format> = 0 Epson FX, LQ compatible

 1 Laser Jet

 2 Desk Jet

 3 PostScript

Response Syntax:

<acknowledge><cr>[<printer_data>]

<printer_data>

This data can directly be sent to the printer to get a screen copy on paper.

Example:

The following program reads the instrument screen (print) data and copies this data to the file Qpfile. Hereafter, you can copy this file 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 instrument. The data transfer speed (baud rate) is set to 19200 and after the output it is set back 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 instrument power-on
                                     'defaults.
PRINT #1, "PC 19200"                 'Programs instrument to the maximum
                                     'baud rate.
GOSUB Acknowledge                   'Input acknowledge from instrument.
CLOSE #1
OPEN "COM1:19200,N,8,1,CS,DS,RB7500" FOR RANDOM AS #1
                                     'Programs COM1 port parameters to
                                     'match with the new instrument
                                     'settings.
PRINT #1, "QP 0,1"                  'Sends QUERY PRINT data command.
                                     '(actual screen for LaserJet print)
GOSUB Acknowledge                   'Input acknowledge from instrument.
PRINT
PRINT "Busy reading print data !"
PRINT
GOSUB Response
PRINT #1, "PC 1200"                 'Programs instrument back to the
                                     'default baud rate.
GOSUB Acknowledge                   'Input acknowledge from instrument.

PRINT "Print data copied to file 'QPFILE'."
PRINT "You can copy the file contents to the Laser 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  
'instrument. This routine inputs the acknowledge  
'response from the instrument. If the response is non-zero,  
'the previous command was not correct or was not correctly  
'received by the instrument. Then an error message is  
'displayed and the program is aborted.
```

Acknowledge:

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

```
'
***** Response subroutine *****
'This subroutine reads bytes from the RS232 buffer as long
'as they enter. When no bytes enter for 1 second, the program
'assumes that the instrument 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 *****
```

=====

QUERY SETUP

QS

Purpose:

Queries the present acquisition setup data from the instrument.

Command Syntax:

QS [<setup_no>]<cr>

where,

<saved_setup_no> = 0 to 10

This is the register number where a setup is stored. Also see the description of the Save Setup (SS) command (0 is 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 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_dat> 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 instrument.

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

Command Syntax:

QW returns list of available trace numbers

QW <trace_no>[,V|S]

<trace_no> = <decimal number> assigned to the following Trace Sources:

<trace_no> Trace Source:

10	MinMax voltage trace INPUT 1
	Modes: VOLTS/AMPS/HERTZ
	POWER
	TRANSIENTS
	SCOPE
11	24-bits Min/Max/Avg record trace 1
	Modes: VOLTS/AMPS/HERTZ (see Note)
	POWER (see Note)
	HARMONICS (see Note)
	SAGS & SWELLS
	SCOPE (see Note)
	Meter (see Note)
20	MinMax current trace INPUT 2
	Modes: VOLTS/AMPS/HERTZ
	POWER
	TRANSIENTS
	INRUSH CURRENT
	SCOPE (see Note)
21	24-bits Min/Max/Avg record trace 2
	Modes: VOLTS/AMPS/HERTZ (see Note)
	POWER (see Note)
	HARMONICS (see Note)
	SAGS & SWELLS
	SCOPE (see Note)

Note: Availability depends on other instrument setup variables.

V		v	Trace values (samples) only
S		s	Setup (administration) data only. When V or S is omitted, trace vales 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 voltage trace INPUT 1
,
'*****
'* 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 instrument
Resp$ = ""            'Clears the total Response string
GOSUB Response        'Writes waveform data to Resp$ & files
GOSUB Interpret.Admin 'Interprets waveform administration data
                        'See also Appendix C
GOSUB Interpret.Samples 'Interprets waveform sample data
GOSUB Create.CSV      'Creates Wave.CSV file from waveform data
                        'as input for Excel, for example.

END
```

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

Acknowledge:

```
INPUT #1, ACK           'Reads acknowledge from instrument.
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 instrument has terminated its response. All
'bytes that enter the buffer are appended to the string Resp$
'and are written to the following files:
'File Waveform : the waveform data bytes
'File Waveresp : the waveform ASCII values

```

Response:

```

start! = TIMER
'Wait for bytes (maximum 1 s) to enter RS232 buffer
WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
WEND
IF LOC(1) > 0 THEN      'If RS232 buffer contains bytes
    OPEN "WaveForm" FOR OUTPUT AS #2
                        'File to contain the waveform data bytes
    docount = 1
    total.count& = 0
    DO
        ' LOC(1) gives the number of bytes waiting:
        total.count& = total.count& + LOC(1)
        ScopeInput$ = INPUT$(LOC(1), #1)    'Input bytes
        PRINT #2, ScopeInput$;
        PRINT total.count&;
        Resp$ = Resp$ + ScopeInput$
        start! = TIMER
        WHILE ((TIMER < (start! + 1)) AND (LOC(1) = 0))
        WEND
        docount = docount + 1
    LOOP WHILE LOC(1) > 0    'Repeat as long as bytes enter
    CLOSE #2
    PRINT
END IF

```

,

```

'***** Write the total Response string to file WaveResp

```

,

```

OPEN "WaveResp" FOR OUTPUT AS #3
PRINT "Response data length = "; LEN(Resp$)
PRINT #3, "Response data length = "; LEN(Resp$)
FOR i = 1 TO LEN(Resp$)
    PRINT #3, ASC(MID$(Resp$, i, 1));
NEXT i
CLOSE #3: RETURN

```

,

Interpret.Admin:

```

Resp.Count = 1          'Byte counter for Resp$
SumCheck1% = 0         'Sumcheck byte for Resp$
'
'***** Interpret the <trace_admin> waveform data bytes
'***** in the Resp$ string (see appendix C).
'
'***** 2 bytes <trace_admin> block trailing : #0
IF MID$(Resp$, Resp.Count, 2) <> "#0" GOTO Wave.Error
Resp.Count = Resp.Count + 2
'
'***** 1 byte <block_header>
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb <> 128 AND nb <> 0 GOTO Wave.Error
Resp.Count = Resp.Count + 1
'
'***** 2 bytes <block_length>
Block1.Length = ASC(MID$(Resp$, Resp.Count, 1)) * 256
Block1.Length = Block1.Length + ASC(MID$(Resp$, Resp.Count + 1, 1))
Resp.Count = Resp.Count + 2
'
'***** 1 byte <trace_result> : 0, 1, or 2
Trace.Result = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Trace.Result
IF Trace.Result < 0 OR Trace.Result > 2 GOTO Wave.Error
Resp.Count = Resp.Count + 1
'
'***** 1 byte <y_unit>
Y.Unit = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Y.Unit
Resp.Count = Resp.Count + 1
PRINT "<y_unit>          =" ; Y.Unit,
'
'***** 1 byte <x_unit>
X.Unit = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + X.Unit
Resp.Count = Resp.Count + 1
PRINT "          <x_unit>          =" ; X.Unit
'
'***** 2 bytes <y_divisions>
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Sample.Byte
Y.Divisions = Sample.Byte * 256
Sample.Byte = ASC(MID$(Resp$, Resp.Count + 1, 1))
SumCheck1% = SumCheck1% + Sample.Byte
Y.Divisions = Y.Divisions + Sample.Byte
Resp.Count = Resp.Count + 2
PRINT "<y_divisions>          =" ; Y.Divisions,
'
'***** 2 bytes <x_divisions>
Sample.Byte = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck1% = SumCheck1% + Sample.Byte
X.Divisions = Sample.Byte * 256
Sample.Byte = ASC(MID$(Resp$, Resp.Count + 1, 1))
SumCheck1% = SumCheck1% + Sample.Byte
X.Divisions = X.Divisions + Sample.Byte

```

```
Resp.Count = Resp.Count + 2
PRINT "      <x_divisions>      ="; X.Divisions
```

```

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

```

```
, PRINT "      <x_step>          ="; X.Step
```

```

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

```

```
nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))  
ELSE
```

```
,
```

```

    nb = nb * 256                                'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(3) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(3) = nb
Resp.Count = Resp.Count + 3
'*****
'* Further calculation after 'Signed.Samples' determination
'*****
'***** 3 bytes <x_resolution> = <mantissa_high><mantissa_low><exponent>
'***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
'***** <x_resolution> = <sign><mantissa> E <sign><exponent>
'*****           Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256                        'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                                'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(4) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(4) = nb
Resp.Count = Resp.Count + 3
'*****
'* Further calculation after 'Signed.Samples' determination
'*****
'***** 3 bytes <y_at_0> = <mantissa_high><mantissa_low><exponent>
'***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
'***** <y_at_0> = <sign><mantissa> E <sign><exponent>
'*****           Example: +123E-4 = 123 / 10000 = 0.0123
FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 2
NEXT i
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb >= 128 THEN
    nb = - (256 - nb) * 256                        'Negative value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
ELSE
    nb = nb * 256                                'Positive value
    nb = nb + ASC(MID$(Resp$, Resp.Count + 1, 1))
END IF
exponent(5) = ASC(MID$(Resp$, Resp.Count + 2, 1))
YXvalue#(5) = nb
Resp.Count = Resp.Count + 3
'*****
'* Further calculation after 'Signed.Samples' determination
'*****
'***** 3 bytes <x_at_0> = <mantissa_high><mantissa_low><exponent>
'***** <mantissa> = <mantissa_high> * 256 + <mantissa_low>
'***** <x_at_0> = <sign><mantissa> E <sign><exponent>
'*****           Example: +123E-4 = 123 / 10000 = 0.0123

```



```

FOR i = 0 TO 2
    SumCheck1% = (SumCheck1% + ASC(MID$(Resp$,Resp.Count+i,1))) MOD 256
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 256
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$,Resp.Count+i,1))) MOD 256
NEXT i
Hours$ = MID$(Resp$, Resp.Count, 1)
Hours$ = Hours$ + MID$(Resp$, Resp.Count + 1, 1)
Minutes$ = MID$(Resp$, Resp.Count + 2, 1)
Minutes$ = Minutes$ + MID$(Resp$, Resp.Count + 3, 1)
Seconds$ = MID$(Resp$, Resp.Count + 4, 1)
Seconds$ = Seconds$ + MID$(Resp$, Resp.Count + 5, 1)
Resp.Count = Resp.Count + 6
PRINT "    <time_stamp>      = "; Hours$+":"+Minutes$+":"+Seconds$
,
'***** 1 byte <check_sum>
Check.Sum% = ASC(MID$(Resp$, Resp.Count, 1))
IF Check.Sum% <> (SumCheck1% MOD 256) GOTO Wave.Error
Resp.Count = Resp.Count + 1
PRINT "<check_sum> ="; Check.Sum%; " & ";
PRINT "SumCheck1 MOD 256 ="; SumCheck1% MOD 256
RETURN
Wave.Error:
PRINT "Waveform admin error at byte   :"; Resp.Count
PRINT "Waveform decimal byte value   ="; ASC(MID$(Resp$,Resp.Count,1))

```

```
PRINT "SumCheck so far (MOD 256)      ="; SumCheck1% MOD 256  
CLOSE: END
```

Interpret.Samples:

```

'***** Interpret the <trace_samples> waveform data bytes
'***** in the Resp$ string (see appendix C).
'*****
'***** 1 byte separator admin/samples : ,
'***** 2 bytes <trace_samples> block trailing : #0
,
SumCheck2% = 0
IF MID$(Resp$, Resp.Count, 3) <> ",#0" GOTO Wave2.Error
Resp.Count = Resp.Count + 3
,
'***** 1 byte <block_header>
nb = ASC(MID$(Resp$, Resp.Count, 1))
IF nb <> 129 AND nb <> 0 GOTO Wave2.Error
Resp.Count = Resp.Count + 1
,
'***** 2 bytes <block_length>
Block2.Length& = ASC(MID$(Resp$, Resp.Count, 1))
Block2.Length& = Block2.Length& * 256
Block2.Length& = Block2.Length& + ASC(MID$(Resp$, Resp.Count+1, 1))
Resp.Count = Resp.Count + 2
PRINT "Number of sample chars ="; Block2.Length&
OPEN "Samples" FOR OUTPUT AS #4
PRINT #4, "Number of sample chars ="; Block2.Length&
,
'***** 1 byte <sample_format>
Sample.Format = ASC(MID$(Resp$, Resp.Count, 1))
SumCheck2% = SumCheck2% + Sample.Format
IF (Sample.Format AND 128) = 128 THEN
    Signed.Samples = 1
ELSE
    Signed.Samples = 0
END IF
IF (Sample.Format AND 112) = 64 THEN 'bits 6, 5, 4
    MinMax.Samples = 1 'Min/Max=100
ELSEIF (Sample.Format AND 112) = 96 THEN
    MinMax.Samples = 2 'Min/Max/Ave=110
ELSEIF (Sample.Format AND 112) = 0 THEN
    MinMax.Samples = 0 'Normal=000
ELSEIF (Sample.Format AND 112) = 112 THEN
    IF MID$(Query$, 5, 1) = "1" THEN 'TrendPlot
        MinMax.Samples = 2 'Min=Max=Ave=111
    ELSE 'Average Min/Max
        MinMax.Samples = 1 'Min=Max=111
    END IF
ELSE
    MinMax.Samples = 7 'Unknown format!
END IF
Sample.Bytes = Sample.Format AND 7
IF Sample.Bytes = 1 THEN 'Single-byte samples
    CLimit = C128 : CMaxim = C256
ELSE 'Double-byte samples
    CLimit = C32768 : CMaxim = C65536
END IF

```

```

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

```

```
        END IF  
    NEXT j  
,
```

```

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

```

```

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

```


Resp.Count = Resp.Count + Sample.Bytes

```

IF i=1 OR i=2 OR i = Nbr.Of.Samples-1 OR i = Nbr.Of.Samples THEN
  IF (Signed.Samples = 0) OR (Sample.Value&(i) < CLimit) THEN
    Ampl.Value = Sample.Value&(i) * Y.Resol  'Positive value
  ELSE  'Negative value
    Ampl.Value = - ((CMaxim - Sample.Value&(i)) * Y.Resol)
  END IF
  PRINT "Sample"; i; "="; Sample.Value&(i); Ampl.Value
END IF
PRINT #4, "Sample"; i; "="; Sample.Value&(i); Ampl.Value
NEXT i
'
'***** 1 byte <check_sum>
Check.Sum% = ASC(MID$(Resp$, Resp.Count, 1))
IF Check.Sum% <> (SumCheck2% MOD 256) GOTO Wave2.Error
Resp.Count = Resp.Count + 1
PRINT "<check_sum> ="; Check.Sum%; " & ";
PRINT "SumCheck2 MOD 256 ="; SumCheck2% MOD 256
PRINT #4, "<check_sum> ="; Check.Sum%; " & ";
PRINT #4, "SumCheck2 MOD 256 ="; SumCheck2% MOD 256
'
'***** 1 byte CR
C.R = ASC(MID$(Resp$, Resp.Count, 1))
IF C.R <> 13 GOTO Wave2.Error
Resp.Count = Resp.Count + 1
CLOSE #4: RETURN
Wave2.Error:
PRINT "Waveform sample error at byte :"; Resp.Count
PRINT "Waveform decimal byte value   ="; ASC(MID$(Resp$,Resp.Count,1)
PRINT "SumCheck so far (MOD 256)     ="; SumCheck2% MOD 256
CLOSE: END

```

Create.CSV:

```

'
'*****
'***** Convert the total Response string to file Wave.CSV
'***** as input file for Excel (spreadsheet), for example.
'*****
'
OPEN "Wave.CSV" FOR OUTPUT AS #4
PRINT #4, "Title      , ";
IF MID$(Query$, 4, 2) = "10" THEN
    PRINT #4, "Voltage trace"
ELSEIF MID$(Query$, 4, 2) = "11" THEN
    PRINT #4, "Record trace 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 'Record trace
    PRINT #4, "Type          ,"; "Record 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"
IF Y.Unit > 3 OR Y.Unit = 0 THEN PRINT #4, "No print"
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"
IF Y.Unit > 3 OR Y.Unit = 0 THEN PRINT #4, Y.Scale; "No print"

```

PRINT #4,

```

'***** Sample values x,y (time,amplitude)
Time.Value = X.Zero           'Start at x-offset
MinMax.Flag = MinMax.Samples 'Switch flag (2, 1, 0)
FOR i = 1 TO Nbr.Of.Samples
  IF (Signed.Samples = 0) OR (Sample.Value&(i) < CLimit) THEN
    'Positive value
    Amplit.Value = Sample.Value&(i) * Y.Resol
  ELSE
    'Negative value
    Amplit.Value = - ((CMaxim - Sample.Value&(i)) * Y.Resol)
  END IF
  IF MinMax.Samples = 2 THEN           'Min/Max/Ave waveform
    IF MinMax.Flag = 2 THEN
      MinMax.Flag = MinMax.Flag - 1
      PRINT #4, Time.Value; ", "; Amplit.Value; ", ";
    ELSEIF MinMax.Flag = 1 THEN
      MinMax.Flag = MinMax.Flag - 1
      PRINT #4, Amplit.Value; ", ";
    ELSE
      MinMax.Flag = 2
      PRINT #4, Amplit.Value
      Time.Value = Time.Value + X.Resol
    END IF
  END IF
  IF MinMax.Samples = 1 THEN           'Min/Max waveform
    IF MinMax.Flag = 1 THEN
      MinMax.Flag = 0
      PRINT #4, Time.Value; ", "; Amplit.Value; ", ";
    ELSE
      MinMax.Flag = 1
      PRINT #4, Amplit.Value
      Time.Value = Time.Value + X.Resol
    END IF
  END IF
  IF MinMax.Samples = 0 THEN           'Single waveform
    PRINT #4, Time.Value; ", "; Amplit.Value
    Time.Value = Time.Value + X.Resol
  END IF
NEXT i
CLOSE #4: RETURN
'***** End 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. 2001,8,14

Example:

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

```

'***** 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 instrument.
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
'instrument. This routine inputs the acknowledge
'response from the instrument. If the response is non-zero,
'the previous command was not correct or was not correctly
'received by the instrument. Then an error message is
'displayed and the program is aborted.

Acknowledge:
INPUT #1, ACK          'Reads acknowledge from instrument.
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 instrument settle itself before you send the
next command.

Example:

The following example resets the instrument and waits for 2
seconds to let the instrument execute the reset and become
ready for next commands.

The instrument 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 instrument.
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 instrument.
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
'instrument. This routine inputs the acknowledge
'response from the instrument. If the response is non-zero,
'the previous command was not correct or was not correctly
'received by the instrument. Then an error message is
'displayed and the program is aborted.

```

Acknowledge:

```

INPUT #1, ACK 'Reads acknowledge from instrument.
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 *****

```

=====

RECALL SETUP RS

Purpose:

Recalls an internally stored setup. This setup must have been stored in the instrument 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 10 : Screen/Setup memories

Response Syntax:

<acknowledge><cr>

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

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.

'***** 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 instrument
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 instrument 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 instrument.
PRINT
PRINT "Original settings restored"
CLOSE #1
END
```

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

Acknowledge:

```
INPUT #1, ACK          'Reads acknowledge from instrument.  
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 instrument.

```
'
                                     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 instrument.
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
'instrument. This routine inputs the acknowledge
'response from the instrument. If the response is non-zero,
'the previous command was not correct or was not correctly
'received by the instrument. Then an error message is
'displayed and the program is aborted.

Acknowledge:
INPUT #1, ACK           'Reads acknowledge from instrument.
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 instrument on.

This only works when the instrument is powered via the power adapter.

Command Syntax:

SO<cr>

Response Syntax:

<acknowledge><cr>

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

=====

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 10 : Screen/Setup memories

When <setup_reg> is omitted, the number
1 is assumed.

Response Syntax:

<acknowledge><cr>

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

=====

STATUS QUERY

ST

Purpose:

Queries the error status of the instrument.
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 instrument 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 instrument when the acknowledge response is non-zero (ACK <> 0).

```

'
'***** 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 instrument.
GOSUB Acknowledge.Status 'Input acknowledge from instrument
'and the status value if the
'acknowledge value is non-zero.
END

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

Acknowledge.Status:
INPUT #1, ACK 'Reads acknowledge from instrument.
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 instrument status *****
'
This subroutine gives you further information if the
'acknowledge reply from the instrument is non-zero.

Status.display:
PRINT #1, "ST"                      'Sends the STATUS query.
GOSUB Acknowledge.Status            'Inputs acknowledge from instrument.
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:

```
'
                                     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 instrument.
END

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

Acknowledge:
INPUT #1, ACK                           'Reads acknowledge from instrument.
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. 2001,9,14

Response Syntax:

<acknowledge><cr>

Example:

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

```
'
                                     Begin example program *****
CLS                                     'Clears the PC screen.
OPEN "COM1:1200,N,8,1,CS,DS,RB2048" FOR RANDOM AS #1
PRINT #1, "WD 2001,9,14"             'Sets the real time clock
                                     'to September 14, 2001
GOSUB Acknowledge                     'Input acknowledge from instrument.
END

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

Acknowledge:
INPUT #1, ACK                         'Reads acknowledge from instrument.
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 instrument with a new time setting.


```
'
                                Page 3.81

'***** 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. (15.28).
GOSUB Acknowledge                'Input acknowledge from instrument.
END

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

Acknowledge:
INPUT #1, ACK                    'Reads acknowledge from instrument.
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 *****
```

The instrument 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 instrument.

<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 instrument 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 instrument 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 instrument receives data while it does not expect any data. This can occur as follows:

- The instrument 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 instrument, 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.

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: 128 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> = <acquisition> | <record_trace>
 = <binary_character>. Specifies which
 function created this particular trace:
 <acquisition> = 1 The trace is a direct result of the
 trace acquisition.
 <record_trace>= 2 The trace is a result of the RECORD
 function (recording numerical results).

<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.
For the Fuke 43B, 1-2-5 or 2-4-10 scale.

<x_scale> = <float>
Number of units per x division. Depending on application mode, 1-2-5 or variable scale.

<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 0 or 129.

<block_length>= <unsigned_integer>
This (2-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
 or Min=Max=Average trace samples
 <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 occur in random sampling, for example.

<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

Hexadecimal value		ASCII character		Decimal value	
Hex	Char	Hex	Char	Dec	Char
00	NUL	0		20	SP 32
01	SOH	1	!	33	
02	STX	2	"	34	
03	ETX	3	#	35	
04	EOT	4	\$	36	
05	ENQ	5	%	37	
06	ACK	6	&	38	
07	BEL	7	'	39	
08	BS	8	(40	
09	HT	9)	41	
0A	LF	10	*	42	
0B	VT	11	+	43	
0C	FF	12	,	44	
0D	CR	13	-	45	
0E	SO	14	.	46	
0F	SI	15	/	47	
10	DLE	16		30	0 48
11	XON	17	1	49	
12	DC2	18	2	50	
13	XOF	19	3	51	
14	DC4	20	4	52	
15	NAK	21	5	53	
16	SYN	22	6	54	
17	ETB	23	7	55	
18	CAN	24	8	56	
19	EM	25	9	57	
1A	SUB	26	:	58	
1B	ESC	27	;	59	
1C	FS	28	<	60	
1D	GS	29	=	61	
1E	RS	30	>	62	
1F	US	31	?	63	
40	@	64		70	p 112
41	A	65		71	q 113
42	B	66		72	r 114
43	C	67		73	s 115
44	D	68		74	t 116
45	E	69		75	u 117
46	F	70		76	v 118
47	G	71		77	w 119
48	H	72		78	x 120
49	I	73		79	y 121
4A	J	74		7A	z 122
4B	K	75		7B	{ 123
4C	L	76		7C	124
4D	M	77		7D	} 125
4E	N	78		7E	~ 126
4F	O	79		7F	127
50	P	80			
51	Q	81			
52	R	82			
53	S	83			
54	T	84			
55	U	85			
56	V	86			
57	W	87			
58	X	88			
59	Y	89			
5A	Z	90			
5B	[91			
5C	\	92			
5D]	93			
5E	^	94			
5F	_	95			

Hexadecimal value

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	f	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	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	178	D2	Ò	210	F2	ò	242
93	"	147	B3	179	D3	Ó	211	F3	ó	243
94	"	148	B4	180	D4	Ô	212	F4	ô	244
95	•	149	B5	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	185	D9	Ø	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	188	DC	Ü	220	FC	ü	252
9D		157	BD	189	DD	Ý	221	FD	ý	253
9E	?	158	BE	190	DE	Þ	222	FE	þ	254
9F	ÿ	159	BF	191	DF	ß	223	FF		255