SPDM id 200
Programming Guide

Version 1.1
July 2004

id Quantique
Ch. de la Marbrerie 3
CH-1227 Carouge
Switzerland

sales@idquantique.com
www.idquantique.com

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

No license, express or implied, to any intellectual property rights is granted herein, except that a license is hereby granted to copy and reproduce this specification for internal use only. Contact id Quantique for information on further licensing agreements and requirements. id Quantique disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. id Quantique assumes no liability whatsoever, and disclaims any express or implied warranty, relating to sale and/or use of id Quantique products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. id Quantique products are not intended for use in medical, life saving, or life sustaining applications.

id Quantique may make changes to documents, specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." id Quantique reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Table of contents

Tutorial

Where to begin

Programming and accessing your Single-Photon Detector Module (SPDM) using its RS-232C interface is simple and allows the implementation of complex functions. In this chapter, we will discuss hot to connect the SPDM to a computer using a RS-232C interface and how to use a terminal program to test the connection.

Connecting your SPDM to a serial port

Before trying to connect your SPDM to a computer, please make sure that both devices are turned off.

IMPORTANT

Connecting the interface port of your SPDM to the serial of a computer when they are turned on, may permanently damage these devices.

Connect the interface port located on the rear panel of your SPDM to a serial port on the computer.

IMPORTANT

Make sure that you are using the right cable. The cable specifications are:

DB9/DB9 – Male/Female – Straight Thru Cable

You can now turn your SPDM on by using the front panel ON/OFF switch.

Using a terminal program to test the connection

In order to remotely program and control the id 200 SPDM, one needs to be able to send and receive information to and from a RS-232C port. This can be done in several ways, like, for example, by writing a Basic or C program or using the Labview demonstration program. In this chapter, we will consider the use of a terminal program in order to communicate with the SPDM.

NOTE

The HyperTerminal terminal application comes with all Windows platforms. It can be launched from the Start menu.

NOTE

Please refer to Appendix 1 for a list of the interface parameters. If you are not familiar with the configuration of a terminal program, please refer to Appendix 2, where the configuration of the HyperTerminal program is outlined.

Now that the SPDM and the computer have been properly connected and switched on, please launch the terminal program and open the connection.
You are now ready to send the first command to your SPDM.

Please type:

```
:Device:Systemstate?
```

The SPDM will reply:

```
COOLING
```

This indicates that you have correctly connected the SPDM and the computer, and that they can exchange information. Wait for a few minutes until the cooling phase of the SPDM is over and the assigned cooling temperature has been reached.

Type:

```
:Device:Systemstate?
```

You will read:

OPERATING

Your SPDM is now ready to count photons. Type:

:Trigger:Source Internal

You will read:

OK

You have set your SPDM in internal trigger mode. You should now specify the trigger rate. Type:

:Trigger:Rate 10

You will read:

OK

You may want to query the present trigger rate, in order to make sure that you have set it correctly. Type:

:Trigger:Rate?

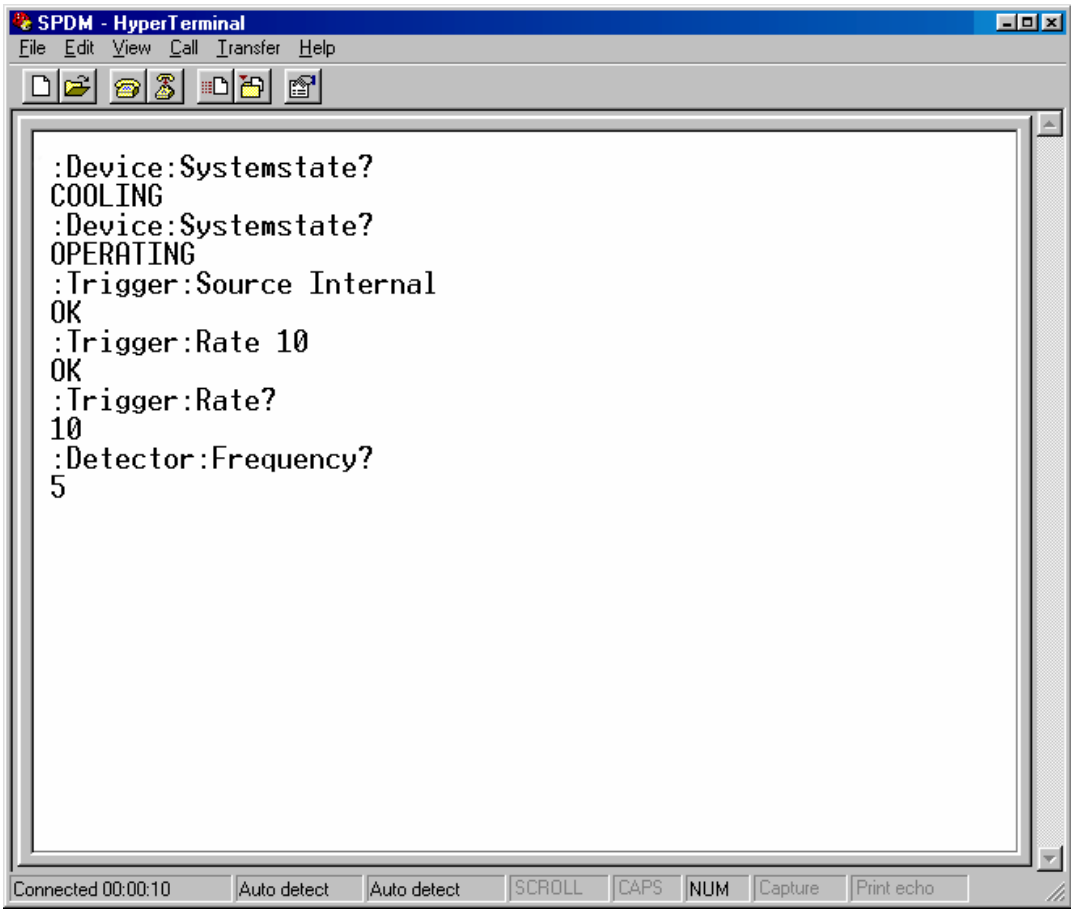
You will read:

10

You can now check what the counting frequency is. Type:

:Detector:Frequency?

You will read a number depending, among other things, on the detector illumination and the gate width.
The following figure represents this a screen capture of this session.



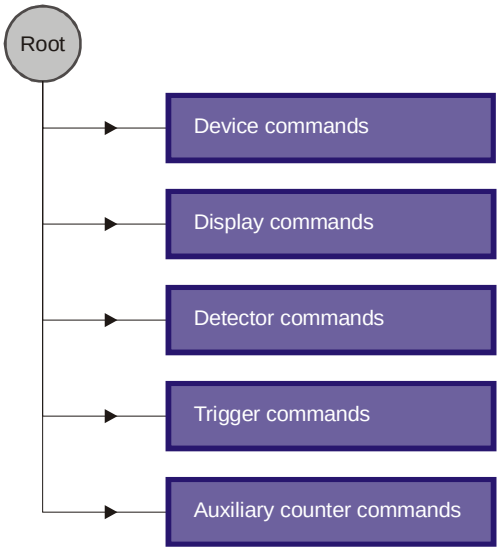
Command Overview

Introduction

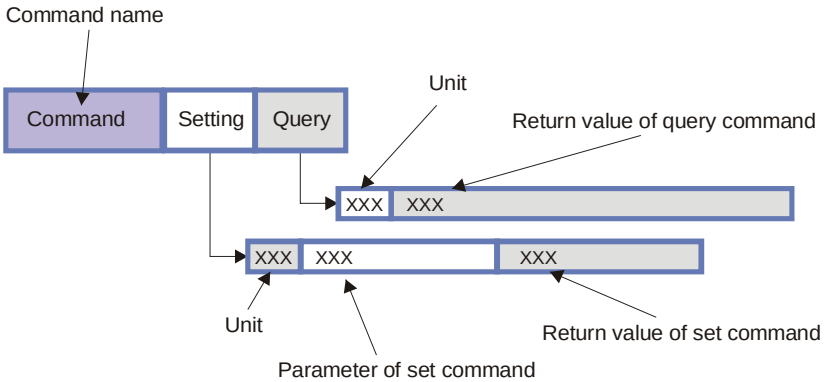
This chapter briefly presents an overview of the commands that can be use to control the id 200 Single-Photon Detector Module (SPDM).

There are four groups of commands:

- Device commands: These commands are used to obtain information about the status and the firmware version of the SPDM, as well as starting or stopping it.
- Display commands: These commands are used to set the display mode of the SPDM
- Trigger commands: These commands are used to set the parameters of the trigger, as well as querying the trigger counter.
- Detector commands: These commands are used to set the parameters of the detector, as well as querying the detector counter.
- Auxiliary counter commands: These commands are used to set the parameters of the auxiliary counter, as well as querying the auxiliary counter.

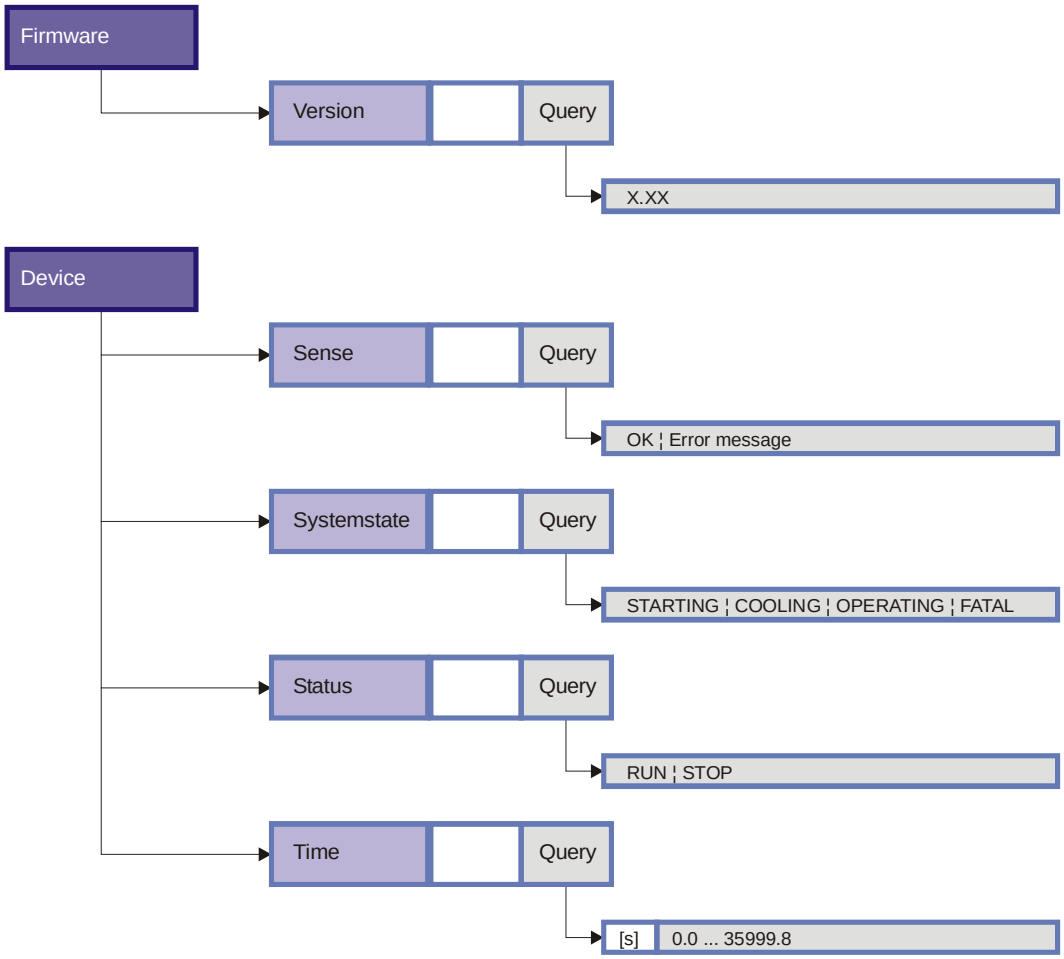


Each command group is presented in a hierarchical diagram listing syntax, parameters and parameters format, and return values and return values format.

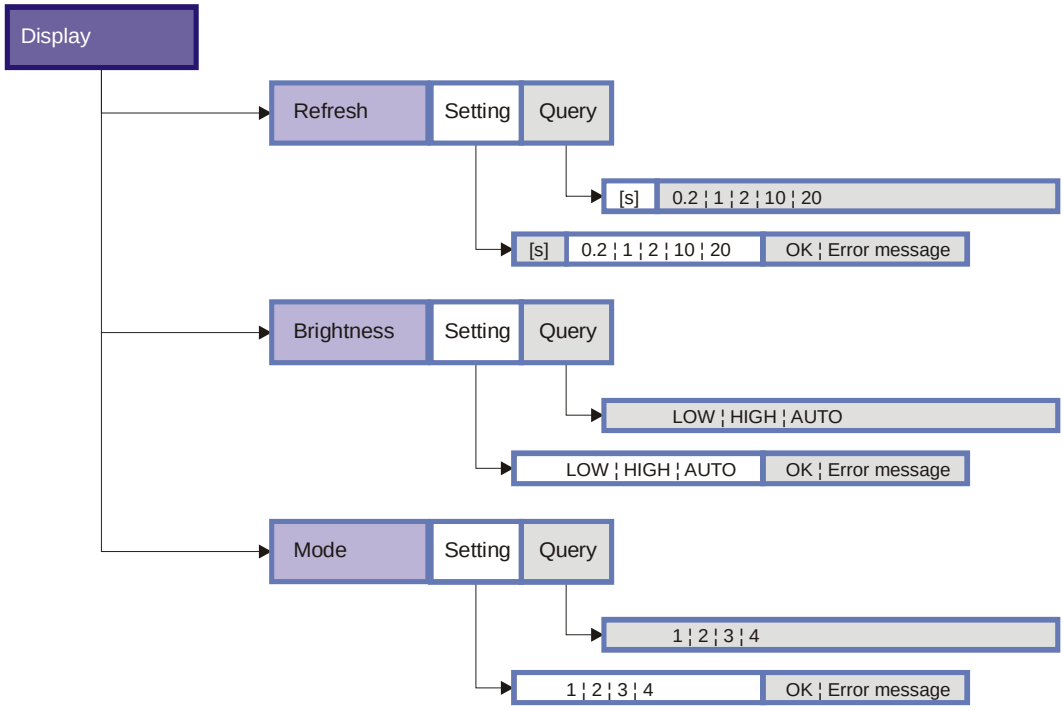


The description of a command mentions its name and whether it is a query command or a setting and query command. For query commands, its return values are listed. For setting and query commands, both its parameters and its return values are listed. For parameters and return values, their units are mentioned.

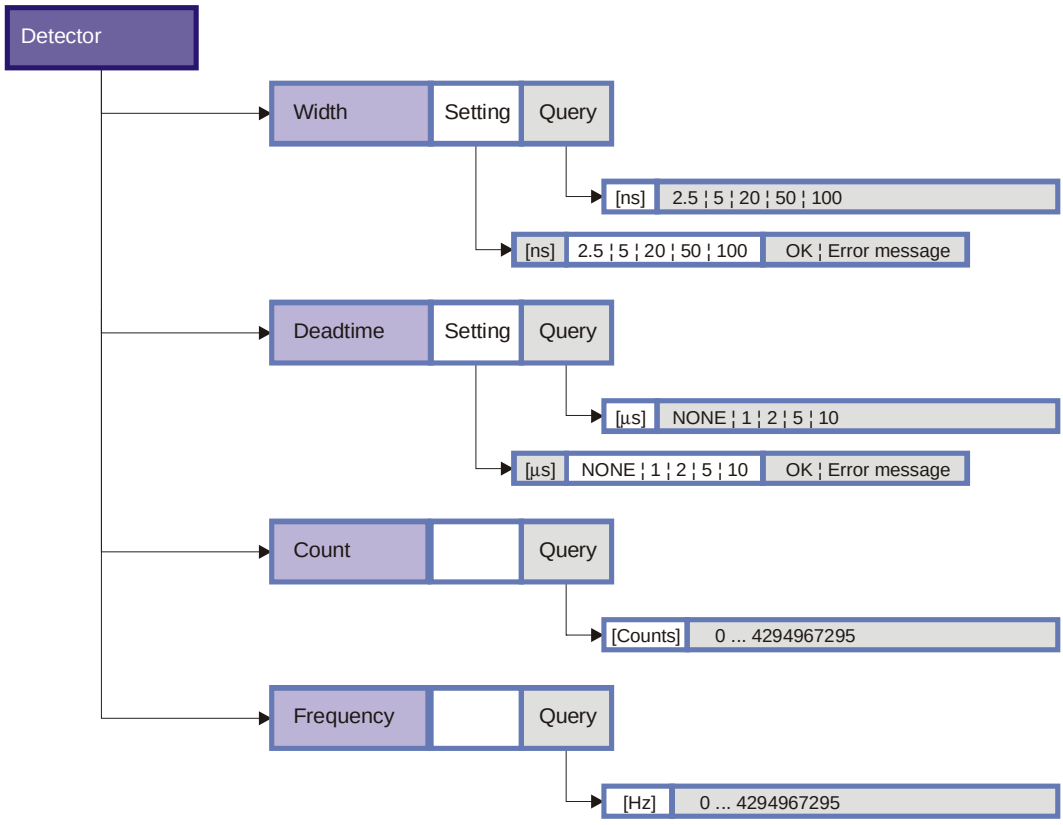
Device commands group



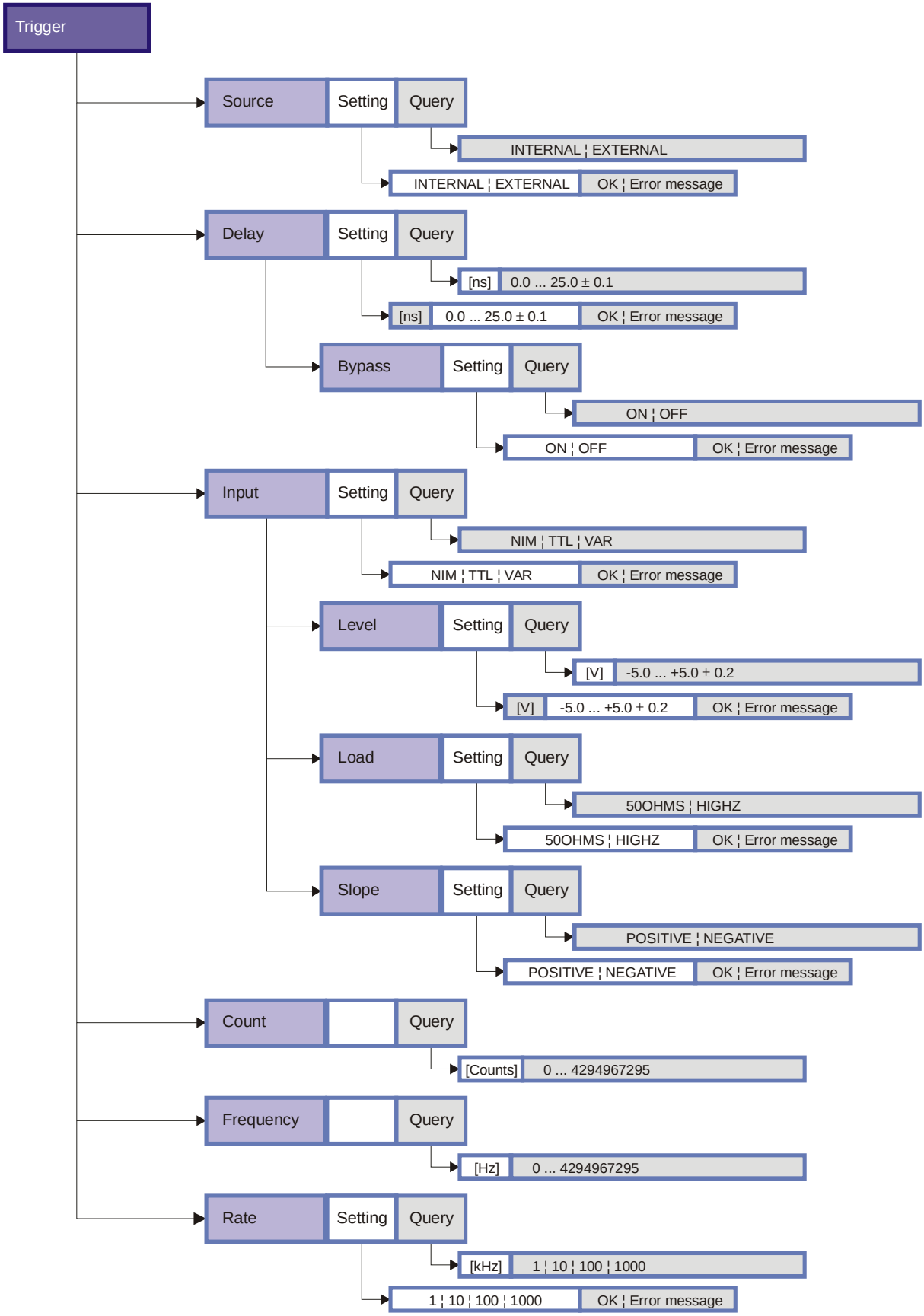
Display commands group



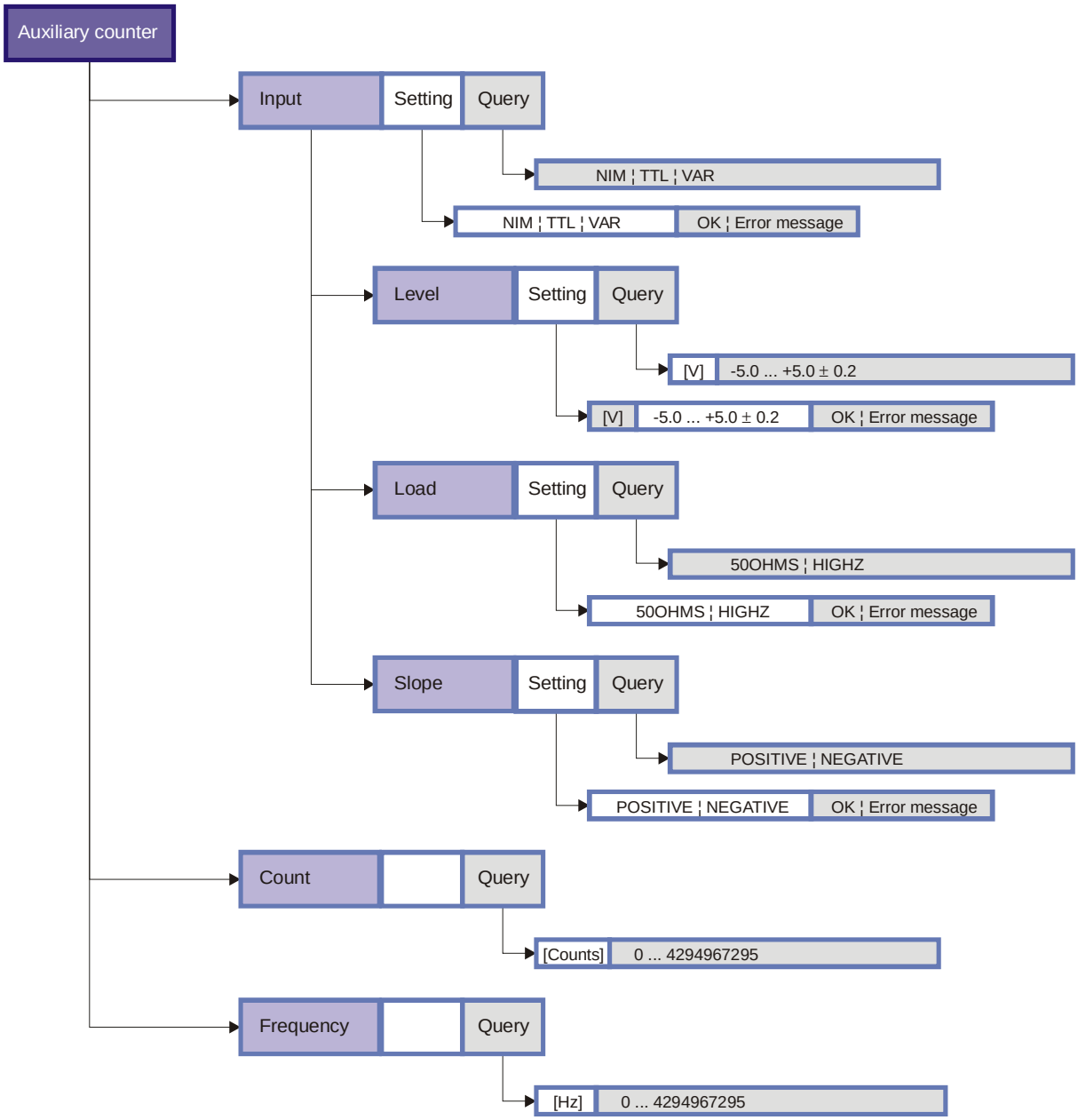
Detector commands group



Trigger commands group



Auxiliary counter commands group



Interfacing and programming your id 200 SPDM

Introduction

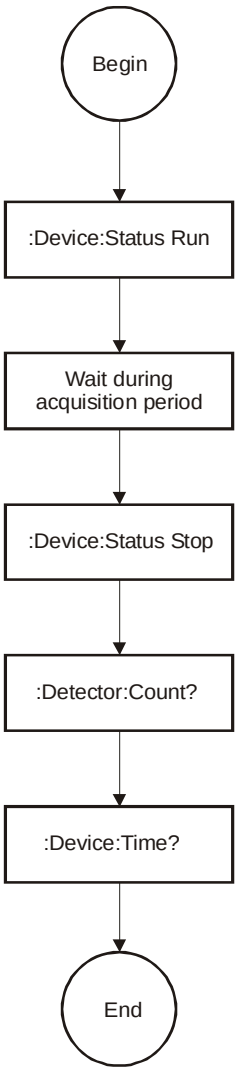
The id 200 SPDM has three counters – one for the detector, one for the trigger and one for the auxiliary input. This chapter explains how to perform standard remote measurement using your id 200 SPDM and its interface of these counters. The procedure for measuring the number of counts during a certain acquisition time, as well as that for reading the frequency of one of the counters are outlined. The chapter also presents a demonstration program written in C, as well as a Labview virtual instrument that can be used to control the id 200 SPDM. The concepts presented in this chapter will be illustrated using the detector counter, but they can be readily applied to the two other counters.

Measuring count numbers

In order to measure the number of counts recorded by one of these counters during a well-defined acquisition period, one should follow the following procedure:

- The timing clock of the id 200 SPDM must be reset (`:Device:Status Run` command).
- The application must then wait for a time equal to the acquisition period.
- When the end of the acquisition period is finished, the counters should be freezed (`:Device:Status Stop` command).
- The number of counts can now be read from the SPDM (`:Detector:Count?` command).
- The exact acquisition time can also be read (`:Device:Time?` command).

The following figure illustrates this procedure with a flow chart.

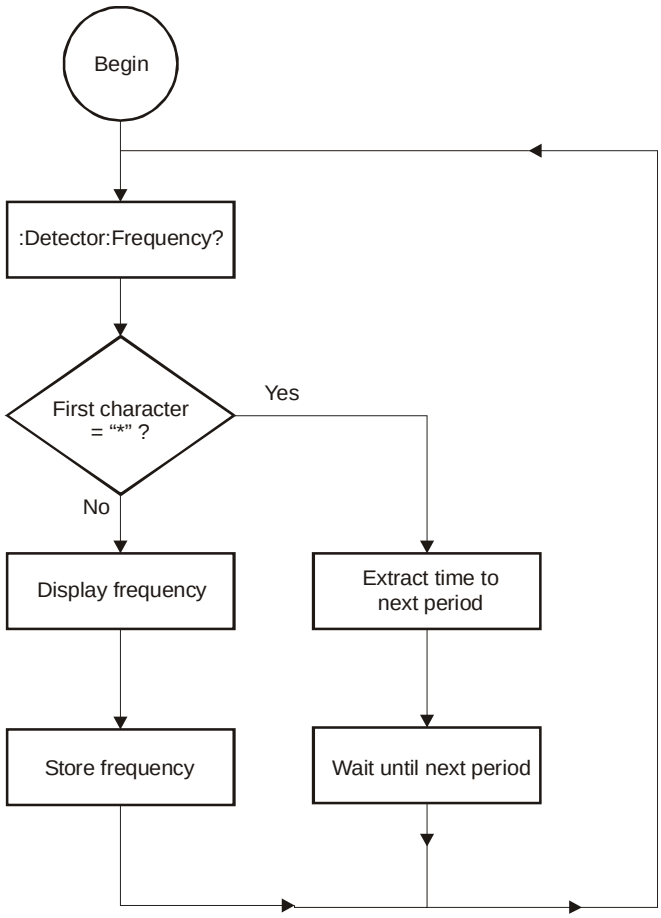


Reading a frequency

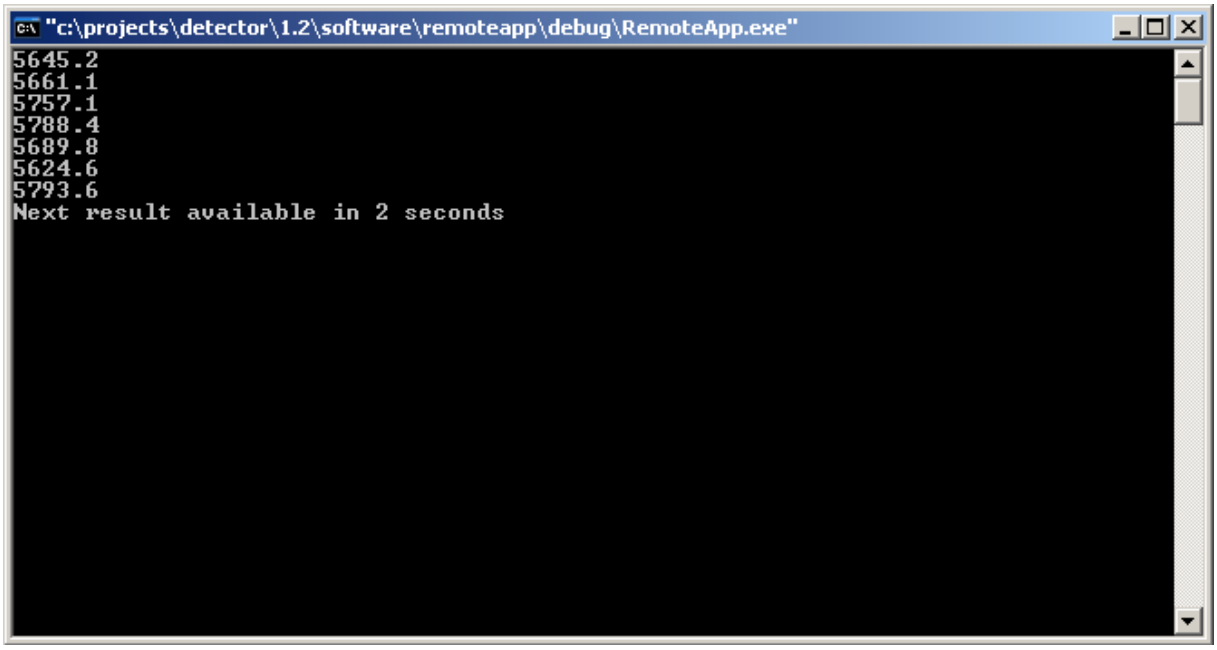
A simple command can be used to read the frequency of one of the counters. In the case of the detector counter, one should simply send the `:Detector:Frequency?` command to the id 200 SPDM.
When the refresh period of the frequency measurement is long (typically 10 or 20 s), it is always difficult to synchronize the acquisition application on the computer and the id 200 SPDM in such a way that each frequency result is read once and only once. The id 200 SPDM interface integrates a mechanism that solves this problem. When the frequency of a counter is queried several times during a refresh period, the id 200 SPDM returns its value only the first time. At subsequent occasions, it will return a character string starting with a "*" and followed by the time until the next refresh period. As soon as the next refresh period is reached and the frequency queried again, the id 200 SPDM will return a frequency value again once.
The following procedure should thus be used to read a frequency:

- The frequency of the detector is read from the id 200 SPDM (`:Detector:Frequency?` command).
- If the returned value starts with a "*" character, it means that the frequency has not been updated since the last read out. The program should thus extract the time until the next refresh period and wait, before reading the frequency again.
- If the returned value does not start with a "*" character, it means that a new frequency value has been read. It can thus be displayed and stored.
- The program can then go back to the beginning of the procedure.

The figure below shows a flow chart illustrating this procedure:

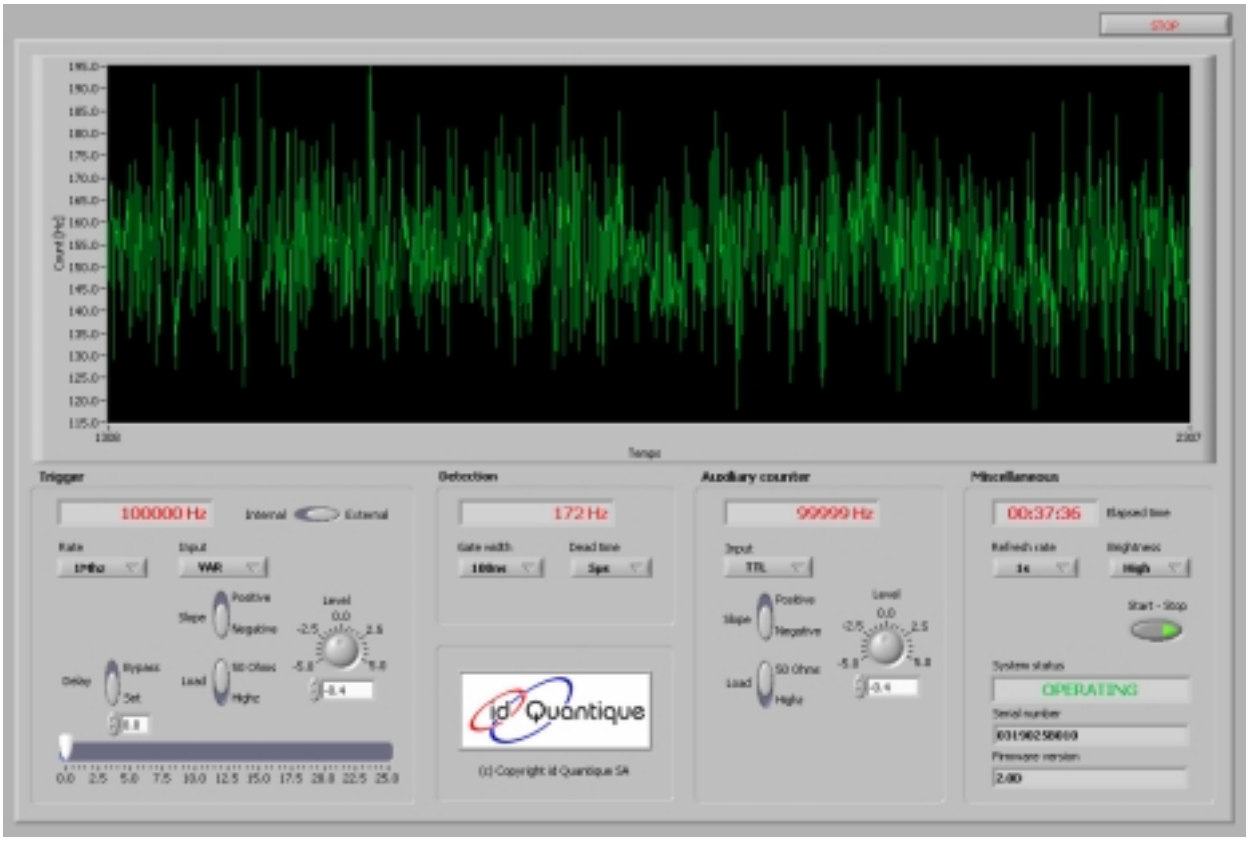


A sample acquisition program is supplied to illustrate how to perform a frequency read out. Both the source file and a compiled version are supplied. The source code is also listed in the Appendix.
The program reads the frequency of the detector and displays it in a console window. As long as a new result is not available, the program displays "Next result available in X seconds". When the new result becomes available, it is displayed and the program waits for the next one.
The figure below shows the window obtained during an execution of the program.



Labview Virtual Instrument

In order to simplify the use of the id 200 SPDM in remote mode with its interface, a Labview virtual instrument (VI) is also supplied. This VI allows setting all the parameters of the id 200 SPDM. It also displays as a function of time the number of counts recorded. The figure below shows the control window of this VI.



NOTE
The Labview program is necessary to run this virtual instrument and is not supplied with the interface package. Please contact your local Labview dealer.

Command Reference

Introduction

This chapter describes the commands for the id 200 Single-Photon Detector Module (SPDM). The information in this chapter will help you program the SPDM over the RS-232C interface.

The commands are presented in alphabetical order.

For each command description:

- where the phrase “sets or queries” is used, the command setting can be queried by omitting the parameter and appending a “?” to the last command keyword.

  For example,

  :Trigger:Source Internal

  can be queried with

  Trigger:Source?

- when a command is used to set a parameter, the last keyword is separated by one compulsory space character.

- commands and parameters can be typed either in upper or lower case. The SPDM is not case sensitive.

- the square brackets, [], are used to indicate a parameter list. The brackets are not part of the command and should not be sent to the SPDM.

- the character “|” is used as an OR to separate alternative options. The “|” character is not part of the command and should not be sent to the SPDM.

- when a parameter or a return value is a number coded in a string, the number type is written in between “<” and “>”.

  For example,

  <INTEGER>

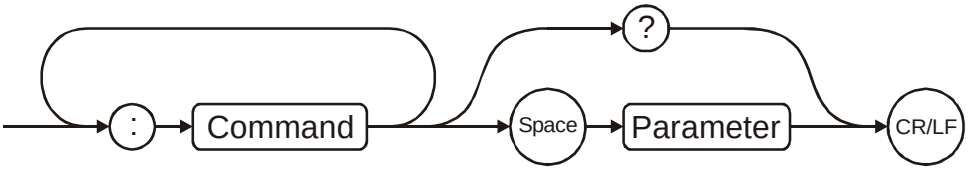
- when a parameter or a return value consists of a number, the minimum and maximum values are listed separated by a “…” character.

  For example, the possible return values of the Trigger:Count? command, are listed:

  Return Value
  0 … 4294967295
  Integer

All commands return a character string. A valid query command always returns the queried value or an error message. A valid setting command returns a “OK” string or an error message.

The following figure illustrates the interpretation of a command by the id 200 SPDM. It looks first for a “:” character followed by a string. If an additional “:” character is present, the first keyword is followed by a second one. A query command is indicated by a “?” directly appended to the keyword string. If the “?” is absent, the command is a setting command and is followed by a space and a parameter. In all case the command line finishes by a carriage return or a line feed.



:AuxCounter:Count?

This commands queries the number of events registered by the auxiliary counter since the last time it was reset.

Command type

Query only

Return Value

Character string coding an integer.
<INTEGER>
Range: [0 … 4294967295]
Units: Counts

Example

:AuxCounter:Count?
23896

:AuxCounter:Frequency?

This commands queries the frequency measured by the auxiliary counter over the previous display refresh period. The first time the command is sent to the SPDM during a refresh period, it returns the frequency measured during the previous period. After that, it will return a "*" character string followed by the remaining acquisition time in the current period. This indicates that the frequency value has not been updated yet and that the value previously read is still up to date.

Command type

Query only

Return Value

The first time a value is read, it consists of a character string coding a real number with fixed number of decimal places.
Subsequently, the SPDM returns a "" string followed by the time until the next acquisition period. This time is returned with one decimal place.*
*[<REAL>|"*X.X"]*
Units: Hz or seconds

Comments

The number of decimal places of the number returned depends on the refresh rate.

Example

:AuxCounter:Frequency?
2540.3
:AuxCounter:Frequency?
*2.4

:AuxCounter:Input

This commands sets and queries the input type of the signals that are to be recorded by the auxiliary counter. Upon successful setting of a parameter, this command returns a character string containing "OK".

Command type

Setting and Query

Parameters

[NIM|TTL|VAR]

Return Value

Setting command:
OK

Query command:
[NIM|TTL|VAR]

Example

:AuxCounter:Input TTL

OK

:AuxCounter:Input?

TTL

:AuxCounter:Input:Level

This commands sets and queries the input voltage level of the signals to be recorded by the auxiliary counter, when its input type is set to variable (see :AuxCounter:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

<REAL>

Range: [-5.0 … +5.0]

Step: 0.2

Unit: Volt

Return Value

Setting command:

[OK|Error message]

Query command:

<REAL>

Range: [-5.0 … +5.0]
Step: 0.2

Comments

The input level setting accuracy is 0.2 V. Different values are rounded.

Example

:AuxCounter:Input:Level –0.4
OK
:AuxCounter:Input:Level?
-0.4

:AuxCounter:Input:Load

This commands sets and queries the impedance of the input of the auxiliary counter, when its input type is set to variable (see :AuxCounter:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[50OHMS|HIGHZ]

Return Value

Setting command:

OK

Query command:

[50OHMS|HIGHZ]

Example

:AuxCounter:Input:Load 50ohms

OK

:AuxCounter:Input:Load?

50OHMS

:AuxCounter:Input:Slope

This commands sets and queries the active edge of the input signals to be recorded by the auxiliary counter, when its input type is set to variable (see :AuxCounter:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[POSITIVE|NEGATIVE]

Return Value

Setting command:

OK

Query command:

[POSITIVE|NEGATIVE]

Example

:AuxCounter:Input:Slope Negative

OK

:AuxCounter:Input:Slope?

NEGATIVE

:Detector:Count?

This commands queries the number of events registered by the detector counter since the last time it was reset.

Command type

Query only

Return Value

Character string coding an integer.
<INTEGER>
Range: [0 … 4294967295]
Units: Counts

Example

:Detector:Count?

298656

:Detector:Deadtime

This commands sets and queries the deadtime value to be applied on the gate generator after each registered avalanche. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[NONE|1|2|5|10]

Return Value

Setting command:

OK

Query command:

[NONE|1|2|5|10]

Units: Microseconds

Example

:Detector:Deadtime 5

OK

:Detector:Deadtime?

5

:Detector:Frequency?

This commands queries the frequency measured by the detector counter over the previous display refresh period. The first time the command is sent to the SPDM during a refresh period, it returns the frequency measured during the previous period. After that, it will return a “*” character string followed by the remaining acquisition time in the current period. This indicates that the frequency value has not been updated yet and that the value previously read is still up to date.

Command type

Query only

Return Value

The first time a value is read, it consists of a character string coding a real number with fixed number of decimal places.
Subsequently, the SPDM returns a "" string followed by the time until the next acquisition period. This time is returned with one decimal place.*
*[<REAL>|"*X.X"]*
Units: Hz or seconds

Comments

The number of decimal places of the number returned depends on the refresh rate.

Example

:Detector:Frequency?
890
:Detector:Frequency?
*0.8

:Detector:Width

This commands sets and queries the width of the gate applied on the detector. Upon successful setting of a parameter, this command returns a character string containing "OK".

Command type

Setting and Query

Parameters

[2.5|5|20|50|100]

Return Value

Setting command:

OK

Query command:

[2.5|5|20|50|100]

Units: Nanoseconds

Example

:Detector:Width 100
OK
:Detector:Width?
100

:Device:Sense?

This commands is used to verify that a SPDM is connected to the RS-232 port of the computer and turned on. It returns a character string containing *"Ok"* if a SPDM replies and nothing otherwise.

Command type

Query only

Return Value

Ok

Example

:Device:Sense?

OK

Comments

If no SPDM is connected to the computer or if a SPDM is connected but turned off, this command does not return any response .

:Device:Serial?

This command returns the serial number.

Command type

Query only

Return Value

<String>

Example

:Device:Serial?

0319025B010

:Device:Status

This commands sets and queries the status of the SPDM. It can be either “Run” or “Stop”. If the SPDM is in run mode, its counters and frequency meters continuously perform events acquisition. Otherwise, in stop mode, they are freezed. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Set and Query

Parameters

[RUN|STOP]

Return Value

Setting command:

OK

Query command:

[RUN|STOP]

Example

:Device:Status RUN

OK

:Device:Status?

RUN

:Device:SystemState?

This commands queries the state of the SPDM and returns a character string. Immediately after having been turned on, the SPDM performs self-test and is in a “Starting” state. Once this phase has been completed, the SPDM starts cooling the avalanche photodiode to the set temperature. It is in “Cooling” state. Once the set temperature has been reached, the gating of the avalanche starts and the SPDM is in “Operating” state. Finally, if a problem occurs the SPDM’s state changes to “Fatal”.

Command type

Query only

Return Value

[STARTING|COOLING|OPERATING|FATAL]

Example

:Device:SystemState?

OPERATING

:Device:Time?

This commands queries the time, in seconds, since the SPDM was put in “RUN” mode with an accuracy of one tenth of a second.

Command type

Query only

Return Value

Real number with one decimal place.
<REAL>
Range: 0.0 … 359999.8
Units: seconds

Example

:Device:Time?

243.6

Comments

The maximum value of the returned time is 359999.8 seconds. Once the maximum value is reached, the time will start at 0.0 again.

:Display:Brightness

This commands sets and queries the display brightness of the SPDM. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Set and Query

Parameters

[LOW|HIGH|AUTO]

Return Value

Setting command:

OK

Query command:

[LOW|HIGH|AUTO]

Example

:Display:Brightness High

OK

:Display:Brightness?

HIGH

:Display:Mode

This commands sets and queries the display mode used by the SPDM to display its counters and frequency meters on its front panel. Two values appear simultaneously on the front panel display.

The display modes indices correspond to the following options:

Mode 1: Trigger Frequency and Detector Frequency
Mode 2: Detector to Trigger Ratio and Auxiliary Counter Frequency
Mode 3: Detector Frequency and Auxiliary Counter Frequency
Mode 4: Detector Counter and Auxiliary Counter

Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Set and Query

Parameters

[1|2|3|4]

Return Value

Setting command:

OK

Query command:

[1|2|3|4]

Example

:Display:Mode 1

OK

:Display:Mode?

1

:Display:Refresh

This commands sets and queries the refresh rate of the display of the SPDM.
Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Set and Query

Parameters

[0.2|1|2|10|20]

Return Value

Setting command:

OK

Query command:

[0.2|1|2|10|20]

Units: seconds

Example

:Display:Refresh 1

OK

:Display:Refresh?

1

:Firmware:Version?

This commands queries the version number of the firmware currently installed in the SPDM.

Command type

Query only

Return Value

"#.#X"
 # represents a single digit integer ranging between 0 and 9.
X represents a single letter.

Example

:Firmware:Version?

2.0C

:Trigger:Count

This commands queries the number of events registered by the trigger counter since the last time it was reset.

Command type

Query

Return Value

Character string coding an integer.
<INTEGER>
Range: [0 … 4294967295]
Units: Counts

Example

:Trigger:Count?

89243

:Trigger:Delay

This commands sets and queries the delay between a trigger event and a gate. Upon successful setting of a parameter, this command returns a character string containing "OK".

Command type

Setting and Query

Parameters

<REAL>

Range: [0.0 … 25.0]

Step: 0.1

Return Value

Setting command:

OK

Query command:

<REAL>

Range: [0.0 … 25.0]
Step: 0.1
Units: Nanoseconds

Example

:Trigger:Delay 18.6

OK

:Trigger:Delay?

18.6

Comments

The delay setting accuracy is 0.1 ns. Different values are rounded.

:Trigger:Delay:Bypass

This commands sets and queries the status of the delay bypass. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[ON|OFF]

Return Value

Setting command:

OK

Query command:

[ON|OFF]

Example

:Trigger:Delay:Bypass On

OK

:Trigger:Delay:Bypass?

ON

:Trigger:Frequency

This commands queries the frequency measured by the detector counter over the previous display refresh period. The first time the command is sent to the SPDM during a refresh period, it returns the frequency measured during the previous period. After that, it will return a "*" character string followed by the remaining acquisition time in the current period. This indicates that the frequency value has not been updated yet and that the value previously read is still up to date.

Command type

Query only

Return Value

The first time a value is read, it consists of a character string coding a real number with fixed number of decimal places.
Subsequently, the SPDM returns a "" string followed by the time until the next acquisition period. This time is returned with one decimal place.*
*[<REAL>|"*X.X"]*
Units: Hz or seconds

Example

:Trigger:Frequency?

100000

:Trigger:Frequency?

*0.5

Comments

The number of decimal places of the number returned depends on the refresh rate.

:Trigger:Input

This commands sets and queries the input type of the signals that serve as triggers when the trigger source is set to "External". Upon successful setting of a parameter, this command returns a character string containing "OK".

Command type

Setting and Query

Parameters

[NIM|TTL|VAR]

Return Value

Setting command:

OK

Query command:

[NIM|TTL|VAR]

Example

:Trigger:Input NIM

OK

:Trigger:Input?

NIM

:Trigger:Input:Level

This commands sets and queries the input voltage level of the trigger signals, when the trigger source is set to external and the trigger input type variable (see :Trigger:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

<REAL>

Range: [-5.0 … +5.0]

Step: 0.2

Return Value

Setting command:

[OK|Error message]

Query command:

<REAL>

Range: [-5.0 … +5.0]
Step: 0.2
Units: Volt

Example

:Trigger:Input:Level 2.0

OK

:Trigger:Input:Level?

2.0

Comments

The input level setting accuracy is 0.2 V. Different values are rounded.

:Trigger:Input:Load

This commands sets and queries the impedance of the input of the trigger, when the trigger source is set to external and its input type to variable (see :Trigger:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[50OHMS|HIGHZ]

Return Value

Setting command:

OK

Query command:

[50OHMS|HIGHZ]

Example

:Trigger:Input:Load HighZ

OK

:Trigger:Input:Load?

HIGHZ

:Trigger:Input:Slope

This commands sets and queries the active edge of the trigger signals, when the trigger source is set to external and its input type to variable (see :Trigger:Input). Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[POSITIVE|NEGATIVE]

Return Value

Setting command:

OK

Query command:

[POSITIVE|NEGATIVE]

Example

:Trigger:Input:Slope Positive

OK

:Trigger:Input:Slope?

POSITIVE

:Trigger:Rate

This commands sets and queries the trigger rate, when the trigger source is set to internal. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[1|10|100|1000]

Return Value

Setting command:

[OK|Error message]

Query command:

[1|10|100|1000]

Units: kHz

Example

:Trigger:Rate 100

OK

:Trigger:Rate?

100

:Trigger:Source

This commands sets and queries the trigger source. Upon successful setting of a parameter, this command returns a character string containing “OK”.

Command type

Setting and Query

Parameters

[INTERNAL|EXTERNAL]

Return Value

Setting command:

OK

Query command:

[INTERNAL|EXTERNAL]

Example

:Trigger:Source Internal

OK

:Trigger:Source?

INTERNAL

Errors

Introduction

This chapter lists the errors that can be returned by the id 200 SPDM. If a command is sent to the id 200 SPDM and an error occurs, it will return an error message, instead of the message “OK”. This is true both for setting and query commands.

Error messages

ERROR: Invalid command

This error message is returned whenever the id 200 SPDM cannot decode the command.
Check command spelling and try again.

ERROR: Invalid parameter

This error message is returned whenever the value of the parameter of a command sent to the id 200 SPDM is not valid. Check parameter value and command syntax.

ERROR: Internal error

If this error message is returned, please contact technical support and provide information about the configuration and set-up that were used.

ERROR: Fatal error – contact technical support

This message is caused by a hardware failure and is not related to the command sent. An error message will simultaneously be displayed on the front panel display of your id 200 SPDM. Please write down this message and contact technical support.

ERROR: Unknown error

If this error message is returned, please contact technical support and provide information about the configuration and set-up that were used.

Appendix

RS-232C Interface configuration

The following parameters must be selected to configure the RS-232C interface in order to connect a SPDM to a computer.

Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control / handshaking	None

Configuring the Windows Hyperterminal accessory

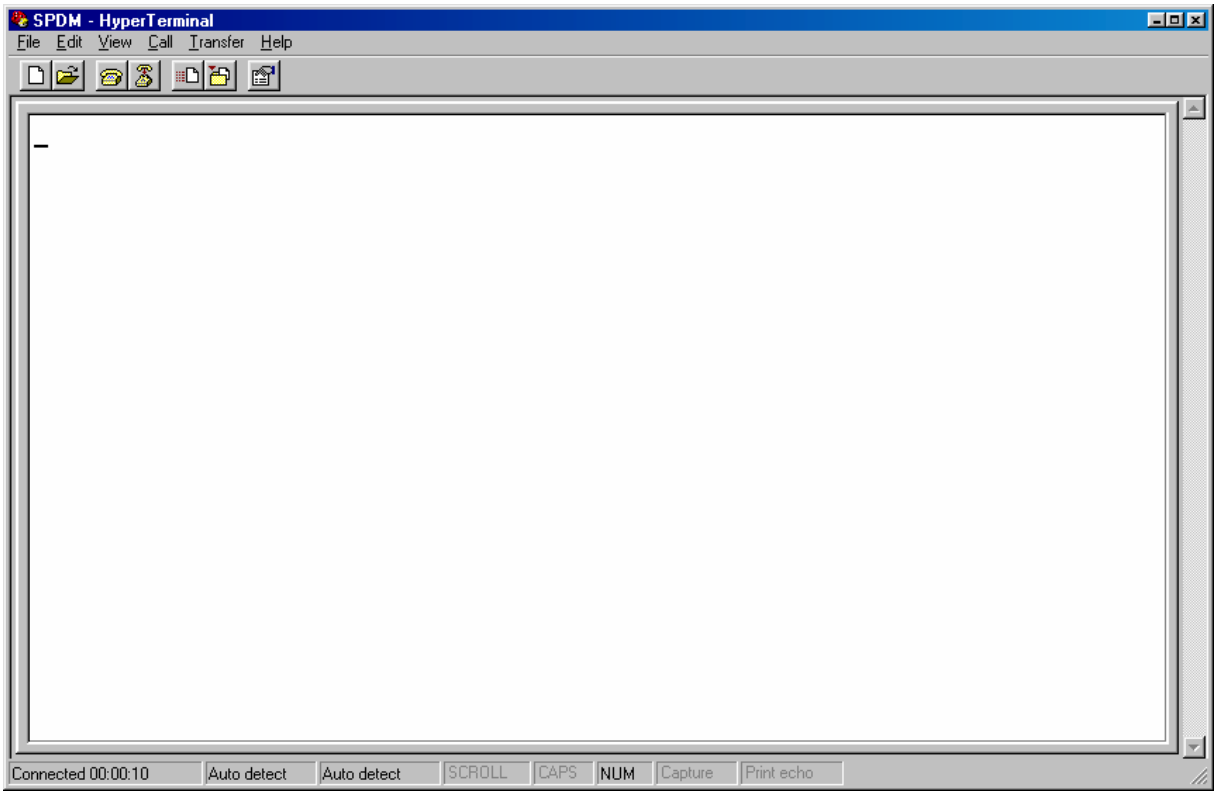
A terminal program can be used to send commands to the SPDM, to read its counters and to query parameters. In this chapter, we will explain how to configure the Windows HyperTerminal program to perform this task and how to open a connection.

Launching the HyperTerminal program

The HyperTerminal is a terminal program that can be found on any Windows platform. To launch it, go into the Start menu, then Programs, then Accessories, and finally Communications. Choose HyperTerminal. The following window will appear.

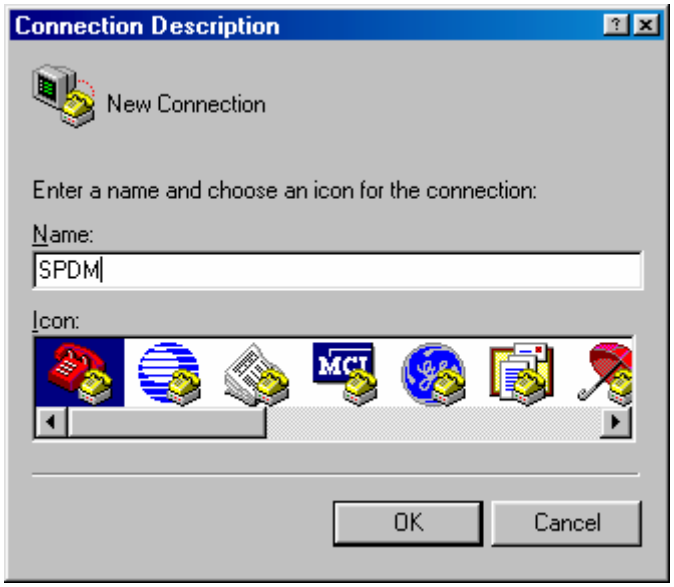
Note

Not only the HyperTerminal application, but any terminal program can be used for this task. The HyperTerminal is presented here as an example, but other programs will work just as well.

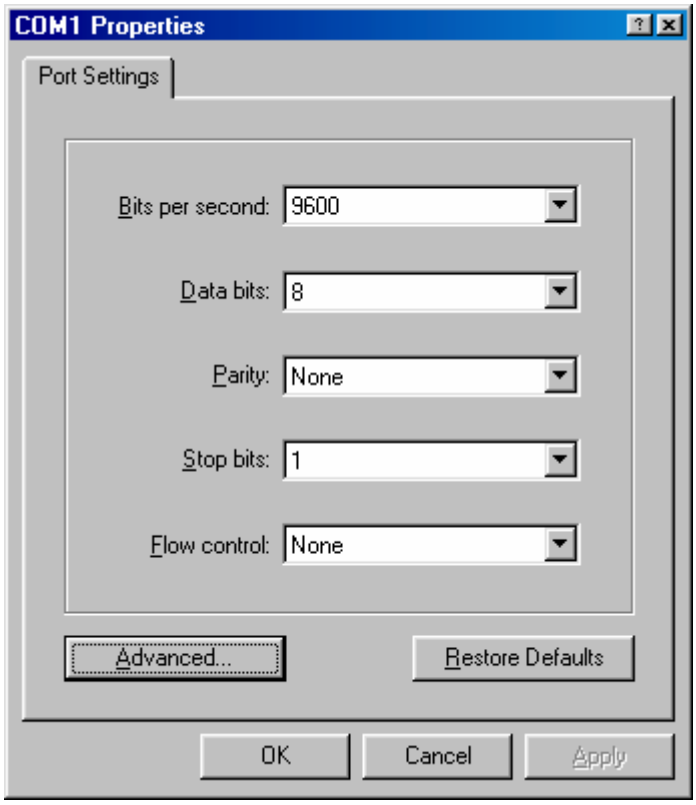


Opening a new connection

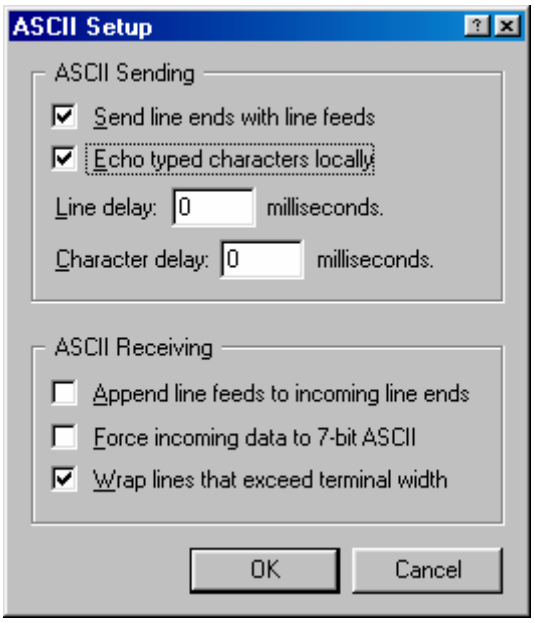
In the HyperTerminal window, pull the File menu down and select New Connection. The following window will appear.



Type a name, like for example SPDM, and click on the OK button. A window will appear and ask you to select the right port to use. Please select the corresponding port – here COM1 – and click on the OK button.
A window will open and ask you to define the properties of the selected port. Make sure that the parameters listed in the previous appendix are selected (see also the next figure) and press OK.



You are now connected to your SPDM and should be able to communicate with it. In order to display this information properly, please pull the File menu down and select the Properties item. Click on the ASCII Setup button. A new window will appear (see next figure). Make sure that the correct boxes are checked and click on the OK button.



You are now ready to send and receive information to and from your SPDM.

Demonstration acquisition program in C++

The following sample C++ program comes with the id 200 SPDM interface package. Both a source and compiled version are supplied. The program reads the detection frequency of the id 200 SPDM and displays it.

The source file is listed here:

```
// RemoteApp.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
using namespace std;

HRESULT DisplayError(const char* errorMessage, HRESULT hr = S_OK)
{
    if(hr == S_OK)
    {
        printf("\n%s.\n", errorMessage);
    }
    else
    {
        LPTSTR lpMsgBuf;

        if(!FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
           FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, hr,
           MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR) &lpMsgBuf, 0, NULL))
        {
            return GetLastError();
        }

        printf("\nError: %s.\nReason: %s\n", errorMessage, lpMsgBuf);
        LocalFree(lpMsgBuf);
    }

    return hr;
}

class CDetectorDevice
{
private:
    HANDLE m_hDevice;

public:
    CDetectorDevice() :
        m_hDevice(INVALID_HANDLE_VALUE)
    {
    }

    ~CDetectorDevice()
    {
        Close();
    }

public:
    HRESULT Open(const char* szPort)
    {
        m_hDevice = CreateFile(szPort, GENERIC_READ | GENERIC_WRITE, 0, NULL,
                               OPEN_EXISTING, 0, NULL);

        if(m_hDevice == INVALID_HANDLE_VALUE)
        {
            return GetLastError();
        }

        DCB dcb;
        ZeroMemory(&dcb, sizeof(dcb));
        dcb.DCBlength = sizeof(dcb);
        GetCommState(m_hDevice, &dcb);

        dcb.BaudRate = 9600;
        dcb.ByteSize = 8;
        dcb.fBinary = true;
        dcb.fParity = false;
        dcb.Parity = NOPARITY;
```

```
        dcb.StopBits = ONESTOPBIT;
        SetCommState(m_hDevice, &dcb);

        COMMTIMEOUTS timeouts;
        timeouts.ReadIntervalTimeout = 100;
        timeouts.ReadTotalTimeoutMultiplier = 2;
        timeouts.ReadTotalTimeoutConstant = 100;
        timeouts.WriteTotalTimeoutMultiplier = 2;
        timeouts.WriteTotalTimeoutConstant = 100;
        SetCommTimeouts(m_hDevice, &timeouts);

        string result;
        HRESULT hr;

        // First send any wrong request to ensure we clear any pending buffer
        // Ingore any error
        hr = SendHighLevelCommand("X", result);

        // Then ensure our instrument answers the SENSE command
        hr = SendHighLevelCommand(":DEVICE:SENSE", result);

        if(hr != S_OK)
        {
            Close();
            return hr;
        }

        if(result.compare("OK") != 0)
        {
            Close();
            return ERROR_DEVICE_NOT_CONNECTED;
        }

        return S_OK;
    }

    void Close()
    {
        if(m_hDevice != INVALID_HANDLE_VALUE)
        {
            CloseHandle(m_hDevice);
            m_hDevice = INVALID_HANDLE_VALUE;
        }
    }

public:
    HRESULT SendHighLevelCommand(string command, string& result)
    {
        DWORD bytesWritten = 0;
        DWORD bytesRead = 0;

        // Add a carriage return to our command
        command.push_back('\r');

        // Send the command over the serial port
        if(!WriteFile(m_hDevice, command.c_str(), (DWORD) command.size(),
           &bytesWritten, NULL))
        {
            return GetLastError();
        }

        // Empty our buffer and reserve at least 200 characters
        result.resize(0);
        result.reserve(200);

        while(1)
        {
            char ch;

            // Read only one char
            if(!ReadFile(m_hDevice, &ch, sizeof(ch), &bytesRead, NULL))
            {
                return GetLastError();
            }

            // If no character was returned, return an error
            if(bytesRead == 0)
```

```
            {
                return ERROR_TIMEOUT;
            }

            // Ignore carriage returns
            if(ch == '\r')
            {
                continue;
            }

            // Stop reading on line feeds
            if(ch == '\n')
            {
                break;
            }

            // Insert received character into our result buffer
            result.push_back(ch);

            if(result.size() >= 200)
            {
                break;
            }
        }

        return S_OK;
    }
};

int main()
{
    CDetectorDevice detector;

    HRESULT hr = detector.Open("COM1:");

    if(hr != S_OK)
    {
        // If the device was not opened properly, exit with an error code
        return DisplayError("Unable to find instrument", hr);
    }

    string result;

    // Loop until user presses a key
    while(!kbhit())
    {
        // In this example, we read and display the measured frequency of the
        // detector
        hr = detector.SendHighLevelCommand(":DETECTOR:FREQUENCY?", result);

        if(hr != S_OK)
        {
            return DisplayError("Unable to send command", hr);
        }

        // Verify if any error occured
        if(result.find("ERROR") == 0)
        {
            return DisplayError(result.c_str());
        }

        // If we didn't receive at least one character, ignore the result
        if(result.size() < 0)
        {
            continue;
        }

        if(result[0] == '*')
        {
            // No result, we have to wait a little bit until the result is ready

            // Remove the asterix
            result.erase(result.begin());

            // Round to 1 second
            size_t pos = result.find('.');
```

```
            if(pos != string::npos)
            {
                result.erase(result.begin() + pos, result.end());
            }

            if(result == "0")
            {
                // Zero seconds left: we do nothing because the result will be
                // printed soon
            }
            else if(result == "1")
            {
                // One second left
                printf("\rNext result available in 1 second   ", result.c_str());
            }
            else
            {
                // Several seconds left
                printf("\rNext result available in %s seconds  ", result.c_str());
            }
        }
        else
        {
            // Display the measured result
            printf("\r%s%-50s\n", result.c_str(), "");
        }
    }

    return 0;
}
```