

Operator's Manual

**Model SM-2020 and SM-2020CT
5½ Digit Digital Multimeters**

Signametrics Corporation

October 1998

CAUTION

In no event shall Signametrics or its Representatives be liable for any consequential damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other loss) arising out of the use of or inability to use Signametric's products, even if Signametrics has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential damages, the above limitations may not apply to you.

© 1994 Signametrics Corp. Printed in the USA. All rights reserved. Contents of this publication must not be reproduced in any form without the permission of Signametrics Corporation.

TABLE OF CONTENTS

1.0 INTRODUCTION.....	6
1.1 SAFETY CONSIDERATIONS.....	6
1.2 MINIMUM REQUIREMENTS	6
2.0 SPECIFICATIONS, SM-2020 AND SM-2020CT	7
2.1 DC VOLTAGE.....	7
2.2 RESISTANCE, 2-WIRE AND 4-WIRE.....	7
2.3 AC VOLTAGE, TRUE RMS.....	9
2.4 DC CURRENT.....	10
2.5 AC CURRENT, TRUE RMS.....	11
2.6 ACCURACY NOTES.....	12
2.7 OTHER SPECIFICATIONS	13
2.8 SM-2020CT SPECIFICATIONS	14
3.0 GETTING STARTED.....	15
3.1 SETTING THE SM-2020 ADDRESS.....	15
3.2 INSTALLING THE SM-2020 BOARD.....	16
3.3 INSTALLING THE SOFTWARE.....	17
3.4 STARTING THE DOS CONTROL PANEL.....	17
4.0 SM-2020 INPUT CONNECTORS.....	18
5.0 MEASUREMENT FUNCTIONS.....	20
6.0 MAKING MEASUREMENTS	23
6.1 VOLTAGE MEASUREMENTS	23
6.2 CURRENT MEASUREMENTS	23
6.3 RESISTANCE MEASUREMENTS	24
7.0 SM-2020 DOS LIBRARY AND INTERFACE.....	25
7.1 DISTRIBUTION FILES	25
7.2 CONFIGURATION FILES	26
7.3 USING THE SM-2020 LIBRARY.....	26
7.4 DOS CONTROL PANEL.....	27
7.5 DOS DEFAULT MODES AND PARAMETERS	27
7.6 MOUSE OPERATION	28
7.7 DOS COMMAND LANGUAGE.....	29
<i>ArmAnalogTrigger (SM-2020CT only)</i>	29
<i>ArmTrigger (SM-2020CT only)</i>	31
<i>CalibrateDmm</i>	32
<i>ClearRelative</i>	32
<i>ClearToFile</i>	33
<i>DisArmTrigger (SM-2020CT only)</i>	33
<i>DmmBufferReady (SM-2020CT only)</i>	34
<i>Dmm_Ready (SM-2020CT only)</i>	35
<i>GetCalDate</i>	35
<i>GetDmmFuncRange</i>	36
<i>GetDmmFunction</i>	36
<i>GetDmmRange</i>	36
<i>GetDmmRate</i>	38
<i>GetDmmResolution</i>	38
<i>GetFreq (SM-2020CT only)</i>	40

<i>GetFreqStr (SM-2020CT only)</i>	40
<i>GetGrdVer</i>	42
<i>GetID</i>	42
<i>GetPeriod (SM-2020CT only)</i>	42
<i>GetPeriodStr (SM-2020CT only)</i>	43
<i>GetVer</i>	43
<i>HasCountTrig (SM-2020CT only)</i>	44
<i>InitDmm</i>	46
<i>IsAuto4WRef</i>	47
<i>IsAutoRange</i>	47
<i>IsRelative</i>	48
<i>IsToFile</i>	48
<i>ReadBuffer (SM-2020CT only)</i>	49
<i>ReadBurstBuffer (SM-2020CT only)</i>	50
<i>ReadDmm, ReadDmmFloat</i>	52
<i>ReadDmmNorm</i>	53
<i>ReadDmmStrg</i>	53
<i>RefMeasure4WOhms</i>	54
<i>SetAuto4WRef, ClearAuto4WRef</i>	54
<i>SetAutoRange, ClearAutoRange</i>	55
<i>SetDmmFuncRange</i>	55
<i>SetDmmFunction</i>	56
<i>SetDmmRange</i>	56
<i>SetDmmRate</i>	56
<i>SetDmmResolution</i>	57
<i>SetRelative</i>	57
<i>SetToFile</i>	58
<i>Trigger (SM-2020CT only)</i>	59

8.0 SM-2020 DMM WINDOWS INTERFACE..... 61

8.1 DISTRIBUTION FILES.....	61
8.2 CONFIGURATION FILES.....	63
8.3 USING THE SM-2020 DRIVER.....	65
8.4 VISUAL BASIC FRONT PANEL APPLICATION.....	65
8.5 WINDOWS DLL DEFAULT MODES AND PARAMETERS.....	67
8.6 USING THE SM2020 DLL WITH LABWINDOWS/CVI® VERSION 3.1.....	67
8.7 WINDOWS COMMAND LANGUAGE.....	70
<i>DMM4WOhmsRefMeasure</i>	70
<i>DMMArmAnalogTrigger (SM-2020CT only)</i>	71
<i>DMMArmTrigger (SM-2020CT only)</i>	73
<i>DMMCalibrate</i>	74
<i>DMMDelay</i>	76
<i>DMMDisArmTrigger (SM-2020CT only)</i>	76
<i>DMMFrequencyStr (SM-2020CT only)</i>	77
<i>DMMGetCalDate</i>	78
<i>DMMGetFnRange</i>	79
<i>DMMGetFunction</i>	81
<i>DMMGetGrdVer</i>	82
<i>DMMGetID</i>	82
<i>DMMGetInfo</i>	84
<i>DMMGetRange</i>	86
<i>DMMGetRate</i>	88
<i>DMMGetResolut</i>	89
<i>DMMGetVer</i>	91
<i>DMMHasCountOption</i>	91
<i>DMMInit</i>	92
<i>DMMInitialize</i>	94
<i>DMMIsAuto4WRef</i>	94

<i>DMMIsAutoRange</i>	96
<i>DMMIsRelative</i>	96
<i>DMMPeriodStr (SM-2020CT only)</i>	97
<i>DMMRead, DMMReadDbl</i>	98
<i>DMMReadBuffer (SM-2020CT only)</i>	99
<i>DMMReadBufferStr (SM-2020CT only)</i>	100
<i>DMMReadFrequency (SM-2020CT only)</i>	101
<i>DMMReadingStr</i>	102
<i>DMMReadNorm</i>	103
<i>DMMReadPeriod (SM-2020CT only)</i>	104
<i>DMMReady (SM-2020CT only)</i>	105
<i>DMMSet4WRef</i>	106
<i>DMMSetAutoRange</i>	107
<i>DMMSetFnRange</i>	108
<i>DMMSetFunction</i>	109
<i>DMMSetRange</i>	110
<i>DMMSetRate</i>	111
<i>DMMSetRelative</i>	112
<i>DMMSetResolut</i>	114
<i>DMMTerminate</i>	115
<i>DMMTrigger (SM-2020CT only)</i>	116
<i>DMMTrigrBufReady (SM-2020CT only)</i>	117
9.0 MAINTENANCE	118
9.1 PERFORMANCE TESTS.....	119
9.2 DC VOLTAGE TEST.....	119
9.3 RESISTANCE TEST, 2-WIRE.....	120
9.4 RESISTANCE TEST, 4-WIRE.....	120
9.5 AC VOLTAGE TEST.....	121
9.6 DC CURRENT TEST.....	122
9.7 AC CURRENT TEST.....	123
9.8 FREQUENCY COUNTER TEST (SM-2020CT ONLY).....	124
9.9 CALIBRATION.....	125
10.0 WARRANTY AND SERVICE	126

1.0 Introduction

Congratulations! You have purchased a Personal Computer (PC) Plug-in instrument with analog and systems performance that rivals the best, all-in-one box, instruments. The SM-2020 series DMMs are easy to setup and use, have sophisticated analog and digital circuitry to give very repeatable measurements, and are protected to handle any unexpected situations your measurement environment may encounter. To get years of reliable service from these DMMs, please take a few moments and review this manual before installing and using this precision instrument.

Note: This manual describes both the SM-2020 and SM-2020CT DMMs. The SM-2020CT is identical to the SM-2020 except for the addition of a Frequency Counter and External triggering. All references to the "SM-2020" also apply to the SM-2020CT. Features unique to the SM-2020CT will be identified as so.

1.1 Safety Considerations

Safety Considerations

The SM-2020 is capable of measuring up to 300 VDC or 250 VAC across the Volt HI and LO terminals, and can also measure common mode signals that "float" the SM-2020 above EARTH ground by up to 300 VDC or 250 VAC. When making common mode measurements, the majority of the circuits inside the SM-2020 are at the common mode voltage. **These voltages can be lethal and can KILL! After installing your SM-2020, check to see that there are wires or ribbon cables from your PC trapped inside the SM-2020.**

The SM-2020 comes installed with three shields (bottom, edge, and top) that **must not be removed for performance as well as safety reasons.** Removal of these shields and/or improper assembly of the shields can result in lethal voltages occurring within your PC. Be sure to check your installation before closing the cover on your personal computer.

Warning

Check to see that no loose wires or ribbon cables infringe upon any of the internal circuits of the SM-2020, as this may apply measurement voltages to your computer!

To avoid shock hazard, install the SM-2020 only into a computer that has its power connector connected to a power receptacle with an earth safety ground.

When making any measurements above 50 VDC or 40 VAC, only use Safety Test Leads. An example of these are the ITT Pomona model 5901 Bench Test Kit, or the ITT Pomona model 55677A DMM Test Lead Kit. ITT Pomona Electronics can be reached at (909) 469-2900. (These test leads are also available from many electronic distributors. They are often referred to as "Fluke Safety DMM leads".)

1.2 Minimum Requirements

The SM-2020 series of system DMMs are precision plug-in boards that are compatible with IBM type personal computers (PCs), from the 286 AT class to the Pentium. They require one full length expansion slot on the ISA bus, either 8 bits or 16 bits. A mouse must be installed. For DOS applications, you will need a minimum of 512k memory and DOS 3.0 or higher. There are no special graphics requirements when using DOS. The SM-2020 also comes with a Windows' DLL, for Windows' Version 3.0 or greater.

2.0 Specifications, SM-2020 and SM-2020CT

2.1 DC Voltage

- > 1000 M Ω Input Resistance, 300 mV & 3 V Ranges
- 10 M Ω Input Resistance, 30 V & 300 V Ranges

\pm (% of reading + Number of counts) [1]

Range	Full Scale 5 1/2 Digits	Resolution	Accuracy 18 $^{\circ}$ C to 28 $^{\circ}$ C		
			One Year	Two Years	Three years
300 mV	300.000 mV	1 μ V	.024 + 6 [2]	.034 + 6 [2]	.042 + 6 [2]
3 V	3.00000 V	10 μ V	.010 + 4	.013 + 6	.017 + 7
30 V	30.0000 V	100 μ V	.022 + 5	.031 + 7	.040 + 8
300 V	300.000 V	1 mV	.018 + 5	.026 + 6	.030 + 7

[1] To convert "Number of counts" to volts, multiply the "Resolution" by the "Number of counts". For example, the 300 mV Range has 6 counts of error, or 6 μ V. The 300 mV Range accuracy for one year is .024% + 6 μ V.

[2] Within one hour of DCV zero, using Relative control.

DCV Noise Rejection Normal Mode Rejection, at 50, 60, or 400 Hz \pm 0.5%, is better than 80 dB for readings of 10 or slower. Common Mode Rejection (with 1 k Ω lead imbalance) is better than 120 dB for these same conditions.

2.2 Resistance, 2-wire and 4-wire

\pm (% of reading + Number of counts) [3]

Range	Full Scale 5 1/2 Digits	Resolution	Accuracy 18 $^{\circ}$ C to 28 $^{\circ}$ C		
			One Year	Two Years	Three Years
300 Ω	300.000 Ω	.001 Ω	.030 + 6 [4]	.050 + 6 [4]	.060 + 6 [4]
3 k Ω	3.00000 k Ω	.01 Ω	.029 + 4 [4]	.040 + 5 [4]	.050 + 6 [4]
30 k Ω	30.0000 k Ω	.1 Ω	.029 + 5	.040 + 6	.050 + 6
300 k Ω	300.000 k Ω	1 Ω	.029 + 5	.040 + 6	.050 + 6
3 M Ω [5]	3.00000 M Ω	10 Ω	.190 + 10	.280 + 10	.330 + 10
30 M Ω [5]	30.0000 M Ω	100 Ω	.650 + 10	.700 + 10	.750 + 10

[3] To convert "Number of counts" to ohms, multiply the "Resolution" by the "Number of counts". For example, the 300 Ω Range has a One Year Accuracy of .030% + .006 Ω .

[4] Within one hour of Ohms zero, using Relative control.

[5] For 2-wire Ohms only.

The SM-2020 measures resistance by sourcing a constant current, and then measuring the resultant voltage across the resistor. The value of this constant current is shown in the table below.

Resistance Range	Current Level
300 Ω	1 mA
3 k Ω	1 mA
30 k Ω	100 μ A
300 k Ω	10 μ A
3 M Ω	1 μ A
30 M Ω	122 nA

Using 4-wire Ohms 4-wire Ohms is used to make very repeatable low ohms measurements, with practical usage down to 1 m Ω . The **Voltage (V,W)** Inputs are the "Source" input terminals (i.e. they provide the current stimulus in the ohms measurement), and the **I, 4W** Inputs are the "Sense" inputs. The Source Hi and Sense Hi leads are connected to one side of the resistor, and the Source LO and Sense LO leads are connected to the other side of the resistor. Both Sense leads should be closest to the body of the resistor.

4-wire Ohms measurements are advantageous for making measurements 100 k Ω and below. Using this method for resistors above 300 k Ω is not recommended, as the extra set of inputs can actually *degrade* the accuracy, due to additional leakage paths.

4-wire Ohms reading rates are normally 1/10th the rate of all other functions. To speed this up, you can make a 3-wire Ohms measurement. See **SetAuto4WRef** in Section 7 and **DMMSet4WRef** in Section 8 for more details.

2.3 AC Voltage, True RMS

- **Input Resistance** 1 M Ω , shunted by < 80 pF, all ranges
- **Crest Factor** 3 at Full Scale, increasing to 7 at Lowest Specified Voltage

Range	Full Scale 5 1/2 Digits	Lowest Specified Voltage	Resolution
300 mV	300.000 mV	5.000 mV [6]	1 μ V
3 V	3.00000 V	10.00 mV	10 μ V
30 V	30.0000 V	100.0 mV	100 μ V
250 V [7]	250.000 V	1.000 V	1 mV

[6] Between 5 mV and 10 mV add 200 counts of error to the table below. In many computer installations, if the SM-2020 is not near a noisy board, usable voltage measurements of 1.000 mV can be obtained. Please contact *Signametrics* for your special requirements.

[7] Limited to 8×10^6 Volt x Hz.

Range	Frequency	Accuracy 18 $^{\circ}$ C to 28 $^{\circ}$ C [9]		
		One Year	Two Years	Three Years
300 mV	10 Hz - 20 Hz	2.5 + 400	2.8 + 400	2.9 + 400
	20 Hz - 45 Hz	1.0 + 400	1.2 + 400	1.3 + 400
	45 Hz - 1 kHz	0.3 + 300	0.4 + 300	0.5 + 300
	1 kHz - 10 kHz	0.4 + 350	0.5 + 400	0.6 + 400
	10 kHz - 50 kHz	2.0 + 400	2.1 + 400	2.2 + 400
	50 kHz - 100 kHz	4.0 + 500	4.3 + 500	4.5 + 500
3 V - 250 V[10]	10 Hz - 20 Hz	2.5 + 300	2.6 + 300	2.7 + 300
	20 Hz - 45 Hz	1.0 + 300	1.2 + 300	1.3 + 300
	45 Hz - 1 kHz	0.1 + 275	0.2 + 275	0.3 + 275
	1 kHz - 10 kHz	0.3 + 275	0.4 + 275	0.5 + 275
	10 kHz - 50 kHz	1.0 + 300	1.1 + 300	1.2 + 300
	50 kHz - 100 kHz	4.0 + 500	4.3 + 500	4.5 + 500

[8] To convert "Number of counts" to volts, multiply the "Resolution" by the "Number of counts". For example, the 300 mV Range has a One Year Accuracy of 0.3% + 300 μ V from 45 Hz to 1 kHz

[9] For sinewave inputs.

[10] Limited to 8×10^6 Volt x Hz. For example, the largest frequency input at 250 V is 32 kHz, or 8×10^6 .

ACV Noise Rejection Common Mode rejection, for 50 Hz or 60 Hz with 1 k Ω imbalance in either lead is better than 60 dB.

2.4 DC Current

- **Burden Voltage** < 0.5 V for the 3 mA and 30 mA Range, <1.2 V for the 300 mA Range
- **Protected** with 500 mA fuse (5x20mm, 250 V Fast)

± (% of reading + Number of counts) [11]

Range	Full Scale 5 1/2 Digits	Resolution	Accuracy 18°C to 28°C [12]		
			One Year	Two Years	Three Years
3 mA	3.00000 mA	10 nA	.070 + 8	.090 + 8	.120 + 8
30 mA	30.0000 mA	100 nA	.090 + 15	.120 + 17	.140 + 17
300 mA	300.000 mA	1 uA	.090 + 15	.120 + 17	.140 + 17

[11] To convert "Number of counts" to amps, multiply the "Resolution" by the "Number of counts". For example, the 3 mA Range has a One Year Accuracy of 0.070% + 80 nA.

[12] Within one hour of DC Current zero, using Relative control.

2.5 AC Current, True RMS

Input Characteristics

- **Burden Voltage** < 0.5 V RMS for the 3 mA and 30 mA Range, <1.2 V RMS for the 300 mA Range
- **Crest Factor** 3 at Full Scale, increasing to 7 at Lowest Specified Current
- **Protected** with 500 mA fuse (5x20mm, 250 V Fast)

Range	Full Scale 5 1/2 Digits	Lowest Specified Current	Resolution
3 mA	3.00000 mA	.05000 mA	10 nA
30 mA	30.0000 mA	.5000 mA	100 nA
300 mA	300.000 mA	5.000 mA	1 uA

± (% of reading + Number of counts) [13]

Range	Frequency	Accuracy 18°C to 28°C		
		One Year	Two Years	Three Years
3 mA	10 Hz - 20 Hz	2.2 + 400	2.3 + 400	2.4 + 400
	20 Hz - 45 Hz	0.8 + 400	0.9 + 400	1.0 + 400
	45 Hz - 1 kHz	0.7 + 300	0.8 + 300	0.9 + 300
	1 kHz - 10 kHz [14]	0.8 + 400	0.9 + 400	1.0 + 400
30-300 mA	10 Hz - 20 Hz	2.0 + 300	2.1 + 300	2.2 + 300
	20 Hz - 45 Hz	0.8 + 300	0.9 + 300	1.0 + 300
	45 Hz - 1 kHz	0.7 + 300	0.8 + 300	0.9 + 300
	1 kHz - 10 kHz [14]	0.8 + 400	0.9 + 400	1.0 + 400

[13] To convert "Number of counts" to amps, multiply the "Resolution" by the "Number of counts". For example, the 3 mA Range has a One Year Accuracy of 0.7% + 3000 nA from 45 Hz to 1 kHz.

[14] Typically to 20 kHz

2.6 Accuracy Notes

Important All accuracy specifications for DCV, Resistance, DCI, ACV, and ACI apply for the time periods shown assuming the use of System Calibration once every month. (System Calibration is a simple software operation that takes less than 3 seconds to perform. It is performed by calling **S_CAL** on the DOS Control Panel, or from the appropriate DOS or Windows command. See Sections 7, "SM-2020 DOS Library and Interface" and Section 8, "SM-2020 DMM Windows Interface" for more details.)

Accuracy vs. Reading Rates All of the above specifications apply to reading rates of less than 10 readings per second (one reading per second for 4W Ω). For higher reading rates, increase the noise floor for DCV, Resistance, and DCI by the square root of the increase in reading rate from 5 readings per second (rps). For example, the noise floor for the 3 VDC range is 4 counts at 5 rps. At 50 readings per second, or 10x the reading rate, the noise increases by the square root of 10, or 3.16 times. The noise, then, at 50 readings per second is ± 13 counts.

The noise characteristics for the AC functions increases by the same number of counts as the DC functions. For example, the noise floor for 3 VAC, 50 rps, will have digit rattle of an additional ± 13 counts.

Reading Rates vs. Noise Rejection The best AC (50 Hz or 60 Hz) line rejection is obtained at reading rates that are whole number divisions greater than 1 of the line frequency, as shown in the following table. For best AC line rejection you should use the reading rates checked.

Reading Rate (rps)	50 Hz	60 Hz
1	√	√
2	√	√
5	√	√
10	√	√
15		√
20		√
25	√	
30		√
50	√	
60		√

Reading Rates vs. Digits of Resolution For reading rates of 60 readings per second and slower, the SM-2020 has 5 1/2 digits of resolution. For reading rates of 80 and 100 readings per second, the SM-2020 has 4 1/2 digits of resolution. For reading rates > 100 readings per second, there are 3 1/2 digits of resolution.

2.7 Other Specifications

Temperature Coefficient, All Functions	Less than 0.1 x accuracy specification per °C from 0°C to 18°C and 28°C to 50°C.
Reading Rate	1 to 200 readings/sec (0.1 to 20 rps for 4WΩ)
Overload Protection (voltage inputs)	300 VDC, 250 VAC
Isolation	300 VDC, 250 VAC from Earth Ground
Maximum Input (Volt x Hertz)	8x10 ⁶ Volt x Hz normal mode input (across Voltage HI & LO). 1x10 ⁶ Volt x Hz Common Mode input (from Voltage HI or LO relative to Earth Ground).
Safety	Designed to IEC 1010-1 guidelines, for IEC 664 Installation Category I. All DMM inputs are to be used in protected, low energy circuits only.
Calibration	Calibrations are performed by <i>Signametrics</i> in a computer with a 10°C internal temperature rise. All calibration constants are stored in a ASCII data file.
Temperature Range	0°C to 50°C, operating
Power	+5 volts, 300 mA maximum

2.8 SM-2020CT Specifications

The SM-2020CT is identical to the SM-2020 with the addition of a Frequency Counter and External Triggering capability. Specifications for these features follows.

Frequency Measurement, for AC Voltage Input

Input Impedance: 1 M Ω with < 80 pF

Frequency Range	2 Hz - 100 Hz	100 Hz-1 kHz	1 kHz-10 kHz	10 kHz-100 kHz	100 kHz-300 kHz
Resolution	1 mHz	10 mHz	100 mHz	1 Hz	1 Hz
Uncertainty is $\pm 0.01\%$ of reading \pm adder shown	4 mHz [1]	20 mHz	200 mHz	2 Hz	2 Hz
Input Signal Range [2]	10% - 110% of range	10% - 110% of range	10% -110% of range	10% - 110% of range	50% -110% of range

[1] For input signals less than 100 mV, the adder is 8 mHz.

[2] Input voltage required to give a valid reading. For example, 10% -110% of range indicates that in the 300 mVAC range, the input voltage should be 30 mV to 330 mV.

Frequency Measurement, for AC Current Input

Input Impedance: 100 Ω in the 3 mA range, 2 Ω in the 30 mA and 300 mA ranges.

Frequency Range	2 Hz - 100 Hz	100 Hz-1 kHz	1 kHz-10 kHz	10 kHz-30 kHz
Resolution	1 mHz	10 mHz	100 mHz	1 Hz
Uncertainty is $\pm 0.01\%$ of reading \pm adder shown	4 mHz	20 mHz	200 mHz	2 Hz
Input Signal Range, 3 mA Range [3]	10% -500% of range	10% - 500% of range	10% -500% of range	10% - 500% of range
Input Signal Range, 30 mA Range	50% - 500% of range	50% - 500% of range	50% -500% of range	50% -500% of range
Input Signal Range, 300 mA Range	50% -110% of range	50% - 110% of range	50% - 110% of range	50% - 110% of range

[3] Input current required to give a valid reading. For example, 10% -500% of range indicates that in the 3 mA range, the input current should be 0.3 mA to 15 mA.

External Hardware Trigger

Input voltage levels (to trigger)	+3 V to +15 V, or -3 V to -15 V
Minimum current (to trigger)	1 mA
Timing Characteristics	Trigger occurs within $2 \div$ Reading rate
Internal Reading Buffer	Unlimited for reading rates \leq 200 rps. 64 readings for readings rates > 200 rps.
Isolation of trigger input	± 300 V from analog DMM inputs. ± 15 V from computer chassis earth ground.

All other specifications (DC Voltage, 2-wire and 4-wire Resistance, AC Voltage, DC Current, and AC Current) are the same as the SM-2020.

Note: *Signametrics* reserves the right to make changes in materials, specifications or accessories without notice.

3.0 Getting Started

After unpacking the SM-2020, please inspect for any shipping damage that may have occurred, and report any claims to your transportation carrier.

The SM-2020 is shipped with the SM-2020 PC System Digital Multimeter; a 5 1/4" floppy disk which contains a DOS and Windows Control Panel, "C" drivers, Windows drivers, and calibration data; and this Operator's manual.

3.1 Setting the SM-2020 Address

Before installing the SM-2020, you must first set the base address to one that does not conflict with any other PC cards. The factory default is 0x300, or HEX 300. The address is determined by the set of switches located near the bottom of the DMM. (Bit "H" is the MSB.) The SM-2020 occupies four I/O locations, with the two least significant (LS) address bits assumed to be 0. Therefore, the *two LS address bits are not* part of the hardware switch when determining the HEX address. The hardware address must also match the contents of the configuration file **DMM.CFG** (see below). (The SM-2020 does not use any interrupts from the PC bus. There is no need to set one.)

Examples:

0x300 has a switch pattern of "11000000", corresponding to a **DMM.CFG** file of "2020 0x300". 0x200 has a pattern of "10000000". 0x304 has a pattern of "11000001". After changing the switch setting, the **DMM.CFG** file must be changed to have the same address as indicated by the switch pattern. For example, if you change the pattern to "11000001", the contents of **DMM.CFG** should be changed to "2020 0x304". Use any ASCII text editor to do so.

Note: The SM-2020CT **DMM.CFG** file has the same format, but the identifier is "2021" instead of "2020".

Tip: If you need assistance converting a HEX number to BINARY, **Windows 3.1 Accessories Group** has a **Calculator** that has this capability. Be sure to set the View to Scientific in the Calculator.

Version 2.10 and Later Software

Version 2.10 and later software can support up to four DMMs in one chassis. To do so, the **DMM.CFG** file must contain the base address of each DMM as well as the DMM type. No empty lines are allowed between each record. The following **DMM.CFG** contents indicates DMM number 0 (the first DMM) is a SM-2020 at Hex address 0x204, and DMM number 1 is a SM-2020CT at 0x300:

```
2020 0x204
2021 0x300
```

An appropriate **DMM.CFG** file is created during the Windows **SETUP** of the DMM software. You can also easily modify the **DMM.CFG** file using any ASCII text editor.

The **SM20CAL.DAT** file supplied with your DMM has a unique calibration record for that DMM. (See "**Calibration**" at the end of this manual.) When using multiple DMMs, the **SM20CAL.DAT** file must have a calibration record for each DMM. You can combine the unique calibration records of each DMM into one **SM20CAL.DAT** file using any ASCII text editor. Be certain there are no empty lines between calibration records. The first record is for DMM number 0 (the first DMM), the second record is for DMM number 1 (the second DMM), etc.

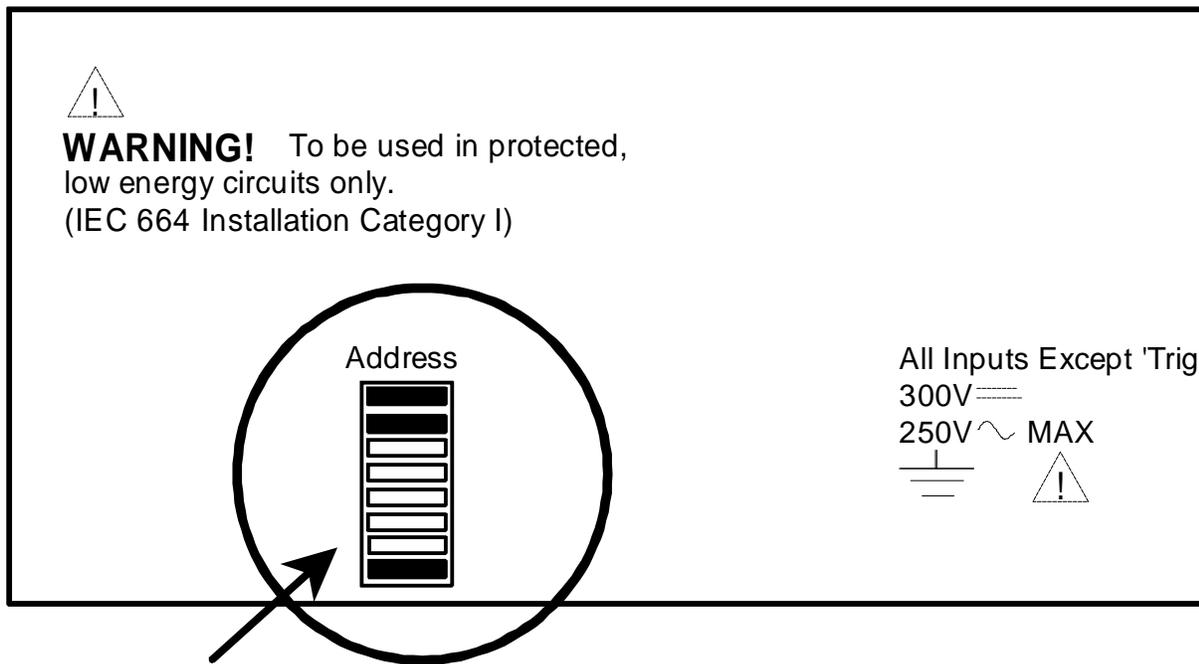


Figure 3-1. Base address set to 0x304. (Dark switch position corresponds to "1".) The **DMM.CFG** file should contain "2020 0x304" for SM-2020 and "2021 0x304" for SM-2020CT.

3.2 Installing the SM-2020 Board

Warning

To avoid shock hazard, install the SM-2020 only into a personal computer that has its power line connector connected to an AC receptacle with an Earth Safety ground.

After installation, check to see that no loose wires or ribbon cables infringe upon any of the internal circuits of the SM-2020, as this may apply measurement voltages to your computer!

After setting the address, carefully plug the SM-2020 into any 8 or 16 bit full length bus. If possible, choose an empty slot away from any high speed boards (e.g. video cards) or the power supply. **Please be patient during the installation process!** The SM-2020 comes with 4 safety input jacks. Because of their necessary size, they are a tight fit in many PC chassis. (This is especially true for IBM AT and Gateway computers.) Insert the bracket end of the SM-2020 into your PC first, watching for any interference between the safety input jacks and your PC chassis. "Walking" each end of the SM-2020 into the chassis may be helpful. **Be patient! You should only have to install it once!**

3.3 Installing the Software

Using DOS: Insert the SM-2020 floppy disk into your drive A. Type **A:DOSSETUP**, which will copy all files from the distribution disk **DOS_STUF** directory into a new directory on your hard drive called **C:\SM2020D**.

Using Microsoft Windows: Insert the SM-2020 floppy disk into your drive A. From the File Manager, choose the A: drive and run "**SETUP**".

Note: If you will be using both DOS and Windows, you must install both DOS and Windows files separately, as described above.

3.4 Starting the DOS Control Panel

You can gain familiarity with the SM-2020 by exercising its measurement functions using the DOS Control Panel. (Remember, your computer must have a mouse installed.) Go into the directory that contains your SM-2020 files (generally **C:\SM2020D**), and run the program "**SM2020D.EXE**". The DMM's relays should click, and the Control Panel (shown in the next section) should appear. If you do not hear the relays click, it is most likely due to a conflict of the SM-2020 address and another card. Change the base address of the SM-2020, and try again. You need to also change the file **DMM.CFG** to reflect the new base address. (See "Setting the SM-2020 Address" above.) Both **SM20CAL.DAT** and **DMM.CFG** should reside in the same directory as **SM2020D.EXE**.

The DOS Control Panel is operated with a mouse. All functions are accessed using the left mouse button. When the SM-2020 is operated at very slow reading rates, you may have to hold down the left mouse button longer than usual for the program to acknowledge the mouse click.

Note: The SM-2020 front panel powers up in DCV, 5 readings per second, 5 1/2 digits, Autorange. Because the input impedance of the 300 mV and 3V DC ranges are virtually infinite, an open input will not read 0 volts. It will instead read whatever charge is associated with the signal conditioning of the DMM. As this electrical charge changes, the SM-2020 will Autorange to the appropriate range. Hence, with an open input, you may hear the SM-2020 relays clicking every few seconds, as the range change occurs. This is perfectly normal.

4.0 SM-2020 Input Connectors

Before using the SM-2020, please take a few moments and review this section to understand where the voltage, current, or resistance inputs should be applied. **This section contains important information concerning voltage and current limits. Do not exceed these limits, as personal injury or damage to the instrument may result.**

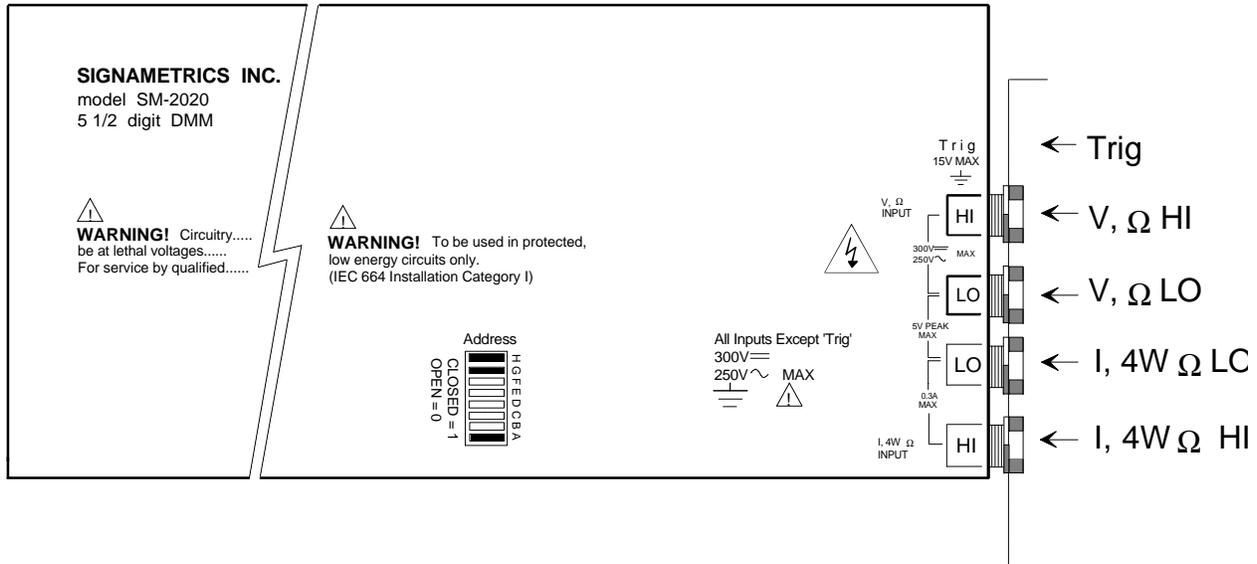


Figure 4-1. The SM-2020 input connectors.

V, W HI This is the positive terminal for all Volts and $2W\Omega$ measurements. It is also the Source HI for $4W\Omega$ measurements. The maximum input across **V, W HI** and **V, W LO** is 300 VDC or 250 VAC.

V, W LO This is the negative terminal for all Volts and $2W\Omega$ measurements. It is also the Source LO for $4W\Omega$ measurements. **Do not float this terminal or any other SM-2020 terminal more than 300 VDC or 250 VAC above Earth Ground.** (Also, see **Trig** below.)

I, 4WW HI This is the positive terminal for all Current measurements. It is also the Sense HI for $4W\Omega$ measurements. The maximum input across **I, 4WW HI** and **I, 4WW LO** is 300 mA. Do not apply more than 40 V peak across these two terminals!

I, 4WW LO This is the negative terminal for all Current measurements. In the Current modes, it is protected with a **1/2A, 250 V Fast Blow fuse** (5 x 20 mm). It is also the Sense LO for $4W\Omega$ measurements. **V, W LO** and **I, 4W W LO** should never have more than 5 V peak across them.

Trig (SM-2020CT only) This is used to externally trigger a reading from the SM-2020CT. It uses a miniature 2.5 mm two pin phone jack, available at many electronic hardware distributors. The outer conductor (attachment nut to the metal bracket) is at Earth Ground potential. The other conductor has no polarity requirement -- you can use either a positive or negative trigger. The trigger signal should be in the range of 3 V to 15 V peak. **The maximum voltage above Earth Ground of the trigger conductor should not exceed 15 V peak.**

5.0 Measurement Functions

All of the SM-2020 measurement functions are accessible from the DOS Control Panel (Figure 5.1), the Windows Control Panel (Figure 5.1B), from the Windows DLL drivers or the DOS library . (To gain familiarity with the SM-2020, run the Windows 3.1 or Windows 95 'setup.exe' to install and run the DMM, as described in the previous section. This section describes the SM-2020's capabilities, using the DOS and Windows control panels, to guide you through each function. The features of the SM-2020CT are described as well, using the Windows control panel.

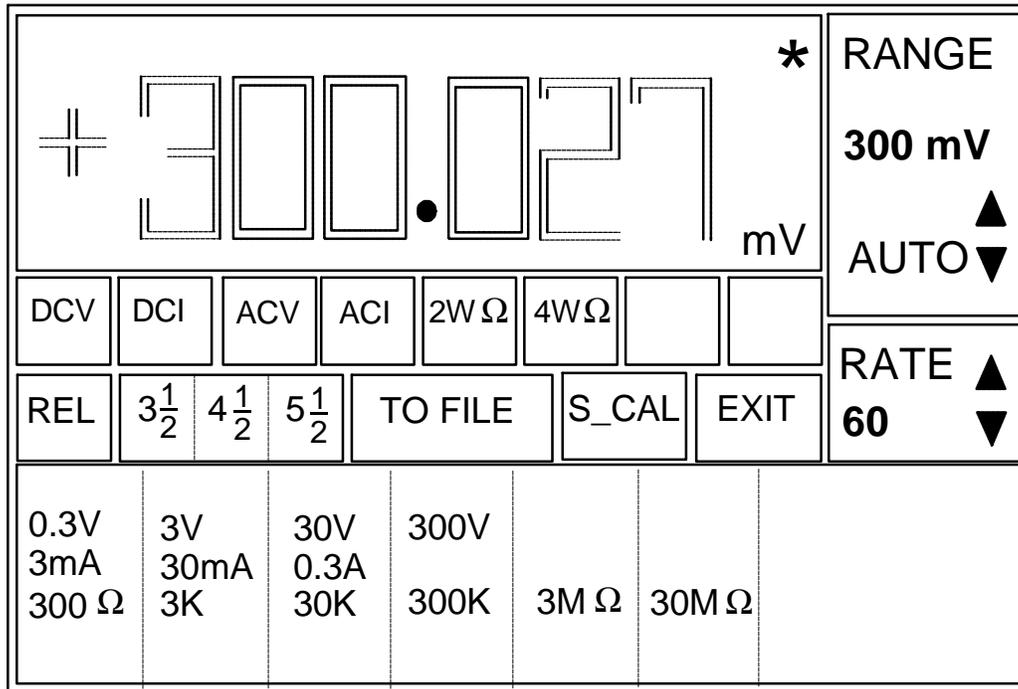


Figure 5-1. The DOS Control Panel for the SM-2020. (The SM-2020CT has a very similar panel. See the text below.) The bottom rows of Ranges are actually context sensitive such that only "0.3V 3V 30V 300V" appear when in Voltage Functions, "3mA 30mA 0.3A" appear when in Current Functions, etc.

DCV Measures from 1 μ V to 300 V, at 1 reading per second (rps) to 200 rps. As you change the reading rate, the resolution automatically changes to give you optimum results. Use the Volts, Ω (V,W) HI & LO terminals, being certain to always leave the I, 4WW HI and LO terminals unconnected.

DCI Measures from 10 nA to 300 mA, at 1 rps to 200 rps. Use the I, 4WW terminals, being certain to always leave the V,W HI & LO terminals unconnected.

ACV Measures from 1 mV (with 1 μ V resolution) to 250 V, from 10 Hz to 100 kHz. The ACV function is AC coupled, and measures the true RMS value of the wave form. As with virtually all true RMS measuring meters, the SM-2020 may not read a perfect zero with a shorted input. This is normal. The reading rates are from 1 rps to 200 rps. Use the V,W HI & LO terminals, being certain to always disconnect the I, 4WW HI and LO terminals.

ACI Measures from 50 μ A (with 10 nA resolution) to 300 mA, at 1 rps to 200 rps. Use the I, 4WW terminals, being certain to always disconnect the V,W HI & LO terminals.

2WW Measures from 1 mΩ to 30 MΩ, at 1 rps to 200 rps. Use the **V,W HI & LO** terminals, being certain to always disconnect the **I, 4WW HI** and LO terminals.

4WW Measures from 1 mΩ to 30 MΩ, at 0.1 rps to 20 rps. Use the **V,W HI & LO** terminals (these are the "Source" terminals) in conjunction with the **I, 4WW HI & LO** terminals (these are the "Sense" terminals). The Source HI and Sense HI leads are connected to one side of the resistor, and the Source LO and Sense LO leads are connected to the other side. Both Sense leads are closest to the body of the resistor. We do not recommend using **4WW** when making measurements above 300 kΩ.

Freq (SM-2020CT only) Not shown in the DOS control panel above, this activates the Frequency counter. Measures from 3 Hz to 300 kHz. When activated, the control panel alternately updates the amplitude reading followed by the frequency reading. Note that the reading rate is slower than indicated when frequency is activated. In the Windows control panel, the inverse of frequency, period (**Per**), is also selectable. The DOS control panel does not have the Period function.

RELATIVE This is the Relative function. When activated, the last reading is stored and subtracted from all subsequent readings. This is a very important function when making low level DCV measurements, or in 2WΩ. For example, when using 2WΩ, you can null out lead resistance by shorting the leads together and clicking on **REL**. When making low level DC voltage measurements (in the uV region), first apply a copper short to the **V,W HI & LO** input terminals, allow the reading to stabilize for a few seconds, and click on **RELATIVE**. This will correct for any offsets internal to the SM-2020.

3 1/2, 4 1/2, 5 1/2 (DOS panel only) Changes the resolution of the SM-2020. For 60 readings per second (rps) or lower, the SM-2020 has 5 1/2 digits of resolution. For 80 rps to 100 rps, the SM-2020 has 4 1/2 digits of resolution. For > 100 rps, there are 3 1/2 digits of resolution.

RATE Controls the SM-2020 reading rate. Changing the RATE upwards will also automatically change the resolution commensurate with the resolution available. Changing the RATE downwards will not automatically change the resolution.

RANGE Can be set to **AUTO** or manual by clicking on the up or down arrows of the DOS panel, or by pressing the appropriate range button in the lower part of the Windows panel..

To File This stores readings into a DOS file called **DMM.DAT** with a time stamp on each reading. Clicking on this box again closes the file and restores the Control Panel to normal operation.

S_Cal or Self Cal. This function is the System Calibration that corrects for internal gain and scale factor errors. It is required at least once every month to meet the SM-2020 accuracy specifications. We recommend that you also perform this function whenever there are external environmental changes (e.g. the temperature in your work environment increases by more than 5°C). This function takes less than a few seconds to perform.

.3V, 3V, 30V, 300V, etc. This row of buttons contains the ranges of the SM-2020. Clicking on any box will change the SM-2020 to manual ranging, and will lock the DMM in that range, canceling auto-range mode. The labeling on this bottom row of buttons changes as you change the SM-2020 functions.

Locking a range is useful in many applications, especially to speed up measurements in a system. Another reason to lock a range is to control the input impedance in DCV. The 300 mV and 3 V ranges have virtually infinite input impedance, while the 30 V and 300 V ranges have 10 M Ω input impedance.

You may also want to control the Ohms current used in making resistance measurements. (See Section 2.2 "Resistance, 2-wire and 4-wire" for a table of resistance range vs. current level.) All of the Ohms ranges of the SM-2020 have enough current and voltage compliance to turn on diode junctions. For characterizing semiconductor part types, performing diode tests in a specific range may be helpful.

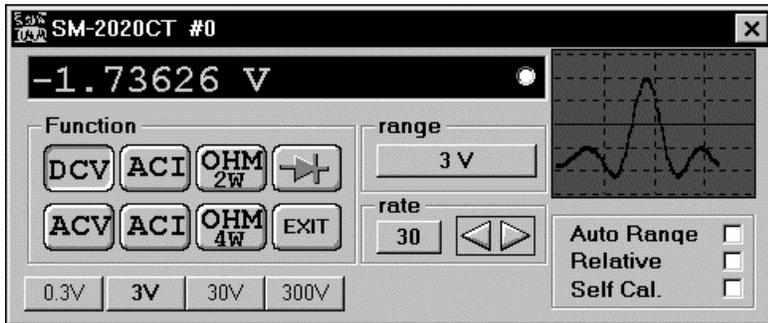


Figure 5-1B. The Windows Control Panel for the SM-2020. The SM-2020CT panel adds the "Per", and "Freq" check boxes whenever ACV or ACI functions are selected.

6.0 Making Measurements

6.1 Voltage Measurements

Making Voltage Measurements is straightforward. There are a few tips you should follow when making voltage measurements:

1) ACV measurements, if possible, should have the NEUTRAL or GROUND attached to the SM-2020 **V,W LO** terminal. See Figure 6-1, below. This prevents any "Common Mode" problems from occurring ("Common Mode" refers to floating the SM-2020 **V,W LO** above Earth Ground.) Common Mode problems can result in noisy readings, or even cause the PC to hang-up under high V x Hz input conditions. In many systems, grounding the source to be measured at Earth Ground (being certain to avoid any ground loops) can give better results.

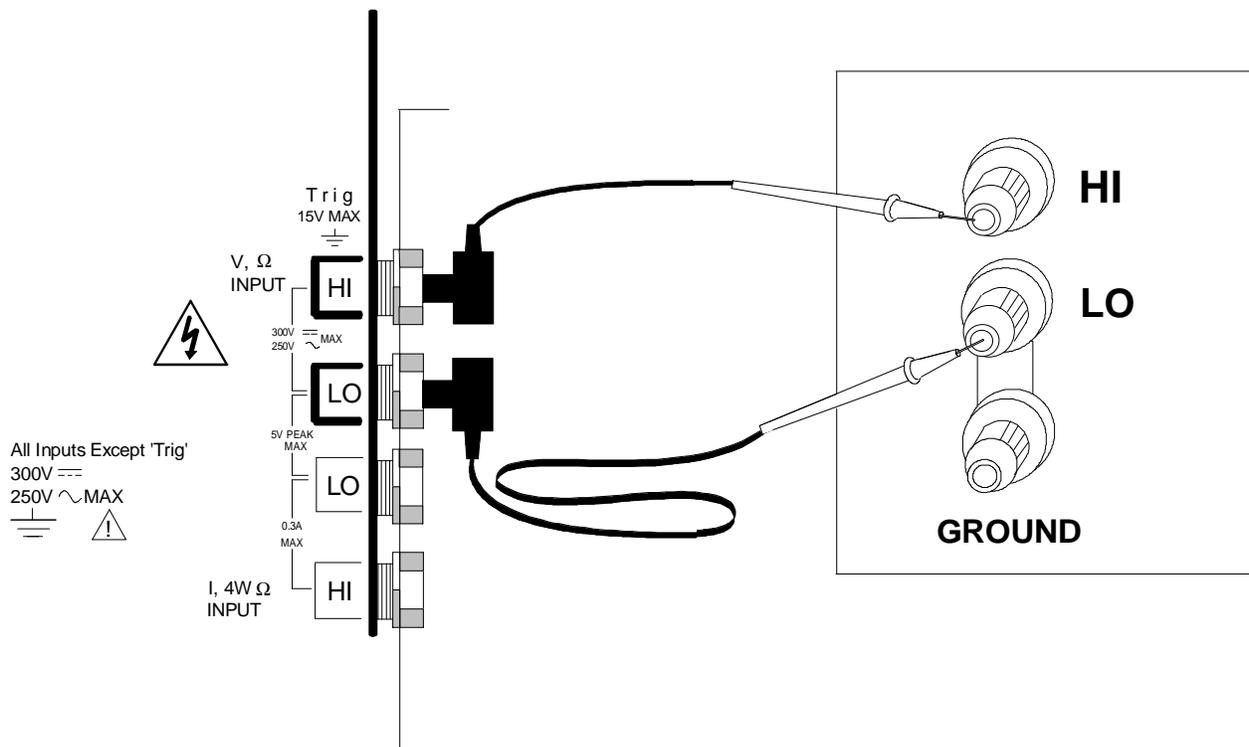


Figure 6-1. Make ACV measurements with the source ground attached to the SM-2020 **V,W LO** to minimize "Common Mode" measurement problems.

2) When making very low level DCV measurements (<100 μ V), you should first short the SM-2020 with a copper wire shorting plug and perform **REL** to eliminate zero errors before making your measurements. Also be aware that your test leads can introduce several μ Volts of error, if they have high thermal voltages relative to copper. To minimize thermal voltaic effects after handling the test leads, you should wait a few seconds before making measurements.

6.2 Current Measurements

Always remember that the Current functions are protected with a 0.5A, 250V fuse. Applying high voltages to the **I,4WW HI & LO** inputs can be dangerous! Think before applying any inputs to these terminals!

When making sensitive DC current measurements, be sure to use the **REL** function to zero out any residual errors of the SM-2020. This is easily accomplished by opening all inputs to the SM-2020 and performing **REL** in the appropriate DCI range.

6.3 Resistance Measurements

2W Ω Measurements

Most resistance measurements can be made using the simple 2-wire Ohms method. Simply connect the **V,W HI** to one end of the resistor, and the **V,W LO** to the other end. If the resistor to be measured is less than 30 k Ω , you should null out any lead resistance errors by first touching the **V,W HI** and **V,W LO** test leads together and then performing a **REL** function. If making measurements above 300 k Ω , you should use shielded or twisted leads to minimize noise pickup. This is especially true for measurements above 1 M Ω .

Knowing the SM-2020 Source current in Ohms measurements may be helpful to ensure repeatability of your measurements. (See Section 2.2 "Resistance, 2-wire and 4-wire" for a table of resistance range vs. current level.)

4W Ω Measurements

Using **4WW** is an extremely repeatable method of measuring resistors in the **300 W** and **30 kW** ranges. (It can also be used in the **300 kW** range, but is usually not necessary.) The SM-2020 **V,W HI** and **V,W LO** terminals source a current through the resistor under test, and the **I,4WW HI** and **I,4WW LO** sense the resultant voltage. The placement of each of the four test leads is important for making this measurement. The sense leads, **I,4WW HI & LO**, should always be placed closest to the body of the resistor. See Figure 6-2.

The reading rate in **4WW** is approximately 1/10th the **2WW** rate. This can be speeded up significantly by making a three terminal measurement. The DOS command **SetAuto4WRef** and Windows command **DMMSet4WRef** determine whether a four or three terminal measurement is made. The default is true 4-wire sensing of both the HI and LO terminals. When automatic lead compensation is turned off, which gives a three terminal measurement, only the HI side is sensed. Any lead resistance in the LO source lead can potentially cause an error. However, if the LO leads are stable in resistance, you can make one reference reading of the LO lead using **RefMeasure4WOhms** or **DMM4WOhmsRefMeasure** and still make accurate resistance measurements. See Sections 7 and 8 for more details.

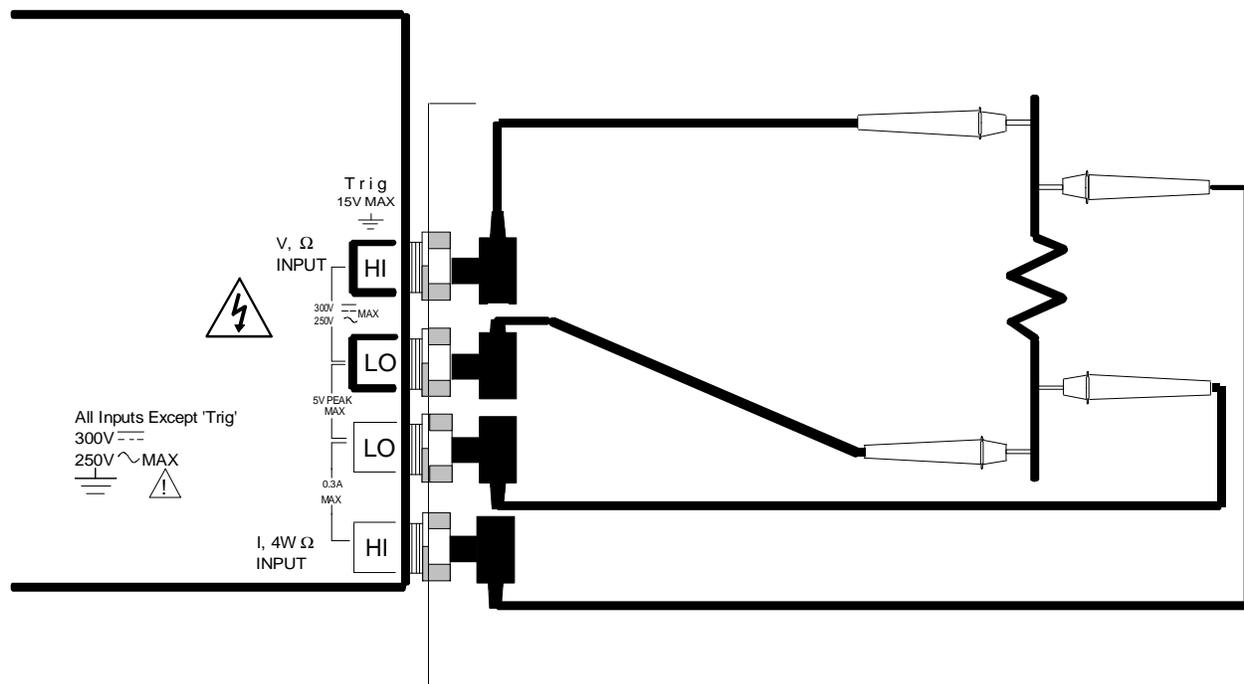


Figure 6-2. The **I,4WW HI & LO** leads should be closest to the body of the resistor when making 4W Ω measurements.

7.0 SM-2020 DOS Library and Interface

7.1 Distribution Files

Before installing, make sure you copy and save the distribution diskette. This chapter describes the DOS functions and tools found in the DOS_STUF directory of the distribution diskette. Other directories, e.g. BORLAND, has support for other compilers such as the Borland compiler version 4.0. Please see the README.TXT files in each of those directories for more information.

The SM2020D.LIB in DOS_STUF is a Microsoft® DOS™ SM-2020 DMM library written using the QuickC™ Large Model library. It is provided with a Control Panel application, SM2020D.EXE, which can be used to verify and demonstrate the DMM and the interface to the library. (The Control Panel is described in Section 5, "Measurement Functions".) The SM2020DV.LIB is a Microsoft® Visual C++™ Library.

The DOS part of the distribution diskette contains the following files:

<u>File</u>	<u>Description</u>
README.TXT	Release notes, latest information at the time of release. Important!
DMM.CFG	Configuration file which tells which DMMs are installed at which I/O addresses. This file is required at run time.
SM20CAL.DAT	Calibration file containing calibration information for the specific DMM. Do not write into this file, unless you are performing an external calibration!
SM2020D.LIB	The "Quick C" import library. Compiled with Large Model library. The driver library should be installed in your working directory. Quick C Ver. 2.5 or newer, with the assembler, is required.
SM2020DV.LIB	The "Visual C++" import library. Compiled with Large Memory Model library.
DMM2020.H & DMM_USER.H	Driver header file, constant definitions, and data structure.
DISP.H	Header file for TXT_GRAP.C with all the definitions.
PROT2020.H	Function prototypes for the library file SM2020D.LIB.
MOUSE.H	Function prototypes and definitions for the mouse support part.
TXT_GRAP.C	C source code for the display module of the DMM control panel.
SM2020D.MAK	The QuickC make file. It is setup for compiling SM2020D.EXE.
SM2020DV.MAK	The Visual C++ make file.
SM2020D.EXE	The executable control panel.
DOSSETUP.BAT	The setup file. Creates the proper directories and copies the proper files.
SM2020D.C	The main module of the control panel.

To install the SM-2020 software in DOS applications, put the distribution diskette in drive A. From the DOS command line, type **A:\DOSSETUP**.

7.2 Configuration Files

The SM-2020 DMM is configured using two configuration files. The file **DMM.CFG** contains the instrument types and base addresses of the DMMs installed in your PC. An example is:

```
2020 0x300
```

which says that an SM-2020 is installed at I/O address HEX 300. Up to four DMMs can be installed in a system. See "**Setting the SM-2020 Address**" discussed earlier for more information.

The file **SM20CAL.DAT** contains calibration records for each DMM. These records start with lines that resemble the lines below. No blank lines between records are allowed.

```
card_id    6      calibration_date    10/19/95
vdc
-2.326167e+002  1.000148
-1.551667e+001  1.000318
-1.041750e+002  1.000315
-2.675000e+000  1.000477
vac
1.437553e+009  1.030814  7
...
```

The first line identifies the DMM and the calibration date. The "card-id" is stored in ROM on each DMM. During initialization the driver uses the information from the **DMM.CFG** file to identify where the DMM is located in I/O space, reads the "card-id", and then reads the corresponding calibration information from the **SM20CAL.DAT** file. The calibration information consists of constants that correct for hardware inaccuracies of the DMM. Note: The **InitDmm** function reads the information from these files to initialize the DMM.

7.3 Using the SM-2020 Library

Install the header files in a directory that will be searched by your C compiler for header files. This header file is known to work with Microsoft Visual C++™ and QuickC™, but has not been tested with other compilers. If using QuickC, install **SM2020D.LIB** in a directory that will be searched by the linker for import libraries.

In using the SM-2020 driver, first call **InitDmm** to read the DMM configuration and calibration information. Call **SetDmmFunction** to set the DMM function. The DMM function constants are defined in the header file, and have names that clearly indicate the function they invoke. Notice that the function codes and ranges are inter-mixed. Don't hard-code the numeric values, since they will probably change. Use **SetAutoRange** to control autoranging. Use **SetDmmRate** to set the reading rate using the **DMMRate*** defined in the **DMM_USER.H** header file. Declare and allocate these structures in your DOS sources:

```
STATE far DMMstate[n];      /* current state of the DMM */
CAL far DMMcal[n];         /* cal data stores */
```

Note: Set *n* to the number of DMMs installed. 1 for a single DMM.

Four functions are provided to return DMM readings. **ReadDmmFloat** returns the next reading as a scaled single-precision (`float`) result, **ReadDmm** returns the next reading as a scaled double-precision (`double`) result, **ReadDmmStrg** returns the next reading as a formatted string ready to be displayed, and **ReadDmmNorm** returns a double-precision (`double`) normalized value (not scaled for range).

7.4 DOS Control Panel

The DOS control panel application, **SM2020D.EXE** is a simple interactive control panel for the SM-2020 DMM. When it loads, it will take a few seconds to initialize the hardware before the control panel is displayed. A more complete description of this panel is found in Section 5 of this manual.

The push-buttons labeled "DCV", "ACV", etc., control the DMM function. As you click on one with the left mouse button, the control panel application will switch the DMM to the selected range and function. The two arrow buttons in the RANGE box control the range. The reading rate is controlled by the two arrow buttons in the RATE box. Autorange mode is selected by clicking the "AUTO" in the range control area. The "S_CAL" push-button resets and recalibrates the DMM, leaving the DMM in the same state prior to "S_CAL". (This is an internal calibration only, and is different from the external calibration which uses the **SM20CAL.DAT** file. "S_CAL" is used to correct for any internal offset drifts due to changes in operating temperature.)

The individual context sensitive range push-buttons at the bottom of the panel set the selected range. The "To_FILE" button saves all readings to a text file named **DMM.DAT**. Three resolution push buttons are at the lower part of the panel for selecting the digits of resolution of the display. Increasing the reading rate above 60 rps and 100 rps will lower the resolution to 4 1/2 digits and 3 1/2 digits, respectively.

The SM-2020CT adds additional capabilities. The "frq" button appears in ACV and ACI and enables the frequency counter. When "frq" is enabled, the frequency and amplitude is shown at the same time. In this mode, the reading rate is slower than indicated. The "trg" button arms the hardware external trigger. When a hardware trigger occurs, 50 readings are taken and placed in a file called **TRIGGER.DAT**. When armed, the DOS panel will wait up to two minutes for a hardware trigger.

The source code file **TXT_GRP.C** contains the interface required for the DOS applications that interact with the driver, and the Control Panel.

The Make file **SM2020D.MAK** is setup to compile **SM2020D.EXE** using QuickC with the assembler option. It is used to compile **SM2020D.C**, and **TXT_GRP.C** as well as the **SM2020D.LIB**.

7.5 DOS Default Modes and Parameters

The DOS default modes and parameters on your SM-2020 and SM-2020CT are set up as follows:

- Function is set to VDC.
- DMM Resolution is set to 5-1/2 Digits.
- Measurement rate is set to RATE_10, or 10 readings per second.
- Autoranging is enabled, with the first range set to 300 V.
- Relative mode is disabled.
- Save Readings to a file (**DMM.DAT**) is disabled.
- 4-wire Ohms lead wire compensation is set to automatic. This gives potentially more accurate but slower readings. This is equivalent to the DOS SetAuto4WRef() command. See SetAuto4WRef() for complete details.

7.6 Mouse Operation

The DOS control panel includes serial mouse support. Slower DMM functions check for mouse events to speed up mouse action. For example, at measurement rates slower than 10 readings per second, or when selecting the Four Wire Ohms function, the **SM2020D.LIB** and the **SM2020DV.LIB** will check for mouse events by calling `MouseEvent()`, located in **TXT_GRAP.C**. This in turn calls `GetMouseEvent(&mEvent)` located in the library. If you prefer to use a mouse function other than `GetMouseEvent()` provided, replace `MouseEvent()` in **TXT_GRAP.C** with your own `MouseEvent()`. To prevent any mouse detection during DMM operation, such as required for embedded control software, you must replace `MouseEvent()` with a "do nothing" function in your source code:

```
void MouseEvent(void){ int i = 0 ; } /* do nothing */
```

If functions included in **TXT_GRAP.C** are not used, exclude it from the compile list in the make file. However, the `MouseEvent()` function must be provided in your source as indicated above.

7.7 DOS Command Language

The following sections contain detailed descriptions of each function of the DOS command language. Those commands that pertain to only the SM-2020CT are indicated. SM-2020 DOS software can support up to four DMMs. The card number is a value from 0 for the first DMM, to 3 for the fourth DMM. The card number is an integer *nDMM* and is the first parameter of each command.

ArmAnalogTrigger (SM-2020CT only)

Description Arm DMM for analog level trigger event.

```
#include "dmm_user.h"
```

```
int ArmAnalogTrigger(int nDMM, int nCount, double dLevel)
```

Remarks

This function sets the analog level trigger threshold of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) to *dLevel*. After the DMM is armed, it continuously takes measurements but does not store them into its internal buffer until the input crosses *dLevel*. Threshold crossing sense is determined by the first measurement following the call of **ArmAnalogTrigger**. If that measurement is lower than the set threshold *dLevel*, subsequent measurements greater than *dLevel* will trigger the DMM. If the first measurement is greater than *dLevel*, subsequent measurements smaller than *dLevel* will trigger. For example, if *dLevel* is 1.00000 V and the first reading after arming the DMM is 0.50000 V, then 1.00001 V (or greater) will trigger the DMM. If *dLevel* is 2.00000 V and the first reading after arming the DMM is 2.50000 V, then 1.99999 V (or smaller) will trigger the DMM. Upon triggering, the DMM saves the next *nCount* readings in the internal buffer.

Autorange is not allowed during an analog level trigger operation. *dLevel* must be within the current range of of the DMM, i.e., in the 3 V range *dLevel* must be within -3.1 and +3.1; in the 300 mV range it must be between -0.31 and +0.31. Units for *dLevel* are normalized

This function is very similar to **ArmTrigger()**. It uses **DmmBufferReady()** to check if a trigger occurred, as described in **ArmTrigger()**. It can also be monitored for completion using **Dmm_Ready()**. The analog level trigger operation can be terminated using the **DisArmTrigger()** function. When complete, the **ReadBurstBuffer()** and **ReadBuffer()** commands can be used to read the burst buffer, as described in the **ArmTrigger()** function.

nCount must be an integer between 1 and 64, inclusive. The current range, function, and rate is used. When using internal buffering such as in **ArmAnalogTrigger()** and **ArmTrigger()**, you can set the reading rate higher than 200 rps, up to 1000 rps. If the current reading rate is set slower than 10 rps, the DMM will make readings at 10 rps.

See the **ArmTrigger()** function for more details.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
NO_ERR	No error found.
TRIG_SAMPL_ERR	Wrong sample value.
ERR_NO_COUNTR	Counter/Trigger is not available.

Example

```
int status;
status = ArmAnalogTrigger(0, 64, 1.2345);
if (status == NO_ERR){
while( !DmmBufferReady(0) ); /* wait for trigger */
    for(i=0; i<=64;i++)
        printf("Samp. %d = %f\n",ReadBuffer(0));
}
```

ArmTrigger (SM-2020CT only)

Description Arm DMM for next hardware trigger event.

```
#include "dmm_user.h"
```

```
int ArmTrigger(int nDMM, int nCount)
```

Remarks

This function puts the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) in a wait mode. On an external hardware trigger the DMM makes *nCount* measurements and stores them in an internal buffer. *nCount* must be an integer between 1 and 64, inclusive. The current range, function, and rate is used. When using internal buffering such as in **ArmTrigger()**, you can set the reading rate higher than 200 rps, up to 1000 rps. If the current reading rate is set slower than 10 rps, the DMM will make readings at 10 rps.

The armed state can be terminated at any time by sending the termination command **DisArmTrigger()**.

Monitoring readiness can be done using the **DmmBufferReady()** or the **Dmm_Ready()** function. The use of **DmmBufferReady()** is limited in that once it returns TRUE, it should not be used again prior to reading the buffer. **DmmBufferReady()** must be used at least once, and return TRUE prior to reading the buffer, since it prepares the buffer for reading by the PC.

When ready, you can optionally read from 1 to *nCount* readings from the DMM using the **ReadBurstBuffer()** or the **ReadBuffer()** functions.

Autorangeing must not be used during any trigger operations. To prevent loss of data, no operations other than **DmmBufferReady()**, **Dmm_Ready()**, or **DisArmTrigger()** is allowed after issuing **ArmTrigger()**, **ArmAnalogTrigger()**, or **Trigger()**.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMM number (0 to 3).
<i>nCount</i>	int Sample count (1 to 64).

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
NO_ERR	No error found.
TRIG_SAMPL_ERR	Wrong sample value.
ERR_NO_COUNTR	Counter/Trigger is not available.

Example

```
int status;
status = ArmTrigger(0, 64);
while( !DmmBufferReady(0) && status == NO_ERR ); /* wait for
trigger */
for(i=0; i<=64;i++)
printf("Samp. %d = %f\n",ReadBuffer(0));
```

CalibrateDmm

Description Internally calibrate the DMM.

```
#include "dmm_user.h"

int CalibrateDmm(int nDMM)
```

Remarks Recalibrate the DMM specified by *nDMM* (DMMs are numbered from 0 to 3), leaving it in its current state after the calibration process.

Return Value int error code

<u>Value</u>	<u>Meaning</u>
NO_ERR	No error found.
ERR_HW_INIT	Could not reinitialize the hardware due to a failure.

Example

```
if(CalibrateDmm(0)==ERR_HW_INIT)
printf("H/W error");
```

Comments This performs an internal DMM calibration and is the same as the **S_CAL** command in the DOS Control Panel. It is not related to the external calibration represented in the **SM20CAL.DAT** file.

ClearRelative

Description Terminate (clear) the DMM relative measurement operation mode.

```
#include "dmm_user.h"

void ClearRelative(int nDMM)
```

Remarks Clear the relative mode of operation on the DMM specified by *nDMM* (DMMs are numbered from 0 to 3). The offset is set to zero. Function changes automatically clears relative mode.

Return Value None

Example

```
ClearRelative(0);
```

ClearToFile

Description Stop saving readings to the file

```
#include "dmm_user.h"
```

```
void ClearToFile(int nDMM)
```

Remarks Clears the flag in the internal data structure of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3), so the DMM stops saving data to the file. The **DMM.DAT** data file is closed.

Return Value None

Example `ClearToFile(0);`

DisArmTrigger (SM-2020CT only)

Description Terminate a trigger operation.

```
#include "dmm_user.h"
```

```
int DisArmTrigger(int nDMM)
```

Remarks This function sends the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) a trigger termination command. If the DMM is waiting for a trigger, it will exit the wait mode, and be ready for a new operation. It can be used following an external hardware or analog level trigger arm command (**ArmAnalogTrigger()**, **ArmTrigger()**, or **Trigger()**). It can be used with no limitation.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
TRUE	DMM is ready.
FALSE	DMM fault occurred in last trigger or disarm operation .
ERR_NO_COUNTER	Counter/Trigger is not available.

Example

```
ArmTrigger(0, 64);  
delay(5.0);  
if ( !Dmm_Ready(0) ) DisArmTrigger(0);
```

DmmBufferReady (SM-2020CT only)

Description Test whether the DMM is ready following a trigger event.

```
#include "dmm_user.h"
```

```
int DmmBufferReady(int nDMM)
```

Remarks This function detects whether or not the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) has received a trigger, and is done loading the buffer. If the DMM is ready, this function also prepares the DMM buffer for new readings. See **ArmTrigger()** for more details.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
TRUE	DMM is ready.
FALSE	DMM is not ready.
ERR_NO_COUNTR	Counter/Trigger is not available.
TRIG_ERR	The last trigger operation failed.

Example See **ArmTrigger()** description.

Dmm_Ready (SM-2020CT only)

Description Test whether the DMM is ready following a trigger event.

```
#include "dmm_user.h"
```

```
int Dmm_Ready(int nDMM)
```

Remarks This function detects whether or not the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) is ready following an external hardware or analog level trigger. It can be used with no limitation. However, if a buffer needs to be read, the **DmmBufferReady()** has to be used once after the DMM is ready, in order to prepare the DMM buffer for reading.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
TRUE	DMM is ready.
FALSE	DMM is not ready.

Example

```
ArmTrigger(0, 64);  
while( !Dmm_Ready(0) );  
DmmBufferReady(0);  
for(i=0; i<=64;i++)  
printf("Samp. %d = %f\n", i,ReadBuffer(0));
```

GetCalDate

Description Return the calibration date string.

```
#include "dmm_user.h"
```

```
char *GetCalDate(int nDMM)
```

Remarks This function reads the calibration date string of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) from the calibration record and returns a pointer to a string. It uses the Card ID code to select the appropriate record.

Return Value The return value is a pointer to the calibration date string.

Example

```
printf("%s",GetCalDate(0)); /* print a cal date */
```

GetDmmFuncRange

Description	Get the DMM composite function and a range. <pre>#include "dmm_user.h" int GetDmmFuncRange(int nDMM)</pre>
Remarks	Gets the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) composite function and range code. Composite codes are defined in dmm_user.h .
Return Value	int composite function/range code
Example	<pre>int func_range; func_range = GetDmmFuncRange(0); /* get the combo code */ if(func_range == VDC_300) printf("high DCV level");</pre>

GetDmmFunction

Description	Get the DMM function. <pre>#include "dmm_user.h" int GetDmmFunction(int nDMM)</pre>
Remarks	Gets the function of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) as defined in dmm_user.h . Possible return values are: VDC , VAC , IDC , IAC , OHMS2W and OHMS4W .
Return Value	Integer function code
Example	<pre>if(GetDmmFunction(0) == VDC) printf("VDC selected");</pre>

GetDmmRange

Description	Get DMM range. <pre>#include "dmm_user.h" int GetDmmRange(int nDMM)</pre>
Remarks	Gets the range value of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). There are at most six ranges, zero to five, depending on selected function. Zero is the first and lowest range.
Return Value	int range value
Example	<pre>if(GetDmmRange(0) == 0 printf("Lowest range.");</pre>

GetDmmRate

Description	Get DMM reading rate. <code>#include "dmm_user.h"</code> <code>double GetDmmRate(int <i>nDMM</i>)</code>
Remarks	Returns reading rate of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) in double float format. This function returns the actual DMM reading rate, compensating for the slower 4-Wire Ohms rate.
Return Value	double format rate value.
Example	<pre>double rate; rate = GetDmmRate(0); /* read actual measurement rate */</pre>

GetDmmResolution

Description	Get DMM digits of resolution. <code>#include "dmm_user.h"</code> <code>int GetDmmResolution(int <i>nDMM</i>)</code>								
Remarks	Gets the digits of reading resolution of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3).								
Return Value	int resolution value								
	<table><thead><tr><th><u>Value</u></th><th><u>Meaning</u></th></tr></thead><tbody><tr><td><code>THREE_A_HALF</code></td><td>Resolution is set to 3 1/2 digits.</td></tr><tr><td><code>FOUR_A_HALF</code></td><td>Resolution is set to 4 1/2 digits.</td></tr><tr><td><code>FIVE_A_HALF</code></td><td>Resolution is set to 5 1/2 digits.</td></tr></tbody></table>	<u>Value</u>	<u>Meaning</u>	<code>THREE_A_HALF</code>	Resolution is set to 3 1/2 digits.	<code>FOUR_A_HALF</code>	Resolution is set to 4 1/2 digits.	<code>FIVE_A_HALF</code>	Resolution is set to 5 1/2 digits.
<u>Value</u>	<u>Meaning</u>								
<code>THREE_A_HALF</code>	Resolution is set to 3 1/2 digits.								
<code>FOUR_A_HALF</code>	Resolution is set to 4 1/2 digits.								
<code>FIVE_A_HALF</code>	Resolution is set to 5 1/2 digits.								
Example	<pre>if(GetDmmResolution(0)==FIVE_A_HALF) printf("five and a half digits are set");</pre>								

GetFreq (SM-2020CT only)

Description Take a single frequency measurement.

```
#include "dmm_user.h"
```

```
double GetFreq(int nDMM)
```

Remarks Sets counter of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) to frequency counting mode, if not already set. Take a single frequency reading. Do a counter autorange to optimize the counter for the next frequency reading. If the Counter/Trigger is not installed, or the DMM is not in ACV or ACI, a zero frequency value is returned. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

Return Value Double Floating Frequency in Hertz.

Example

```
printf("My wave Frequency is %7.3 Hz\n",GetFreq(0) );
```

GetFreqStr (SM-2020CT only)

Description Take a single frequency measurement.

```
#include "dmm_user.h"
```

```
int GetFreqStr(int nDMM, char far *lpszReading)
```

Remarks Sets counter of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) to frequency counting mode, if not already set. Takes a single frequency reading. Do a single counter autorange to optimize the counter for the next frequency reading. If the Counter/Trigger is not installed, or the DMM is not in ACV or ACI, a zero frequency value is returned. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

lpszReading is a pointer to a buffer of at least 16 characters to hold the converted result. The return value is a formatted string containing the frequency in a five digit format with units.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
TRUE	Reading successfully made.
FALSE	Error occurred during reading.
ERR_NO_COUNTR	Counter/Trigger is not installed.

Example

```
char cBuff[17];  
in status;  
status = GetFreqStr(0, cBuff);  
if(status == TRUE) printf("Frequency is %s\n",cBuff );
```


GetGrdVer

Description	Get software version of firmware #include "dmm_user.h" int GetGrdVer(int nDMM)
Remarks	The SM-2020 has a microcontroller on board, which controls all internal DMM functions and handles communication to the PC based software. The code version of the microcontroller of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) is retrieved by this function.
Return Value	Integer version number.
Example	<code>guarded_ver = GetGrdVer(0);</code>
Comments	It is important to provide this value in any service calls to Signametrics.

GetID

Description	Get ID value of the DMM #include "dmm_user.h" int GetID(int nDMM)
Remarks	The returned ID value and the calibration file, SM20CAL.DAT , card_id entry must match for the DMM to operate. Each card has its own unique ID code value.
Return Value	Returns the card ID of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3).
Example	<code>id_val = GetID(0);</code>
Comments	It is important to provide this value in any service calls to Signametrics.

GetPeriod (SM-2020CT only)

Description	Make a single period measurement. #include "dmm_user.h" double GetPeriod(int nDMM)
Remarks	Sets counter of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) to period measuring mode, if not already set. Takes a single period reading. Do a counter autorange to optimize the counter range for the next period reading. If the Counter/Trigger is not installed, or the DMM is not in ACV or ACI, a zero period value is returned. If the DMM is in autorange, be certain to take an amplitude reading before using this command.
Return Value	Double Floating period (in seconds).
Example	<code>printf("My wave Period is %7.3 S\n",GetPeriod(0));</code>

GetPeriodStr (SM-2020CT only)

Description Take a single period measurement.

```
#include "dmm_user.h"
```

```
int GetPeriodStr(int nDMM, char far *lpszReading)
```

Remarks Sets counter of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) to period measuring mode, if not already set. Takes a single period reading. Do a counter autorange to optimize the counter range for the next period reading. If the Counter/Trigger is not installed, or the DMM is not in ACV or ACI, a zero period value is returned. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

lpszReading is a pointer to a buffer of at least 16 characters to hold the converted result. The return value is a formatted string containing the period in a five digit format with units.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
TRUE	Reading successfully made.
FALSE	Error occurred during reading, e.g. an under range occurred.
ERR_NO_COUNTR	Counter/Trigger is not installed.

Example

```
char cBuff[16];
in status;
status = GetPeriodStr(0, cBuff);
if(status == TRUE) printf("Period is %s\n",cBuff );
```

GetVer

Description Get main driver library software version number

```
#include "dmm_user.h"
```

```
double GetVer(void)
```

Remarks This function provides the **SM2020D.LIB** software version.

Return Value Double floating version number.

Example

```
software_ver = GetVer();
```

Comments It is important to provide this value in any service calls to Signametrics.

HasCountTrig (SM-2020CT only)

Description Test to detect if the Counter/Trigger is available.

```
#include "dmm_user.h"
```

```
int HasCountTrig(int nDMM)
```

Remarks This function detects whether or not the Counter/Trigger is installed in the DMM specified by *nDMM* (DMMs are numbered from 0 to 3).

Return Value Integer status code

<u>Value</u>	<u>Meaning</u>
TRUE	Counter/Trigger is available.
FALSE	Counter/Trigger is not available.

Example

```
int status;  
if(HasCountTrig(0)) status = GetPeriodStr(0, cBuff);
```


InitDmm

Description Initialize the DMM

```
#include "dmm_user.h"
```

```
int InitDmm(int nDMM)
```

Remarks Initialize the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) after reading configuration and calibration information from the files named **DMM.CFG** and **SM20CAL.DAT**. The hardware is set to an initial state which is VDC, 300 V range, 5 readings per second and autoranging mode.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
NO_ERR	DMM initialized successfully
NO_CAL_FILE	Calibration file could not be located, or invalid calibration file format.
ERR_OPEN_CFG	Configuration file could not be located.
ERR_HW_INIT	Invalid or card missing, wrong hardware address, I/O conflict or hardware failure.

Example

```
if( (i=InitDmm(0)) != NO_ERR){  
if(i==ERR_OPEN_CFG) printf("Can't open config file");  
else if(i==NO_CAL_FILE) printf("No SM20CAL.DAT found");  
else if(i==ERR_HW_INIT) Printf("Can't reset H/W.");  
else printf("Fail, can't identify cause.");  
exit(1);/* exit */
```

IsAuto4WRef

Description Return the state of the flag controlling the 4-wire Ohms compensation mode.

```
#include "sm2020.h"
```

```
int IsAuto4WRef(int nDMM)
```

Remarks **IsAuto4WRef()** tests the state of the automatic 4-wire Ohms lead compensation setting of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3). See **SetAuto4WRef**, **ClearAuto4WRef** for more details.

Return Value Integer status code

<u>Value</u>	<u>Meaning</u>
TRUE	4-wire Ohms mode includes automatic compensation.
FALSE	4-wire Ohms mode does not includes auto-compensation.

Example

```
if( !IsAuto4WRef(0) ) RefMeasure4WOhms(0); /* compensate for lead resistance if automatic compensation is not set */
```

IsAutoRange

Description Returns the state of autorange.

```
#include "dmm_user.h"
```

```
int IsAutoRange(int nDMM)
```

Remarks See **SetAutoRange()** and **ClearAutoRange()**.

Return Value Return TRUE if autorange is selected, FALSE if it is not. TRUE and FALSE are defined in **dmm_user.h**.

Example

```
if( IsAutoRange(0) ) printf("DMM is in Autorange");
```

IsRelative

Description	Returns the Relative state of the DMM. #include "dmm_user.h" int IsRelative(int <i>nDMM</i>)
Remarks	Returns the state of the Relative mode in the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3).
Return Value	Return TRUE if DMM is set to relative mode, FALSE if it is not. TRUE and FALSE are defined in dmm_user.h .
Example	<pre>if(IsRelative(0)) printf("DMM is in relative mode");</pre>

IsToFile

Description	Return the status of the "save to a file" flag of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) #include "dmm_user.h" int IsToFile(int <i>nDMM</i>)
Remarks	This function returns TRUE if the "save to a file" flag is set, FALSE otherwise.
Return Value	int flag status
Example	<pre>int save_to_file; save_to_file = IsToFile(0);</pre>

ReadBuffer (SM-2020CT only)

Description	Read a single measurement from the DMM internal buffer. <pre>#include "dmm_user.h"</pre> <pre>double ReadBuffer(int <i>nDMM</i>)</pre>
Remarks	Read a single reading from the internal buffer of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). From 1 to 64 readings can be stored and read. The return value is a double floating value corresponding to the present DMM function and range selected. For example, if the range is set to 300 mVDC, a return value for a 200 mV reading will be 200. For a 3 K Ω reading in the 3 K Ω range it will be 3.0, and for a 250 mA reading in the 300 mA range, 250 will be returned. The last function call prior to reading from the DMM buffer must be DmmBufferReady() . The internal buffer is a FIFO (First In, First Out), where the first sample read by the DMM into the buffer is retrieved first. Subsequent readings from the buffer will retrieve later DMM measurements. Overreading is allowed, but will result in bogus measurements.
Return Value	Double Floating value in selected base units. -100e6 is returned if the Counter/Trigger is not available.
Example	<pre>while(!DmmBufferReady(0)); printf("1st Triggered value = %f\n",ReadBuffer(0));</pre>

ReadBurstBuffer (SM-2020CT only)

Description	Read a single measurement from DMM internal buffer. <code>#include "dmm_user.h"</code> <code>double ReadBurstBuffer(int <i>nDMM</i>)</code>
Remarks	<p>Read a single reading from the internal buffer of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). From 1 to 64 readings can be stored and read. The return value is a double floating value corresponding to the base unit measurement. For example, if the range is set to 300 mVDC, a return value for a 200 mV reading will be 0.2. For a 3 kΩ reading in the 3 kΩ range, it will be 3.0e3. For a 250 mA reading in the 300 mA range, 0.25 will be returned.</p> <p>The last function call prior to reading from the DMM buffer must be DmmBufferReady().</p> <p>The internal buffer is a FIFO (First In, First Out), where the first sample read by the DMM into the buffer is read out first. Subsequent readings from the buffer will retrieve later DMM measurements. Overreading is allowed, but will result in bogus measurements from previous triggers.</p>
Return Value	Double Floating value in selected base units. -100e6 is returned if the Counter/Trigger is not available.
Example	<code>printf("1st Triggered value = %fV\n",ReadBurstBuffer(0));</code>

ReadDmm, ReadDmmFloat

Description Return the current floating-point reading from the DMM specified by *nDMM* (DMMs are numbered from 0 to 3).

```
#include "dmm_user.h"
```

```
float ReadDmmFloat(int nDMM)
```

```
double ReadDmm(int nDMM)
```

Remarks **ReadDmmFloat** reads the current result from the DMM, performs all scaling and conversion required, and returns the result as a 32-bit single-precision floating-point number. **ReadDmm** is the same as **ReadDmmFloat** but it returns a 64-bit double-precision floating-point number. The DMM continuously makes measurement at the set rate without any external software intervention. As a new measurement becomes available, it replaces the old measurement. Therefore, if the rate at which **ReadDmm** or **ReadDmmFloat** is executed does not keep up with the current DMM rate, old measurements are ignored and current measurements are returned. The returned value is scaled to the currently set range as in the following table:

<u>Input, Range and Function</u>	<u>Returned value</u>
300mV	300.0
3V	3.0
30V	30.0
300V	300.0
3mA	3.0
30mA	30.0
300mA	300.0
300Ohm	300.0
3K	3.0
30K	30.0
300K	300.0
3Meg	3.0
30Meg	30.0

Return Value The return value is a float for **ReadDMMFloat** and double for **ReadDMM**. Any value greater than 1e10, or smaller than -1e10 is considered an overrange.

Example 1

```
float f;  
f = ReadDmmFloat(0);
```

Example 2

```
double d;  
d = ReadDmm(0);
```

ReadDmmNorm

Description Return the current floating-point reading from the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) in base units.

```
#include "dmm_user.h"
```

```
double ReadDmmNorm(int nDMM)
```

Remarks **ReadDmmNorm** reads the current result from the DMM, performs all scaling and conversion required, and returns the result as a 64-bit double-precision floating-point number. It is different from **ReadDmm** and **ReadDmmFloat** in its returned value, which is not scaled to the set range. It is instead in the base value as in the following table:

<u>Input, Range and Function</u>	<u>Returned value</u>
300mV	0.300
3V	3.0
30V	30.0
300V	300.0
3mA	3.0e-3
30mA	30.0e-3
300mA	300.0e-3
300Ohm	300.0
3K	3.0e3
30K	30.0e3
300K	300.0e3
3Meg	3.0e6
30Meg	30.0e6

Return Value The return value is a double. Any value greater than 1e10, or smaller than -1e10 is considered an over range.

Example 1

```
double d;                                     d
= ReadDmmNorm(0);
```

ReadDmmStrg

Description Return the next reading from the DMM specified by *nDMM* (DMMs are numbered from 0 to 3), formatted for printing.

```
#include "dmm_user.h"
```

```
char *ReadDmmStrg(int nDMM)
```

Remarks This function reads the next result from the DMM, performs all scaling and conversion required, and returns the result as a string formatted for printing. The print format is determined by the set digits of resolution (see **SetDmmResolution()**), range and function. It is scaled as **ReadDmm**.

Return Value The return value is a pointer to the measurement string.

Example

```
printf("%s",ReadDmmStrg(0)); /* print a measurement */
```

RefMeasure4WOhms

Description	Make a single 4-wire Ohms compensation cycle of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). <pre>#include "sm2020.h"</pre> <pre>void RefMeasure4WOhms(int nDMM)</pre>
Remarks	RefMeasure4WOhms() makes a single 4-wire Ohms lead compensation measurement on the LO sense side. Subsequent 4-wire Ohms measurements are compensated for using the lead resistance measured with this call. In 4-wire Ohms, regardless of the compensation mode, the HI sense side always compensates for any lead resistance on the HI source side. See SetAuto4WRef, ClearAuto4WRef for more details.
Return Value	None.
Example	<pre>RefMeasure4WOhms(0); /* compensate for lead resistance */</pre>

SetAuto4WRef, ClearAuto4WRef

Description	Set 4-wire Ohms automatic compensation mode for the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). <pre>#include "sm2020.h"</pre> <pre>void SetAuto4WRef(void)</pre> <pre>void ClearAuto4WRef(int nDMM)</pre>
Remarks	SetAuto4WRef() sets a flag forcing automatic calibration of the LO lead wires. This is the normal, or default mode for 4-wire Ohms, giving lead resistance compensation for both the HI and LO leads. The 4-wire measurement, however, is slower (about 1/10th the 2-wire mode). The DMM, in 4-wire Ohms, always compensates for the lead resistance of the HI wires, regardless of the SetAuto4WRef() or Clear4WAutoRef() setting. ClearAuto4WRef() causes no automatic calibration of the LO lead wires, giving a 3-wire Ohms mode. This allows a much faster Ohms measurement, but it does not provide for lead wire compensation of the LO leads. Therefore, the user must explicitly invoke the lead wire compensation command RefMeasure4WOhms() periodically and at any time the LO lead wires are changed, for example, when using a scanner or when there are large variations in temperature. The last scenario can be significant due to the high temperature coefficient of copper wire, which is about 0.3% per °C.
Return Value	None.
Example	<pre>ClearAuto4WRef(0); /* disable auto-compensation */</pre>

SetAutoRange, ClearAutoRange

Description	Select autoranging operation, ClearAutoRange operation for the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3). <pre>#include "dmm_user.h"</pre> <pre>void SetAutoRange(int nDMM)</pre> <pre>void ClearAutoRange(int nDMM)</pre>
Remarks	Sets the DMM to autoranging operation mode. The DMM will automatically select the appropriate range for the input. It will remain in this mode until a range is changed or a ClearAutoRange() command is issued. It is compatible with function changes. When in autorange mode, the DMM will up range when the reading is over 103% of the current range (e.g. 309,999 counts). Down ranging will occur when the reading falls below 9% of the current range (e.g. 27,000 counts).
Return Value	None
Example	<pre>SetAutoRange(0); /* begin autoranging operation */</pre>

SetDmmFuncRange

Description	Set the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) to a function and range at the same time. <pre>#include "dmm_user.h"</pre> <pre>void SetDmmFuncRange(int nDMM, char funcrange)</pre>
Remarks	Sets the DMM function and range using a single command. When having to change both function and range, it is faster to use SetDmmFuncRange than to use the SetDmmFunction followed by SetDmmRange . Composite <i>funcrange</i> values are defined in dmm_user.h .
Return Value	None
Example	<pre>SetDmmFuncRange(0, VDC_3V);</pre>
Comments	The SetDmmFuncRange() performs limited internal calibration.

SetDmmFunction

Description	Set the DMM to a function. <pre>#include "dmm_user.h"</pre> <pre>int SetDmmFunction(int nDMM, char function)</pre>
Remarks	Sets the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) to a function defined in dmm_user.h . Setting a new function always selects the third range. For instance, selecting the VDC function sets the range to 30 V, and selecting Ohms sets it to 30 K range. The amount of time the DMM takes to select a new range is proportional to the currently set reading rate.
Return Value	Integer error code
Example	<pre>if(SetDmmFunction(0, VDC) == ERR_FUNC)printf("Wrong func.");</pre>
Comments	The SetDmmFunction() performs a limited internal calibration.

SetDmmRange

Description	Select DMM range. <pre>#include "dmm_user.h"</pre> <pre>int SetDmmRange(int nDMM, char range)</pre>
Remarks	Sets the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) to a range defined in dmm_user.h . There are at most six ranges (Ohms); the smallest number of ranges is three (current). The first range is the lowest (e.g. <code>_300mV,_3mA,_300</code>).
Return Value	Integer error code
Example	<pre>if(SetDmmRange(0, _30V) == ERR_RANGE) printf("Wrong range.");</pre>

SetDmmRate

Description	Set DMM reading rate. <pre>#include "dmm_user.h"</pre> <pre>int SetDmmRate(int nDMM, int rate)</pre>
Remarks	Sets reading rate of the DMM specified by <i>nDMM</i> (DMMs are numbered from 0 to 3) to a pre-set value. The legal rate values are defined in dmm_user.h . This function tests <i>rate</i> for validity and returns an error code. The default rate on initialization is five readings per second (DMMRate5). The 4-Wire Ohms function runs at 1/10th the rate set by <i>rate</i> since it takes ten readings for each measurement.
Return Value	int NO_ERR if no error detected or ERR_RATE if attempting to set illegal <i>rate</i> value.
Example	<pre>SetDmmRate(0, DMMRate10); /* Select 10 readings/Sec. */</pre>

SetDmmResolution

Description Set digits of resolution of the DMM specified by *nDMM* (DMMs are numbered from 0 to 3).

```
#include "dmm_user.h"
```

```
int SetDmmResolution(int nDMM, int resolution)
```

Remarks Sets the digits of reading resolution to one of three selections. The legal resolution values are defined in **dmm_user.h**. This function tests *resolution* for validity and returns an error code. The default resolution on initialization is five and a half digits. This function modifies the print formatting which only effects the string reading function **ReadDmmStrg()**.

<u>Value</u>	<u>Meaning</u>
THREE_A_HALF	Set resolution to 3 1/2 digits.
FOUR_A_HALF	Set resolution to 4 1/2 digits.
FIVE_A_HALF	Set resolution to 5 1/2 digits.

Return Value int error value

<u>Value</u>	<u>Meaning</u>
NO_ERR	No error
ILLEGAL_RESOLUT	Attempting to set illegal <i>resolution</i> value.

Example

```
SetDmmResolution(0, FIVE_A_HALF);  
/* Select 5-1/2 digits */
```

SetRelative

Description Set the DMM to relative measurement operation.

```
#include "dmm_user.h"
```

```
void SetRelative(int nDMM)
```

Remarks Sets the DMM specified by *nDMM* (DMMs are numbered from 0 to 3) to relative operation mode. The current reading is subtracted from all subsequent readings, until the function is changed or **ClearRelative()** function is issued. It is compatible with DMM range changes.

Return Value None

Example

```
SetRelative(0);
```

SetToFile

Description Save all readings in the file **DMM.DAT** for the DMM specified by *nDMM* (DMMs are numbered from 0 to 3).

```
#include "dmm_user.h"
```

```
void SetToFile(int nDMM)
```

Remarks This function sets a flag in the DMM internal data structure indicating that all readings are to be saved in a file **DMM.DAT**. It also opens this file in the current directory.

Return Value None

Example `SetToFile(0);`

An example of the **DMM.DAT** file record is

```
23:34:18    200.765
23:34:19    200.764

23:34:20    200.765
```

Trigger (SM-2020CT only)

Description Trigger SM-2020CT to take n samples from the DMM specified by $nDMM$ (DMMs are numbered from 0 to 3).

```
#include "sm2020.h"
```

```
int Trigger(int  $nDMM$ , int  $n$ )
```

Remarks Following reception of this command the SM-2020CT DMM makes n readings at the currently set function, range and rate, and stores them in an internal buffer. Rate can be set between 10 to 1000 readings per second. No autoranging is allowed for this trigger operation. Between the time the **Trigger** command is issued and the time the buffer is read, no other command should be sent to the DMM. Use **DmmBufferReady()** to monitor when the DMM is ready (ready implies completion of n readings). When ready, you can optionally read a single reading or up to n , using the **ReadBuffer()** or **ReadBurstBuffer()** function. Readiness can also be monitored using the **Dmm_Ready()** function. However, you need to use the **DmmBufferReady()** at least once prior to reading the buffer, since it prepares the buffer for reading when it detects a ready condition.

<u>Parameter</u>	<u>Type/Description</u>
n	int the number of samples the DMM takes following a trigger pulse. The number must be between 1 and 64, inclusive.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
NO_ERR	Operation successfully terminated.
TRIG_SAMPL_ERR	Measurement count is out of allowed range.
ERR_NO_COUNTR	Invalid DMM number.

Example

```
double Buffer[64]; Trigger(0, 64);
while( ! DmmBufferReady(0));
for(i=0; i < 64 ; i++)
    Buffer[i] = ReadBuffer(0);
```


8.0 SM-2020 DMM Windows Interface

8.1 Distribution Files

The main directory of the distribution diskette contains the Microsoft® Windows™ SM-2020 DMM software. To install this software, enter the command "A:SETUP" in the "Run Program" menu of the Windows File Manager; or double-click on the SETUP.EXE file name from the File Manager window. Most files on this diskette are compressed, and must be installed using the SETUP program.

The SM-2020 DMM DLL is a protected-mode Microsoft® Windows™ DLL that will control the Signametrics DMM. It is provided with a sample Visual Basic™ front-panel application to demonstrate the DMM and the interface to the DLL. Check the README.TXT file for more information about the files contained on the diskette. Some important files to note are:

<u>File</u>	<u>Description</u>
DMM.CFG	Configuration file which tells which DMMs are installed at which I/O addresses.
SM20CAL.DAT	Configuration file containing calibration information for each DMM. Do not write into this file unless you are performing an external calibration!
SM2020.LIB	The Windows import library. Install in a directory pointed to by your LIB environment variable.
SM2020.DEF	SM-2020 driver DLL module definition file.
SM2020.DLL	The driver DLL. This should be installed either in your working directory, in the Windows system directory, or in a directory on your PATH . The installation program installs this file in your Windows system directory (usually C:\WINDOWS\SYSTEM).
SM2020.H	Driver header file. Contains the definitions of the function prototypes for the DLL, constant definitions, and data structure definitions. Install in a directory pointed to by your INCLUDE environment variable.
VBRUN300.DLL	Visual Basic run-time interpreter. Install in your C:\WINDOWS\SYSTEM (or equivalent) directory.
THREED.VBX	Visual Basic extension DLL. Install in your C:\WINDOWS\SYSTEM (or equivalent) directory.
SPIN.VBX	Visual Basic extension DLL. Install in your C:\WINDOWS\SYSTEM (or equivalent) directory.
\SM2020FP	Subdirectory containing the Visual Basic front panel application, FP.EXE , its source code, and associated files. This program was generated with Visual Basic version 3.0.

To install the files from the distribution diskette, run the program **A:\SETUP.EXE** from the Program Manager's "File|Run..." dialog, or double-click the file name **A:\SETUP.EXE** from the File Manager. The files contained on the distribution diskette are compressed, and must be installed using **SETUP**.

8.2 Configuration Files

The SM-2020 DMM is configured using two configuration files. The file **DMM.CFG** contains the instrument types and base addresses of the DMMs contained in your PC. A total of four consecutive addresses are used by either the SM-2020 or the SM-2020CT. The SM-2020CT is identified as 2021 in the configuration file. 2020 identifies the SM-2020. The base address is the setting of the base I/O address switches on the card. The following **DMM.CFG** contents indicates a SM-2020CT at Hex address 300:

```
2021 0x300
```

Up to four DMMs can be installed in your PC. DMMs are numbered starting from 0 for the first, to 3 for the fourth. **DMM.CFG** must have a line indicating the instrument type and base address of each DMM. No empty lines are allowed. Use any ASCII text editor to modify **DMM.CFG**, to reflect the DMM type and the base address. The following **DMM.CFG** contents indicates DMM number 0 is a SM-2020 at Hex address 0x204, and DMM number 1 is a SM-2020CT at 0x300:

```
2020 0x204
2021 0x300
```

The file **SM20CAL.DAT** contains calibration information for each DMM. This file may contain multiple records, for up to four DMMs. Each record starts with a header line, followed by calibration data. No empty lines are allowed between each record.

```
card_id    6      calibration_date    10/19/94
vdc
-2.326167e+002  1.000148
-1.551667e+001  1.000318
-1.041750e+002  1.000315
-2.675000e+000  1.000477
vac
1.437553e+009  1.030814  7
...
card_id    588    calibration_date    12/12/95
vdc
-2.314578e+002  1.000137
```

The first line identifies the DMM and the calibration date. The "card-id" is stored in ROM on each DMM. During initialization the driver uses the information from the **DMM.CFG** file to identify where the DMM is located in I/O space, reads the "card-id" and "calibration_date", and then reads the corresponding calibration information from the **SM20CAL.DAT** file.

The **DMMInit** function reads the information from these files to initialize the DMM. **DMMInit** accepts parameters that are the names of these files.

8.3 Using the SM-2020 Driver

Install the **SM2020.H** header file in a directory that will be searched by your C/C++ compiler for header files. This header file is known to work with Microsoft Visual C++™, but has not been tested with other compilers. Install **SM2020.LIB** in a directory that will be searched by the linker for import libraries. Install the **SM2020.DLL** in a location where either your program will do a **LoadLibrary** call to load it, or on the **PATH** so that Windows will load the DLL automatically.

In using the SM-2020 driver, first call **DMMInit** to read the DMM configuration and calibration information. Call **DMMSetFunction** to set the DMM function: the DMM function constants are defined in the header file, and have names that clearly indicate the function they invoke. Notice that the function codes and ranges are inter-mixed. Don't hard-code the numeric values, since they will probably change. Use **DMMSetAutoRange** to control autoranging. Use **DMMSetRate** to set the reading rate using the **DMM_R_*** defines in the header file.

Three functions are provided to return DMM readings. **DMMRead** returns the next reading as a scaled single-precision (`float`) result, **DMMReadDbl** returns the next reading as a scaled double-precision (`double`) result, and **DMMReadingStr** returns the next reading as a formatted string ready to be displayed.

All functions accept a DMM-number parameter, which must be set to zero for the DLL. Each function returns an error code.

The **DMMGetInfo** function returns information about a particular DMM, including its type code (such as "2020"), serial number, reading rate, and function. This information is returned in a **DMMINFO** structure supplied by the caller.

8.4 Visual Basic Front Panel Application

The Visual Basic front panel application, **FP.EXE** (in the **SM2020FP** subdirectory) is a simple interactive control panel for the SM-2020 DMM. When it loads it will take a few seconds to initialize the hardware before the front panel is displayed.

The push-buttons labeled "DCV", "ACV", etc., control the DMM function. As you push one, the front panel application will switch the DMM to the selected range and function. The two spin buttons control the range (upper spin button) and reading rate (lower spin button). Autorange mode is selected by changing the range past the highest programmable range, at which time autorange will be selected. The "S_CAL" push-button resets and recalibrates the DMM, leaving the DMM in the same state prior to "S_CAL". (This is an internal calibration only, and is different from the external calibration which uses the **SM20CAL.DAT** file. "S_CAL" is used to correct for any internal offset drifts due to changes in operating temperature.) The Trigger push-button takes a single reading and displays the result.

The SM-2020CT adds additional capabilities. The "frq" and "per" buttons appear in ACV and ACI and enable the frequency counter. When "frq" is enabled, the frequency and amplitude is shown at the same time. In this mode, the reading rate is slower than indicated. When "per" is enabled, the period is shown. The "X-trg" button arms the hardware external trigger. When a hardware trigger occurs, 50 readings are taken and placed in a file called **TRIGGER.DAT**. When armed, the DOS panel will wait up to two minutes for a hardware trigger.

The indicators in the lower right-hand corner indicate when a trigger is being executed, and when over range and other faults occur.

The source code file **GLOBAL.BAS** (in the **V_BASIC** directory of the distribution diskette) contains the interface and constant definitions required to write Visual Basic applications which interact with the driver DLL, along with some global variables required for this particular front-panel application. See the **README.TXT** file in **V_BASIC** for more information.

8.5 Windows DLL Default Modes and Parameters

The Windows DLL default modes and parameters on your SM-2020 and SM-2020CT are set up as follows:

- Function is set to VDC.
- DMM Resolution is set to 5-1/2 Digits.
- Measurement rate is set to RATE_10, or 10 readings per second.
- Autoranging is enabled, with the first range set to 300 V.
- Relative mode is disabled.
- Save Readings to a file (**DMM.DAT**) is disabled.
- Four-Wire Ohms lead wire compensation is set to automatic. This gives potentially more accurate but slower readings. This is equivalent to the Windows **DMMSet4WRef()** command.

8.6 Using the SM2020 DLL with LabWindows/CVI® Version 3.1

When using the SM2020DLL with LabWindows/CVI, you should make these changes to the **SM2020.H** file:

1. Include a LabWindows/CVI type definitions header file (for instance '**wincvi.h**'), which is needed to call your other DLLs including the SM2020.DLL. See below.

```
#include "wincvi.h"
```

2. Delete '**__export**' from all functions declarations.
3. Replace type '**int**' with type '**short**'
4. Replace type '**unsigned**' with type '**unsigned short**' for instance:
change: `int __export WINAPI DMMSetRate(int nDmm, int nRate);`
to: `short WINAPI DMMSetRate(short nDmm, short nRate);`

The header file (for example: **wincvi.h**) is needed due to compensate for the National Instruments DLL interface limitation. The **wincvi.h** file is shown below:

```
/* filename: WINCVI.H
```

```
    This file contains the required definitions of Window's data types used to call driver DLL's such as  
    Signametrics' SM2020.DLL. The definitions are consistent with Watcom's <windows.h>, as required by  
    National Instruments LabWindows/CVI (R).
```

```
*/
```

```
#ifndef WINCVI__H
```

```
    #define WINCVI__H 1  
    #define FALSE 0  
    #define TRUE 1  
    #define OFF 0  
    #define ON 1  
    /*** common definitions and typedefs ***/  
    #define VOID void  
    #define FAR __far  
    #define near __near  
    #define pascal __pascal  
    #define CDECL __cdecl  
    #define WINAPI __far __pascal  
    #define CALLBACK __far __pascal  
    /*** simple types & common helper macros ***/  
    typedef short BOOL;  
    typedef unsigned char BYTE;  
    typedef unsigned short WORD;  
    typedef unsigned long DWORD;  
    typedef unsigned int UINT;  
    #ifdef STRICT  
        typedef signed long LONG;  
    #else  
        #define LONG long  
    #endif  
    /*** common pointer types ***/  
    #ifndef NULL  
        #define NULL 0  
    #endif  
    typedef DWORD NEAR* PDWORD;  
    typedef DWORD FAR* LPDWORD;  
    typedef void FAR* LPVOID;  
    typedef short SHORT;  
    typedef short __far * LPSHORT;  
    typedef char NEAR* PSTR;  
    typedef char NEAR* NPSTR;  
    typedef char FAR* LPSTR;  
    typedef const char FAR* LPCSTR;  
    typedef BYTE NEAR* PBYTE;  
    typedef BYTE FAR* LPBYTE;  
    typedef int NEAR* PINT;  
    typedef int FAR* LPINT;  
    typedef WORD NEAR* PWORD;  
    typedef WORD FAR* LPWORD;  
    typedef long NEAR* PLONG;  
    typedef long FAR* LPLONG;
```

```
#endif
```

An example application of SM2020 DLL calls from LabWindows/CVI® is shown below. It contains functions **measure_ohms()** and **measure_vdc()**, with sample calls to the SM-2020. NOTE: Although these measurement functions use LabWindows/CVI® and the LabWindows/CVI(R) Test Executive, they are not necessarily coded to LabWindows® instrument driver standards.

```
/* function: measure_ohms, purpose: measure 2-wire ohms */
int measure_ohms(double OHMreading) {
    short ret, i;
    DMMSetFunctions (0, OHMS2W);
    DMMSetAutoRange (0, TRUE);
    /* to settle auto-range and function changes ignore three readings */
    for( i = 0 ; i < 4 ; i++ ) ret = DMMReadNorm (0, & OHMreading);
    return ret;
}
/* function: measure_vdc, purpose: measure DC Volts */
int measure_vdc(double Vreading) {
    short ret, i;
    DMMSetFunctions (0, VDC);
    DMMSetAutoRange (0, TRUE);
    /* to settle auto-range and function changes ignore three readings */
    for( i = 0 ; i < 4 ; i++ ) ret = DMMReadNorm (0, &Vreading);
    return ret;
}
```

The **LABWINDO** directory of the distribution diskette may contain more information about using the SM2020 DLL with LabWindows/CVI® Version 3.11. Please see the files in that directory.

8.7 Windows Command Language

The following sections contain detailed descriptions of each function of the Windows command language. Those commands that pertain to only the SM-2020CT are indicated.

DMM4WOhmsRefMeasure

Description Make a single 4-wire Ohms reference reading.

```
#include "sm2020.h"
```

```
int DMM4WOhmsRefMeasure(int nDmm)
```

Remarks Perform a single lead compensation of the LO lead wires. This command should be used to compensate for lead resistance of the LO lead wires when the 4-wire Ohms automatic compensation mode is off, as described in the **DMMSet4WRef()** function. See **DMMSet4WRef()** for complete details.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Unconfigured DMM
DMM_E_INIT	Uninitialized DMM

DMMArmAnalogTrigger (SM-2020CT only)

Description Arm SM-2020CT for analog level trigger operation.

```
#include "sm2020.h"
```

```
int DMMArmAnalogTrigger(int nDmm, int nSamples, double FAR *dThresh)
```

Remarks

Setup the SM-2020CT for analog level trigger operation. Following reception of this command the DMM makes measurements continuously, waiting for a value which exceeds the threshold, *dThresh*. When this occurs, a trigger is produced with identical processing as in **DMMArmTrigger**. Threshold crossing sense is determined by the first measurement following the call of **DMMArmAnalogTrigger**. If that measurement is lower than the set threshold, *dThresh*, subsequent measurements greater than *dThresh* will trigger the DMM. If the first measurement is greater than *dThresh*, subsequent measurements smaller than *dThresh* will trigger. For example, if *dThresh* is 1.00000 V and the first reading after arming the DMM is 0.50000 V, then 1.00001 V (or greater) will trigger the DMM. If *dThresh* is 2.00000 V and the first reading after arming the DMM is 2.50000 V, then 1.99999 V (or smaller) will trigger the DMM.

The *dThresh* value is in base units, and must be within the DMM range setting. For example, in the 300 mV range, *dThresh* must be within -0.310000 and +0.310000. In the 30 K Ω , range *dThresh* must be between 0.0 and 31.0e3.

Following an analog level trigger event, the DMM makes *nSamples* readings at the set function, range, and reading rate, and stores them in an internal buffer. Autoranging is not allowed when using **DMMAnalogTrigger**. Between the time the **DMMArmAnalogTrigger** is issued and the time the buffer is read, no other command should be sent to the DMM. One exception is the **DMMDisArmTrigger** command.

Use the **DMMTrigrBufReady** to monitor when the DMM is ready. When ready, you can read a single reading or up to *nSamples*, using **DMMReadBuffer** or **DMMReadBufferStr** functions. Readiness can also be monitored using the **DMMReady** function. However, you need to use the **DMMTrigrBufReady** prior to reading the buffer, since it prepares the buffer for reading when it detects a ready condition.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nSamples</i>	int The number of samples the DMM takes following a trigger pulse. This number must be between 1 and 64, inclusive.
<i>dThresh</i>	double FAR Analog level trigger threshold value

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to use of any function.
DMM_TRIG_N	Measurement count is out of allowed range.
DMM_E_DMM	Invalid DMM number.

Example

```
double Buffer[64];
DMMArmAnalogTrigger(0,64,1.5);
while( ! DMMTrigrBufReady(0));
    for(i=0; i < 64 ; i++)
        Buffer[i] = DMMReadBuffer(0);
```

DMMArmTrigger (SM-2020CT only)

Description Arm SM-2020CT for external trigger operation.

```
#include "sm2020.h"
```

```
int DMMArmTrigger(int nDmm, int nSamples)
```

Remarks Setup the SM-2020CT for external hardware trigger operation. Following reception of this command the DMM enters a wait state. After reception of an external trigger pulse, the DMM makes *nSamples* readings at the set function, range, and reading rate; and stores them in an internal buffer. No autoranging is allowed for external trigger operation. Between the time the **DMMArmTrigger** is issued and the time the buffer is read, no other command should be sent to the DMM. One exception is the **DMMDisarmTrigger** command.

Use the **DMMTrigrBufReady** to monitor when the DMM is ready (following trigger and the reading of *nSamples*). When ready, you can optionally read a single reading or up to *nSamples*, using the **DMMReadBuffer** or **DMMReadBufferStr** function. Readiness can also be monitored using the **DMMReady** function. However, you need to use the **DMMTrigrBufReady** prior to reading the buffer, since it prepares the buffer for reading when it detects a ready condition.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nSamples</i>	int The number of samples the DMM takes following a trigger pulse. This number must be between 1 and 64, inclusive.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to use of any function.
DMM_TRIG_N	Measurement count is out of allowed range.
DMM_E_DMM	Invalid DMM number.

Example

```
double Buffer[64];
DMMArmTrigger(0,64);
while( ! DMMTrigrBufReady(0));
    for(i=0; i < 64 ; i++)
        Buffer[i] = DMMReadBuffer(0);
```

DMMCalibrate

Description Calibrate the DMM

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMCalibrate(int nDmm)
```

Remarks This function recalibrates the DMM, and returns it to the current operating mode.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
NO_ERR	DMM is OK.
ERR_HW_INIT	Hardware error due to bad address, conflict, or other hardware problems.
DMM_E_DMM	Invalid DMM number.

Example `status = DMMCalibrate(0); /* a quick internal cal.*/`

Comments This performs an internal DMM calibration and is the same as the **S_CAL** command in the DOS Control Panel. It is not related to the external calibration represented in the **SM20CAL.DAT** file.

DMMDelay

Description Wait for a given time.

```
#include "sm2020.h"
```

```
int DMMDelay(double dTime)
```

Remarks Delay of *dTime* seconds. *dTime* must be a positive double number between 0.0 and 100 seconds.

<u>Parameter</u>	<u>Type/Description</u>
<i>dTime</i>	double Delay time in seconds.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated
DMM_E_DMM	Invalid DMM number.

Example `DMMDelay(1.2); /* wait for 1.2 Sec */`

DMMDisArmTrigger (SM-2020CT only)

Description Abort trigger operation.

```
#include "sm2020.h"
```

```
int DMMDisArmTrigger(int nDmm)
```

Remarks This function sends the DMM a trigger termination command. If the DMM is waiting for a trigger, it will exit the wait mode, and be ready for a new operation. It can be used following an external hardware or analog level trigger arm command (**DMMArmAnalogTrigger**, **DMMArmTrigger**, or **DMMTrigger**). It can be used with no limitation.

Return Value Integer error code

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to using any function
DMM_TRIG_ER	Trigger disarm error.
DMM_E_DMM	Invalid DMM number.

DMMFrequencyStr (SM-2020CT only)

Description Return the next DMM frequency reading, formatted for printing.

```
#include "sm2020.h"
```

```
int DMMFrequencyStr(int nDmm, LPSTR lpszReading)
```

Remarks This function makes frequency measurement and returns the result as a string formatted for printing. The print format is fixed to six digits plus units, e.g., 05.001 Hz. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpszReading</i>	LPSTR Points to a buffer (at least 16 characters long) to hold the converted result.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to using any function.
DMM_E_DMM	Invalid DMM number.
DMM_CNT_RNG	Frequency counter is over or under range.

Example

```
char cBuf[17];  
int status;  
status = DMMFrequencyStr(0, cBuf);
```

DMMGetCalDate

Description Return the calibration date string from the DMM.

```
#include "sm2020.h"
```

```
int DMMGetCalDate(int nDmm, LPSTR lpszCalDate)
```

Remarks This function reads the calibration date string from the structure.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpszCalDate</i>	LPSTR Points to a buffer (at least 16 characters long) to hold the cal date string.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
any positive number	Length of the date string
DMM_E_DMM	Invalid DMM number.

Example

```
char cBuf[16];  
int status;  
status = DMMGetCalDate(0, cBuf);
```

DMMGetFnRange

Description Get DMM range code

```
#include <windows.h>
#include "sm2020.h"
#include "dmm_user.h"
```

```
int DMMGetFnRange(int nDmm)
```

Remarks This function returns the combined DMM function/range code. See **dmm_user.h** for the complete set of codes.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer value corresponding to the currently set DMM function/range, or an error code. The following are a few examples of the returned value.

<u>Value</u>	<u>Meaning</u>
VDC_300mV	First and lowest VDC range (300 mV)
VDC_300V	Highest VDC range.
VAC_300mV	First and lowest VAC range (300 mV)
VAC_300V	Highest VAC range.
IAC_3mA	First and lowest IAC range (3 mA)
IAC_300mA	Highest IAC range.
IDC_3mA	First and lowest IDC range is 3 mA.
IDC_300mA	Highest IDC range.
OHM_2W_300	First 2-wire Ohms range is 300 Ohms.
OHM_2W_300K	Fourth 2-wire Ohms range is 300 k.
OHM_2W_30MEG	The highest 2-wire Ohms range is 30 MOhms.
OHM_4W_3K	Second 4-wire Ohms range is 3 k.
OHM_4W_300K	Fourth 4-wire Ohms range is 300 k
DMM_E_DMM	Invalid DMM number error.

Example

```
int id;if(DMMGetFnRange == VDC_300mV)
printf("Lowest VDC range selected");
```


DMMGetFunction

Description Get DMM function code

```
#include <windows.h>
#include "sm2020.h"
#include "dmm_user.h"
```

```
int DMMGetFunction(int nDmm)
```

Remarks This function returns the DMM function code.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer value corresponding to the current function, or an error code.

<u>Value</u>	<u>Meaning</u>
VDC	Volts DC
VAC	Volts AC
IDC	DC current
IAC	AC current
OHMS2W	2-wire Ohms
OHMS4W	4-wire Ohms
DMM_E_DMM	Invalid DMM number.

Example

```
int id; if(DMMGetFunction == VDC) printf("VDC Function
selected");
```

DMMGetGrdVer

Description Get DMM firmware version

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMGetGrdVer(int nDmm)
```

Remarks This function returns the DMM firmware version.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer value. The return value is the version value or an error code.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Invalid DMM number.

Example `firmwarever = DMMGetGrdVer(0);`

DMMGetID

Description Get DMM ID code

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMGetID(int nDmm)
```

Remarks This function returns the DMM identification code. Each DMM has a unique ID code that must match the calibration file **card_ID** field.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer value card ID code or an error code.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Invalid DMM number.

Example `int id;
id = DMMGetID(0);`

DMMGetInfo

Description Get current setting information for a DMM

```
#include "sm2020.h"
```

```
int DMMGetInfo(int nDmm, LPDMMINFO lpDmm)
```

Remarks Return information about DMM number *nDmm* in the **DMMINFO** structure pointed to by *lpDmm*. If *lpDmm* is **NULL** simply return **DMM_OKAY** if the DMM is configured, or **DMM_E_DMM** if not.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpDmm</i>	LPDMMINFO Points to a DMMINFO structure to contain the current DMM status. If <i>nDmm</i> refers to a valid DMM number and <i>lpDmm</i> is not NULL , information about the DMM will be returned. If <i>lpDMM</i> is NULL , then no information about the DMM will be returned, but the function return value will be DMM_OKAY if <i>nDmm</i> refers to a valid DMM.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM information returned successfully.
DMM_E_DMM	Invalid DMM number.

Example

```
DMMINFO dmm;  
status = DMMGetInfo(0, &dmm);  
status = DMMGetInfo(0, NULL);  
/* return status only */
```


DMMGetRange

Description Get DMM range code

```
#include <windows.h>
#include "sm2020.h"
#include "dmm_user.h"
```

```
int DMMGetRange(int nDmm)
```

Remarks This function returns the DMM range code.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer value corresponding to the currently set DMM range, or an error code.

<u>Value</u>	<u>Meaning</u>
0	First and lowest range (300mV, 300Ohms, 3mA)
1	Second range (3V, 3kOhm, 30mA)
2	Third range (30V, 30kOhm, 300mA)
3	Fourth range (300kOhm)
4	Fifth range (3MOhm)
5	Sixth range (30MOhm)
DMM_E_DMM	Invalid DMM number.

Example

```
int id;
if(DMMGetRange == 0) printf("Lowest range selected");
```


DMMGetRate

Description Get DMM rate

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMGetRate(int nDmm, double FAR *lpdRate)
```

Remarks This function returns a double floating rate in readings per second. The returned value is corrected for the 4-wire Ohms reduced rate.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpdRate</i>	double FAR pointer where the rate is saved.

Return Value Integer value version code or an error code.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Invalid DMM number.

Example

```
int status;
double FAR rate;
status = DMMGetRate(0, &rate);
```

DMMGetResolut

Description Get DMM resolution

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMGetResolut(int nDmm)
```

Remarks This function returns the DMM digits of resolution.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
THREE_A_HALF	3-1/2 digits
FOUR_A_HALF	4-1/2 digits
FIVE_A_HALF	5-1/2 digits
DMM_E_DMM	Invalid DMM number.

Example `resolution = DMMGetResolut(0);`

DMMGetVer

Description Get DMM software version

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMGetVer(double FAR *lpfResult )
```

Remarks This function returns the DMM software version, which is a double floating value.

<u>Parameter</u>	<u>Type/Description</u>
<i>lpfResult</i>	double FAR pointer to the location which holds the version.

Return Value Integer value version code or an error code.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Invalid DMM number.

Example

```
int status;
double d;
status = DMMGetVer(&d);
```

DMMHasCountOption

Description Test for the presence of the Trigger/Counter.

```
#include "sm2020.h"
```

```
BOOL DMMHasCountOption(int nDmm)
```

Remarks Test if the installed DMM is a SM-2020CT.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
TRUE	Counter is present (It's an SM-2020CT)
FALSE	Counter is not present.

DMMInit

Description Initialize a DMM

```
#include "sm2020.h"
```

```
int DMMInit(int nDmm, LPCSTR lpszCfg, LPCSTR lpszCal)
```

Remarks Initialize the DMM numbered *nDmm* (where the first DMM is number 0), after reading configuration and calibration information from the files named *lpszCfg* and *lpszCal*.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpszCfg</i>	LPCSTR Points to the name of the file containing the configuration information for the DMM. Normally, configuration information is read from the file named DMM.CFG located in the current directory used.
<i>lpszCal</i>	LPCSTR Points to the name of the file containing the calibration constants for the DMM. Calibration information is normally read from the file named SM20CAL.DAT located in the current directory.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_DMM	Invalid DMM number.
DMM_E_CFG	Configuration file could not be located, or invalid configuration file format.
DMM_E_CAL	Calibration file could not be located, or invalid calibration file format.

Example

```
/* initialize DMM. Assume I and dResults were declared as
int and double */
I = DMMInit(0, "dmm.cfg", "sm20cal.dat");

I = DMMSetFunction(0,VDC); /* Set function to VDC */
I = DMMSetRange(0, _30V); /* Set Range to 30V DC */

I = DMMSetRate(0,DMM_R_30); /* 30 samples per sec. If this
line is not added, the default reading rate would be 10/s */

I = DMMReadDbl(0,&dResults); /* take a reading */
```


DMMInitialize

Description Initialize a DMM.

```
#include "sm2020.h"
```

```
int DMMInitialize(int nDmm, int iBaseAddress, LPCSTR lpszCal)
```

Remarks **DMMInitialize** is similar to **DMMInit** with one exception. The second parameter is an integer base address rather than a path to the DMM configuration file. Therefore the configuration file **DMM.CFG** is not read.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>iBaseAddress</i>	int Card base address.
<i>lpszCal</i>	LPCSTR Pointer to a string containing the path to the calibration file.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated
DMM_E_DMM	Invalid DMM number.

DMMIsAuto4WRef

Description Test for the status of the 4-wire Ohms lead compensation mode.

```
#include "sm2020.h"
```

```
BOOL DMMIsAuto4WRef(int nDmm)
```

Remarks See also **DMMSet4WRef()**.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
TRUE	4-Wire Ohms automatic compensation is turned on.
FALSE	4-Wire Ohms auto compensation is turned off.

Example

```
if(DMMIsAuto4WRef()) DMM4WOhmsRefMeasure(0);
```


DMMIsAutoRange

Description Get the status of the autorange flag.

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMIsAutoRange(int nDmm)
```

Remarks This function returns the DMM autorange flag state.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value TRUE, FALSE or an error code.

<u>Value</u>	<u>Meaning</u>
TRUE	Autoranging mode is selected.
FALSE	Autoranging mode is not selected.
DMM_E_DMM	Invalid DMM number.

Example

```
int autorange; autorange = DMMIsAutoRange(0);
```

DMMIsRelative

Description Get DMM ID code

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMIsRelative(int nDmm)
```

Remarks This function returns the DMM Relative flag state.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value Integer TRUE, FALSE or an error code.

<u>Value</u>	<u>Meaning</u>
TRUE	Relative mode is selected.
FALSE	Relative mode is not selected.
DMM_E_DMM	Invalid DMM number.

Example

```
int relstate;
relstate = DMMIsRelative(0);
```

DMMPeriodStr (SM-2020CT only)

Description Return the next DMM period reading, formatted for printing.

```
#include "sm2020.h"
```

```
int DMMPeriodStr(int nDmm, LPSTR lpzReading)
```

Remarks This function makes a period measurement and returns the result as a string formatted for printing. The print format is fixed to five digits plus units, e.g., 150.01 ms. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpzReading</i>	LPSTR Points to a buffer (at least 16 characters long) to hold the converted result. The return value will consist of a leading sign, a floating-point value in exponential notation, and a units specifier.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated
DMM_E_INIT	DMM is uninitialized. Must be initialize prior to use of any function.
DMM_E_DMM	Invalid DMM number.
DMM_CNT_RNG	Period measurement H/W is over or under range.

Example

```
char cBuf[17];  
int status;  
status = DMMPeriodStr(0, cBuf);
```

DMMRead, DMMReadDbI

Description Return the next floating-point reading from the DMM.

```
#include "sm2020.h"
```

```
int DMMRead(int nDmm, float FAR *lpfResult)  
int DMMReadDbI(int nDmm, double FAR *lpdResult)
```

Remarks **DMMRead** reads the next result from the DMM, performs all scaling and conversion required, and returns the result as a 32-bit single-precision floating-point number in the location pointed to by *lpfResult*. **DMMReadDbI** reads the next result from the DMM, performs all scaling and conversion required, and returns the result as a 64-bit double-precision floating-point number in the location pointed to by *lpdResult*.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpfResult</i>	float FAR * Points to the location to hold the next reading.
<i>lpdResult</i>	double FAR * Points to the location to hold the next reading.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_DMM	Invalid DMM number.
DMM_E_RANGE	DMM over range error occurred.

Example 1

```
float f;  
int status;  
status = DMMRead(0, &f);
```

Example 2

```
double d;  
int status;  
status = DMMReadDbI(0, &d);
```

DMMReadBuffer (SM-2020CT only)

Description Return the next double floating-point reading from the DMM internal buffer.

```
#include "sm2020.h"
```

```
int DMMReadBuffer(int nDmm, double FAR *lpdResult)
```

Remarks Read the next measurement from the DMM internal buffer, pointed to by the internal buffer pointer, and increment that pointer. Store the measurement as a 64-bit double-precision floating-point number in the location pointed to by *lpdResult*.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpdResult</i>	double FAR * Points to the location which holds the frequency.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to using any function.
DMM_E_DMM	Invalid DMM number.

Example

```
double Buffer[10];
int status;
DMMArmTrigger(0,10);
while( ! DMMTrigrBufReady(0));
for(i=0; i < 10 ; i++)
    status = DMMReadBuffer(0, &Buffer[i]);
```

DMMReadBufferStr (SM-2020CT only)

Description Return the next reading, formatted for printing.

```
#include "sm2020.h"
```

```
int DMMReadBufferStr(int nDmm, , LPSTR lpszReading)
```

Remarks The same as **DMMReadBuffer** except the reading is formatted as a string with units. Measurements are stored as a null terminated string at the location pointed to by *lpszReading*.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpszReading</i>	LPSTR Points to the location which holds the formatted reading string.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. Must be initialize prior to using any function.
DMM_E_DMM	Invalid DMM number.

Example

```
char Buf[17];  
DMMArmTrigger(0,1);  
while( !DMMTrigrBufReady(0));  
DMMReadBufferStr(0, Buf);
```

DMMReadFrequency (SM-2020CT only)

Description Return the next double floating-point frequency reading from the DMM.

```
#include "sm2020.h"
```

```
int DMMReadFrequency(int nDmm, double FAR *lpdResult)
```

Remarks If frequency counter is not engaged, select it. Make a single frequency measurement, and store the result as a 64-bit double-precision floating-point number in the location pointed to by *lpdResult*. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpdResult</i>	double FAR * Points to the location to hold the frequency.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. Must be initialize prior to using any function.
DMM_E_DMM	Invalid DMM number.
DMM_CNT_RNG	Frequency counter is over or under range.

Example 1

```
double d;  
int status;  
status = DMMReadFrequency(0, &d);
```

DMMReadingStr

Description Return the next reading from the DMM formatted for printing.

```
#include "sm2020.h"
```

```
int DMMReadingStr(int nDmm, LPSTR lpszReading)
```

Remarks This function reads the next result from the DMM, performs all scaling and conversion required, and returns the result as a string formatted for printing. The print format is determined by the range and function.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpszReading</i>	LPSTR Points to a buffer (at least 16 characters long) to hold the converted result. The return value will consist of a leading sign, a floating-point value in exponential notation, and a units specifier.

Return Value The return value is one of the following constants, or the string length is OK.

<u>Value</u>	<u>Meaning</u>
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_DMM	Invalid DMM number.
DMM_E_RANGE	DMM over range error occurred.

Example

```
char cBuf[17];  
int status;  
status = DMMReadingStr(0, cBuf);
```

DMMReadNorm

Description Take a reading that is in base value

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMReadNorm(int nDmm, double FAR *lpdRead)
```

Remarks This function returns a double floating readings. The returned value is corrected for base units. That is, it returns 0.3 for 300 mV input and 1e6 for 1.0 MOhm.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpdRead</i>	double FAR Pointer to a location where the reading is saved.

Return Value Integer value version code or an error code.

<u>Value</u>	<u>Meaning</u>
DMM_E_OP	Error during DMM operation.
DMM_E_RANGE	Over/Under range error.
DMM_E_DMM	Invalid DMM number.
DMM_OKAY	Valid return.

Example

```
int status;
double reading;
status = DMMReadNorm(0, &reading);
```

DMMReadPeriod (SM-2020CT only)

Description Return the next double floating-point period reading from the DMM.

```
#include "sm2020.h"
```

```
int DMMReadPeriod(int nDmm, double FAR *lpdResult)
```

Remarks If frequency counter is not engaged, select it. Make a single period measurement, and store the result as a 64-bit double-precision floating-point number in the location pointed to by *lpdResult*. If the DMM is in autorange, be certain to take an amplitude reading before using this command.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>lpdResult</i>	double FAR * Points to the location which holds the period.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully completed.
DMM_E_INIT	DMM is uninitialized. DMM must be initialized prior to using any function.
DMM_E_DMM	Invalid DMM number.
DMM_CNT_RNG	Period measurement hardware is over or under range.

Example 1

```
double d;  
int status;  
status = DMMReadPeriod(0, &d);
```

DMMReady (SM-2020CT only)

Description Return the readiness state of the DMM following a trigger operation.

```
#include "sm2020.h"
```

```
int DMMReady(int nDmm)
```

Remarks Whenever the DMM is done performing a task, such as an analog level trigger, a frequency measurement, or an external hardware trigger, it sends data to a hardware buffer in the SM-2020CT. The **DMMReady** function checks that buffer and returns TRUE if data is present, and FALSE otherwise. Detecting readiness using this function is not sufficient prior to reading the SM-2020 internal buffer. The **DMMTrigrBufReady** function must be used and return TRUE before reading.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
TRUE	DMM is ready.
FALSE	DMM is not ready.
DMM_E_INIT	DMM is uninitialized. Must be initialize prior to using any function.
DMM_E_DMM	Invalid DMM number.

DMMSet4WRef

Description Set 4-Wire Ohms automatic compensation mode.

```
#include "sm2020.h"
```

```
int DMMSet4WRef(int nDmm, BOOL bRef)
```

Remarks

If *bRef* is TRUE, set a flag forcing automatic calibration of the LO lead wires. This is the normal, or default mode for 4-wire Ohms, giving lead resistance compensation for both the HI and LO leads. The 4-wire measurement, however, is slower (about 1/10th the 2-wire mode). The DMM, in 4-wire Ohms, always compensates for the lead resistance of the HI wires, regardless of the **SetAuto4WRef()** or **Clear4WAutoRef()** setting.

If *bRef* is FALSE, the flag is cleared, indicating no automatic calibration of the LO lead wires, giving a 3-wire Ohms mode. This allows a much faster Ohms measurement, but it does not provide for lead wire compensation of the LO leads. Therefore, the user must explicitly invoke the lead wire compensation command **DMM4WOhmsRefMeasure()** periodically and at any time the LO lead wires are changed, for example when using a scanner or when there are large variations in temperature. The last scenario can be significant due to the high temperature coefficient of copper wire, which is about 0.3% per °C.

<u>Parameter</u>	<u>Type/Description</u>
<i>bRef</i>	BOOL Sets a flag to TRUE or FALSE enabling or disabling the 4-wire Ohms automatic compensation function.
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_E_DMM	Invalid DMM number.
DMM_E_INIT	Uninitialized DMM.

Example `DMMSet4WRef(0, FALSE); /* disable auto-compensation */`

DMMSetAutoRange

Description Set autorange status for DMM

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMSetAutoRange(int nDmm, BOOL bAuto)
```

Remarks This function enables or disables autorange operation of the DMM.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>bAuto</i>	BOOL Determines whether or not autoranging is done. The value TRUE enables autoranging, FALSE disables it.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Function succeeded.
DMM_E_DMM	Invalid DMM number.

Example

```
status = DMMSetAutoRange(0, TRUE); /* enable */
status = DMMSetAutoRange(0, FALSE); /* disable */
```

DMMSetFnRange

Description Set the DMM function and range.

```
#include "sm2020.h"  
#include "dmm_user.h"
```

```
int DMMSetFnRange(int nDmm, int nFunc)
```

Remarks This function sets both, the function and range used by the DMM. The table of values is defined as *VDC_300mV*, *VAC_3V*, *IDC_300mA*, *OHM_4W_300K* etc. definitions in the header files.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nFunc</i>	int A pre-defined constant corresponding to the desired function and range.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_DMM	Invalid DMM number.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_FUNC	Invalid DMM function.

Example `status = DMMSetFnRange(0, IDC_3mA);`

DMMSetFunction

Description Set the DMM function.

```
#include "sm2020.h"  
#include "dmm_user.h"
```

```
int DMMSetFunction(int nDmm, int nFunc)
```

Remarks This function sets the function used by the DMM. The table of values is defined by the *VDC*, *VAC*, *IDC*, *IAC*, *OHMS2W*, and *OHMS4W* definitions in the DLL header file.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nFunc</i>	int A pre-defined constant corresponding to the desired function.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_DMM	Invalid DMM number.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_FUNC	Invalid DMM function.

Example `status = DMMSetFunction(0, IDC);`

DMMSetRange

Description Set the DMM range for the present function.

```
#include "sm2020.h"
```

```
int DMMSetRange(int nDmm, int nRange)
```

Remarks This function sets the range used by the DMM for the present function. The table of values is defined by the *_300mV*, *_3mA*, *etc.* definitions in the DLL header file.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nRange</i>	int A pre-defined constant corresponding to the desired range.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_DMM	Invalid DMM number.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_FUNC	Invalid DMM range value.

Example `status = DMMSetRange(0, _300mA);`

DMMSetRate

Description Set the DMM reading rate.

```
#include "sm2020.h"
```

```
int DMMSetRate(int nDmm, int nRate)
```

Remarks This function sets the reading rate used by the DMM (expressed in readings per second). The table of values is defined by the DMM_R_* definitions in the DLL header file. (In 4-wire Ohms, the reading rate is 1/10th the set rate.)

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nRate</i>	int A pre-defined constant (DMM_R_*) corresponding to the desired reading rate.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM initialized successfully.
DMM_E_DMM	Invalid DMM number.
DMM_E_OP	DMM could not be initialized due to configuration error, or due to missing or inoperative hardware.
DMM_E_RATE	Invalid DMM reading rate.

Example `status = DMMSetRate(0, DMM_R_100);`

DMMSetRelative

Description Set the DMM relative reading mode for the present function.

```
#include "sm2020.h"
```

```
int DMMSetRelative(int nDmm, BOOL bRelative)
```

Remarks This function selects relative or absolute reading mode for the DMM. If the *bRelative* parameter value is TRUE the DMM will change to relative reading mode. FALSE, the DMM will change to absolute reading mode. Caution: Do not select **DMMSetRelative** when in the autorange mode.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>bRelative</i>	BOOL TRUE to enter relative mode, FALSE to enter absolute mode.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	DMM mode changed successfully.
DMM_E_DMM	Invalid DMM number.

Example `status = DMMSetRelative(0, TRUE);`

DMMSetResolut

Description Set DMM resolution

```
#include <windows.h>
#include "sm2020.h"
```

```
int DMMSetResolut(int nDmm, int nResolut)
```

Remarks This function sets the DMM digits of resolution.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nResolut</i>	THREE_A_HALF Sets to 3 1/2 digits.
	FOUR_A_HALF Sets to 4-1/2 digits.
	FIVE_A_HALF Sets to 5-1/2 digits.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
ILEGAL_RESOLUT	Wrong resolution.
DMM_E_DMM	Invalid DMM number.

Example `status = DMMSetResolut(0, THREE_A_HALF); /* 3-1/2 */`

DMMTerminate

Description Terminate DMM operation (DLL)

```
#include "sm2020.h"
```

```
int DMMTerminate(int nDmm)
```

Remarks Terminate DMM number *nDmm*. This is an exit routine that is used only where it is needed to terminate one DMM and start a new one at the same *nDmm* location. Otherwise it is not necessary to use this function.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM to be suspended.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
TRUE	DMM Terminated
FALSE	DMM was not initialized, termination is redundant.

Example `DMMTerminate(0); /* Terminate DMM # 0 */`

DMMTrigger (SM-2020CT only)

Description Trigger SM-2020CT to take *nSamples*.

```
#include "sm2020.h"
```

```
int DMMTrigger(int nDmm, int nSamples)
```

Remarks Following reception of this command the SM-2020CT DMM makes *nSamples* readings at the currently set function, range and rate, and stores them in an internal buffer. Rate can be set between 10 to 1000 readings per second. No autoranging is allowed for this trigger operation. Between the time the **DMMTrigger** command is issued and the time the buffer is read, no other command should be sent to the DMM. Use the **DMMTrigrBufReady** to monitor when the DMM is ready (ready implies completion of *nSamples*). When ready, you can optionally read a single reading or up to *nSamples*, using **DMMReadBuffer**. Readiness can also be monitored using the **DMMReady** function. However, it is necessary to use the **DMMTrigrBufReady** once (and only once) prior to reading the buffer, since it prepares the buffer for reading when it detects a ready condition.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.
<i>nSamples</i>	int The number of samples the DMM takes following a trigger pulse. This number must be between 1 and 64, inclusive.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
DMM_OKAY	Operation successfully terminated.
DMM_E_INIT	DMM is uninitialized. Must be initialize prior to using any function.
DMM_TRIG_N	Measurement count is out of allowed range.
DMM_E_DMM	Invalid DMM number.

Example

```
double Buffer[64];
int state;
DMMTrigger(0,64);
while( ! DMMTrigrBufReady(0));
    for(i=0; i < 64 ; i++)
        state = DMMReadBuffer(0, &Buffer[i]);
```

DMMTrigrBufReady (SM-2020CT only)

Description Return the ready state of the DMM following trigger operation.

```
#include "sm2020.h"
```

```
int DMMTrigrBufReady(int nDmm)
```

Remarks Following the completion of an external hardware trigger, analog level trigger, or software trigger operation, the DMM sends the status to indicate acquisition completion. The **DMMTrigrBufReady** function checks the buffer and returns TRUE if data is present, and FALSE otherwise. Once a TRUE status is returned, the **DMMTrigrBufReady** function should not be used again prior to reading the buffer. Detecting readiness using **DMMReady** is not restricted in the number of times it can be used, but it does not prepare the buffer for reading as **DMMTrigrBufReady** does.

<u>Parameter</u>	<u>Type/Description</u>
<i>nDmm</i>	int Identifies the DMM. DMMs are numbered starting with zero.

<u>Value</u>	<u>Meaning</u>
TRUE	DMM is done and buffer is ready to be read.
FALSE	DMM is not ready.
DMM_E_INIT	DMM is uninitialized. Must be initialized prior to using any function.
DMM_E_DMM	Invalid DMM number.

Return Value The return value is one of the following constants.

Example

```
double Buffer[10]; DMMTrigger(0,10);
while( DMMTrigrBufReady(0) == FALSE );
for(i=0; i < 10 ; i++)
    Buffer[i] = DMMReadBuffer(0);
```

9.0 Maintenance

Warning

These service instructions are for use by qualified personnel only. To avoid electric shock, do not perform any procedures in this section unless you are qualified to do so.

This section presents maintenance information for the SM-2020. It includes a performance test and describes the calibration procedure.

Test equipment recommended for the performance test and calibration is listed below. If the recommended equipment is not available, equipment that meets the indicated minimum specifications may be substituted. In general, the calibration equipment should be at least four times more accurate than the SM-2020 specifications.

Table 9-1. Recommended Test Equipment

Instrument Type	Minimum Specifications	Recommended Model
Multi-Function Calibrator	DC Voltage Range: 0-300 V Voltage Accuracy: 25 ppm AC Voltage Range: 0-250 V Voltage Accuracy: 0.025% Resistance Range: 0-19 M Ω Resistance Accuracy: 70 ppm DC Current Range: 0-300 mA Current Accuracy: 0.018% AC Current Range: 50 μ A - 300 mA Current Accuracy: 0.18%	Fluke 5700A

9.1 Performance Tests

This test compares the performance of the SM-2020 with the specifications given in Section 2. The test is recommended as an acceptance test when the instrument is first received, and as a verification after performing the calibration procedure. To ensure proper performance, the test must be performed with the SM-2020 installed in a personal computer, with the covers on. The ambient temperature must be between 18°C to 28°C. Allow the SM-2020 to warm up one-half hour before performing any of the tests. The default reading rate of the SM-2020 should be used in each test.

9.2 DC Voltage Test

The following procedure may be used to verify the accuracy of the DCV function:

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Apply a high quality copper wire short to the SM-2020 **V,W HI & LO** inputs. Select the DCV function, Autorange. Allow the SM-2020 to settle for several seconds, and perform the **REL** function.
3. Apply the following DC voltages to the **V, W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-2. DC Voltage Test

Step	Range	Input	Minimum Reading	Maximum Reading
1	300 mV	0V (short)	-000.006 mV	+000.006 mV
2	300 mV	190 mV	+189.948 mV	+190.052 mV
3	300 mV	-190 mV	-189.948 mV	-190.052 mV
4	3 V	1.9 V	+1.89977 V	+1.90027 V
5	3 V	-1.9 V	-1.89977 V	-1.90027 V
6	30 V	19 V	+18.9953 V	+19.0053 V
7	30 V	-19 V	-18.9953 V	-19.0053 V
8	300 V	190 V	+189.951 V	+190.051 V
9	300 V	-190 V	-189.951 V	-190.051 V

9.3 Resistance Test, 2-wire

The following procedure may be used to verify the accuracy of the 2-wire function.

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Connect the SM-2020 **V,W HI & LO** terminals to the calibrator HI & LO Outputs. Output 0Ω from the calibrator. Allow the SM-2020 to settle for a few seconds, and perform the **REL** function. (This effectively nulls out the lead resistance of your cabling. If you are using a Fluke 5700A Calibrator, the 2-Wire Compensation feature will give a more accurate 2-wire ohms measurement. See the *Fluke 5700A Operator's Manual* for further instructions.)
3. Apply the following Resistance values to the **V, W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-3 Resistance Test, 2-wire

Step	Range	Input	Minimum Reading	Maximum Reading
1	300 Ω	100 Ω	99.964 Ω	100.036 Ω
2	3 kΩ	1 kΩ	.99967 kΩ	1.00033 kΩ
3	30 kΩ	10 kΩ	9.9966 kΩ	10.0034 kΩ
4	300 kΩ	100 kΩ	99.966 kΩ	100.034 kΩ
5	3 MΩ	1 MΩ	.99800 MΩ	1.00210 MΩ
6	30 MΩ	10 MΩ	9.9340 MΩ	10.0660 MΩ

9.4 Resistance Test, 4-wire

The following procedure may be used to verify the accuracy of the 4-wire function.

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Connect the SM-2020 **V,W HI & LO** terminals to the calibrator HI & LO Output. Connect the SM-2020 **I, 4W W HI & LO** terminals to the HI & LO Sense terminals.
3. Select the 4WΩ function on the SM-2020, Autorange. Set the calibrator to 0Ω. Be certain that the calibrator is set to external sense ("EX SNS" on the Fluke 5700A). Allow the SM-2020 to settle for a few seconds, and perform the **REL** function.
4. Apply the following Resistance values to the **V, W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-4 Resistance Test, 4-wire

Step	Range	Input	Minimum Reading	Maximum Reading
1	300 Ω	0 Ω	00.000 Ω	00.006 Ω
2	300 Ω	100 Ω	99.964 Ω	100.036 Ω
3	3 k Ω	0 Ω	.00000 Ω	.00004 Ω
4	3 k Ω	1 k Ω	.99967 k Ω	1.00033 k Ω
5	30 k Ω	10 k Ω	9.9966 k Ω	10.0034 k Ω
6	300 k Ω	100 k Ω	99.966 k Ω	100.034 k Ω

Note: The use of 4-wire Ohms for resistance values above 300 k Ω is not recommended.

9.5 AC Voltage Test

The following procedure may be used to verify the accuracy of the ACV function:

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Apply the following AC voltages to the **V, W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-5. Mid-Frequency AC Voltage Tests
All inputs are at **400 Hz**.

Step	Range	Input	Minimum Reading	Maximum Reading
1	300 mV	10 mV	009.670 mV	10.330 mV
2	300 mV	190 mV	189.130 mV	190.870 mV
4	3 V	100 mV	.09715 V	.10285 V
5	3 V	1.9 V	1.89535 V	1.90465 V
6	30 V	1 V	0.9715 V	1.0285 V
7	30 V	19 V	18.9535 V	19.0465 V
8	250 V	10 V	9.715 V	10.285 V
9	250 V	190 V	189.535 V	190.465 V

Table 9-6. High-Frequency AC Voltage Tests
All inputs are at **50 kHz**.

Step	Range	Input	Minimum Reading	Maximum Reading
1	300 mV	10 mV	009.400 mV	10.600 mV
2	300 mV	190 mV	185.800 mV	194.200 mV
4	3 V	100 mV	.09600 V	.10400 V
5	3 V	1.9 V	1.87800 V	1.92200 V
6	30 V	1 V	0.9600 V	1.0400 V
7	30 V	19 V	18.7800 V	19.2200 V
8	250 V	10 V	9.600 V	10.400 V
9	250 V	100 V	98.700 V	101.300 V

9.6 DC Current Test

The following procedure may be used to verify the accuracy of the DCI function:

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Remove all connections from the SM-2020 inputs. Select the DCI function, Autorange. Allow the SM-2020 to settle for a few seconds, and perform the **REL** function.

3. Apply the following DC currents to the **I,4W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-7. DC Current Test

Step	Range	Input	Minimum Reading	Maximum Reading
1	3 mA	0 mA (open)	-.00008 mA	+.00008 mA
2	3 mA	1 mA	.99922 mA	1.00078 mA
3	30 mA	0 mA (open)	-.0015 mA	+.0015 mA
4	30 mA	10 mA	9.9895 mA	10.0105 mA
5	300 mA	0 mA (open)	-.015 mA	+.015 mA
6	300 mA	100 mA	99.895 mA	100.105 mA

9.7 AC Current Test

The following procedure may be used to verify the accuracy of the ACI function:

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Remove all connections from the SM-2020 inputs. Select the ACI function, Autorange.
3. Apply the following AC currents to the **I,4W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-8. AC Current Test
All Inputs are at **1 kHz**

Step	Range	Input	Minimum Reading	Maximum Reading
1	3 mA	0.1 mA	.09767 mA	.10233 mA
2	3 mA	1 mA	.99000 mA	1.01000 mA
3	30 mA	1 mA	.9767 mA	1.0233 mA
4	30 mA	10 mA	9.9000 mA	10.1000 mA
5	300 mA	10 mA	9.767 mA	10.233 mA
6	300 mA	100 mA	99.000 mA	101.000 mA

9.8 Frequency Counter Test (SM-2020CT only)

The following procedure may be used to verify the accuracy of the Frequency Counter:

1. If you have not done so, install the SM-2020 and place the covers back on to the computer. Ensure that the computer has been on for at least one-half hour, with the covers on, before conducting this test.
2. Select the ACV function, autorange. Turn **frq** on.
3. Apply the following AC voltages to the **V, W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-9. ACV Frequency Counter Test

Step	Range	Input	Minimum Counter Reading	Maximum Counter Reading
1	300 mV	33 mV, 40 Hz	39.988 Hz	40.012 Hz
2	3 V	330 mV, 40 Hz	39.992 Hz	40.008 Hz
3	30 V	3.3 V, 40 Hz	39.992 Hz	40.008 Hz
4	300 V	33 V, 40 Hz	39.992 Hz	40.008 Hz
5	300 mV	33 mV, 100 kHz	99.988 kHz	100.012 Hz
6	30 V	3.3 V, 100 kHz	99.988 kHz	100.012 Hz

2. Select the ACI function, autorange. Turn **frq** on.
3. Apply the following AC currents to the **I,4W HI & LO** terminals. Check to see that the displayed reading on the SM-2020 is within the tolerances listed.

Table 9-10. ACI Frequency Counter Test

Step	Range	Input	Minimum Counter Reading	Maximum Counter Reading
1	3 mA	330 uA, 40 Hz	39.992 Hz	40.008 Hz
2	30 mA	15 mA, 40 Hz	39.992 Hz	40.008 Hz
3	300 mA	150 mA, 40 Hz	39.992 Hz	40.008 Hz

9.9 Calibration

Each SM-2020 DMM uses its own **SM20CAL.DAT** calibration file to ensure the accuracy of its functions and ranges. The **SM20CAL.DAT** file is a text file that contains the DMM identification number, calibration date, and calibration constants for all DMM ranges. For most functions, the calibration constants are scale factor and offset terms that solve the " $y = mx + b$ " equation for each range. An input " x " is corrected using a scale factor term " m " and an offset term " b "; this gives the desired DMM reading, " y ". An example **SM20CAL.DAT** is shown:

```
card_id    6      calibration_date    10/19/94
vdc
-2.326167e+002  1.000148
-1.551667e+001  1.000318
-1.041750e+002  1.000315
-2.675000e+000  1.000477
vac
1.437553e+009  1.030814  7
3.698987e+006  1.024822  7
2.613935e+006  1.036304  29
2.685829e+006  1.030241  30
idc
-2.621e+002    1.021803
-2.546e+002    4.995248
-1.788e+001    4.996209
iac
2.126894e+006  1.055925  31
2.123906e+006  5.156784  31
1.993815e+006  5.127059  31
2w-ohm
1.995817e+003  1.000900
2.010500e+002  1.001050
-9.166667e-001  1.003340
-2.145000e+001  1.003511
-2.315000e+001  1.000927
-2.078333e+001  0.820876
4w-ohm
1.891875e+002  1.000897
2.068750e+001  1.001051
4.312500e+000  1.003347
-7.187500e+000  1.003508
-8.875000e+000  1.000925
0.000000e+000  0.821324
```

The first column under any function, e.g., "**vdc**", is the offset term " b ", expressed as a value proportional to analog-to-digital (a/d) counts. The second column is the scale factor term " m ". Within each function, the " b " and " m " terms are listed with the lowest range at the beginning. For example, under "**4w-ohm**" above, "**1.891875e+002**" represents the offset term for the 300 Ω range, and "**1.000897**" is the scale factor term for the 300 Ω range.

For the AC functions, there is a third column that represents a digital code from 0 to 31 that controls the high frequency performance of each AC function. A large value, e.g., 31, causes the frequency response to go up.

The **SM20CAL.DAT** file is created by performing external calibration. The general calibration algorithm consists of applying a zero value to the DMM followed by a value near the top of each range. Calibration of your SM-2020 is best performed using DOS® calibration software available from Signametrics.

When using multiple DMMs in a single chassis, the **SM20CAL.DAT** file must have a calibration record for each DMM. You can combine the unique calibration records of each DMM into one **SM20CAL.DAT** file using any ASCII text editor. Be certain there are no empty lines between calibration records. The first record is for DMM number 0 (the first DMM), the second record is for DMM number 1 (the second DMM), etc.

10.0 Warranty and Service

The SM-2020 is warranted for a period of one year from date of purchase.

If your unit requires repair or calibration, contact your Signametrics representative. There are no user servicable parts within the SM-2020. Removal of any of the three external shields will invalidate your warranty. For in-warranty repairs, you must obtain a return authorization from Signametrics prior to returning your unit.