Programming Manual

RFP3000 Series

Real-Time Peak Power RF Sensors

Contents

1 SCPI Functions 9
1.1 Command Summary 9
1.2 PwrSnsr_ReadSCPI() 9
1.3 PwrSnsr_ReadSCPIBytes() 10
1.4 PwrSnsr_ReadSCPIFromNamedParser() 11
1.5 PwrSnsr_SendSCPIBytes() 12
1.6 PwrSnsr_SendSCPICommand() 12
1.7 PwrSnsr_SendSCPIToNamedParser() 13

2 Session Management 14
2.1 Commands Summary 14
2.2 PwrSnsr_ClearError() 14
2.3 PwrSnsr_close() 15
2.4 PwrSnsr_FindResources() 15
2.5 PwrSnsr_GetFetchLatency() 16
2.6 PwrSnsr_GetMinimumSupportedFirmware() 16
2.7 PwrSnsr_GetTimeOut() 16
2.8 PwrSnsr_init() 17
2.9 PwrSnsr_reset() 18
2.10 PwrSnsr_self_test() 18
2.11 PwrSnsr_SetFetchLatency() 19
2.12 PwrSnsr_SetTimeOut() 19

3 Measurements 20
3.1 Commands Summary 20
3.2 PwrSnsr_FetchCWArray() 22
3.3 PwrSnsr_FetchCWPower() 23
3.4 PwrSnsr_FetchDutyCycle() 23
3.5 PwrSnsr_FetchEdgeDelay() 24
3.6 PwrSnsr_FetchFallTime() 24
3.7 PwrSnsr_FetchIEEEBottom() 25
3.8 PwrSnsr_FetchIEEETop() 25
3.9 PwrSnsr_FetchOfftime() 26
3.10 PwrSnsr_FetchOvershoot() 26
3.11 PwrSnsr_FetchPeriod() 27
3.12 PwrSnsr_FetchPowerArray() 28
3.13 PwrSnsr_FetchPRF() 30
3.14 PwrSnsr_FetchPulseCycleAvg() 30
3.15 PwrSnsr_FetchPulseOnAverage() 31
3.16 PwrSnsr_FetchPulsePeak() 31
3.17 PwrSnsr_FetchRiseTime() 32
3.18 PwrSnsr_FetchTimeArray() 33
3.19 PwrSnsr_FetchWaveform() 35
3.20 PwrSnsr_FetchWaveformMinMax() 36
3.21 PwrSnsr_FetchWidth() 37
3.22 PwrSnsr_MeasurePower() 37
3.23 PwrSnsr_MeasureVoltage() 38
3.24 PwrSnsr_ReadCWArray() 39
3.25 PwrSnsr_ReadCWPower() 40
3.26 PwrSnsr_ReadDutyCycle() 40
3.27 PwrSnsr_ReadEdgeDelay() 41
3.28 PwrSnsr_ReadFallTime() 41
3.29 PwrSnsr_ReadIEEEBottom() 42

3.30 PwrSnsr_ReadIEEETop() 42
3.31 PwrSnsr_ReadOfftime() 43
3.32 PwrSnsr_ReadOvershoot() 43
3.33 PwrSnsr_ReadPeriod() 44
3.34 PwrSnsr_ReadPowerArray() 45
3.35 PwrSnsr_ReadPRF() 47
3.36 PwrSnsr_ReadPulseCycleAvg() 47
3.37 PwrSnsr_ReadPulseOnAverage() 48
3.38 PwrSnsr_ReadPulsePeak() 48
3.39 PwrSnsr_ReadRiseTime() 49
3.40 PwrSnsr_ReadTimeArray() 50
3.41 PwrSnsr_ReadWaveform() 52
3.42 PwrSnsr_ReadWaveformMinMax() 53
3.43 PwrSnsr_ReadWidth() 54

4 Trigger 55
4.1 Commands Summary 55
4.2 PwrSnsr_GetTrigDelay() 56
4.3 PwrSnsr_GetTrigHoldoff() 56
4.4 PwrSnsr_GetTrigHoldoffMode() 57
4.5 PwrSnsr_GetTrigLevel() 57
4.6 PwrSnsr_GetTrigMode() 58
4.7 PwrSnsr_GetTrigPosition() 58
4.8 PwrSnsr_GetTrigSlope() 59
4.9 PwrSnsr_GetTrigSource() 59
4.10 PwrSnsr_GetTrigStatus() 60
4.11 PwrSnsr_GetTrigVernier() 60
4.12 PwrSnsr_SetTrigDelay() 61
4.13 PwrSnsr_SetTrigHoldoff() 61
4.14 PwrSnsr_SetTrigHoldoffMode() 62
4.15 PwrSnsr_SetTrigLevel() 62
4.16 PwrSnsr_SetTrigMode() 63
4.17 PwrSnsr_SetTrigOutMode() 63
4.18 PwrSnsr_SetTrigPosition() 64
4.19 PwrSnsr_SetTrigSlope() 64
4.20 PwrSnsr_SetTrigSource() 65
4.21 PwrSnsr_SetTrigVernier() 65

5 Acquisition 66
5.1 Commands Summary 66
5.2 PwrSnsr_Abort() 66
5.3 PwrSnsr_Clear() 67
5.4 PwrSnsr_EnableCapturePriority() 67
5.5 PwrSnsr_FetchExtendedWaveform() 68
5.6 PwrSnsr_GetInitiateContinuous() 69
5.7 PwrSnsr_InitiateAquisition() 69
5.8 PwrSnsr_SetInitiateContinuous() 70
5.9 PwrSnsr_Status() 70

6 Channel Functions 71
6.1 Commands Summary 71
6.2 PwrSnsr_GetAverage() 73
6.3 PwrSnsr_GetBandwidth() 73
6.4 PwrSnsr_GetCalFactor() 73
6.5 PwrSnsr_GetCalFactors() 74
6.6 PwrSnsr_GetChannelByIndex() 75
6.7 PwrSnsr_GetChannelCount() 76
6.8 PwrSnsr_GetCurrentTemp() 76

6.9 PwrSnsr_GetDistal() 77
6.10 PwrSnsr_GetEnabled() 77
6.11 PwrSnsr_GetEndGate() 78
6.12 PwrSnsr_GetFilterState() 78
6.13 PwrSnsr_GetFilterTime() 79
6.14 PwrSnsr_GetFirmwareVersion() 79
6.15 PwrSnsr_GetFrequency() 80
6.16 PwrSnsr_GetIsAvgSensor() 80
6.17 PwrSnsr_GetMesial() 81
6.18 PwrSnsr_GetModel() 81
6.19 PwrSnsr_GetOffsetdB() 82
6.20 PwrSnsr_GetPeakHoldDecay() 82
6.21 PwrSnsr_GetPeakHoldTracking() 83
6.22 PwrSnsr_GetProximal() 83
6.23 PwrSnsr_GetPulseUnits() 84
6.24 PwrSnsr_GetSerialNumber() 84
6.25 PwrSnsr_GetStartGate() 85
6.26 PwrSnsr_GetTempComp() 85
6.27 PwrSnsr_GetUnits() 86
6.28 PwrSnsr_SetAverage() 86
6.29 PwrSnsr_SetBandwidth() 87
6.30 PwrSnsr_SetCalFactor() 87
6.31 PwrSnsr_SetDistal() 88
6.32 PwrSnsr_SetEnabled() 88
6.33 PwrSnsr_SetEndGate() 89
6.34 PwrSnsr_SetFilterState() 89
6.35 PwrSnsr_SetFilterTime() 90
6.36 PwrSnsr_SetFrequency() 90
6.37 PwrSnsr_SetMesial() 91
6.38 PwrSnsr_SetOffsetdB() 91
6.39 PwrSnsr_SetPeakHoldDecay() 91
6.40 PwrSnsr_SetPeakHoldTracking() 92
6.41 PwrSnsr_SetProximal() 93
6.42 PwrSnsr_SetPulseUnits() 93
6.43 PwrSnsr_SetStartGate() 94
6.44 PwrSnsr_SetTempComp() 94
6.45 PwrSnsr_SetUnits() 95

7 Time Base Functions 96
7.1 Commands Summary 96
7.2 PwrSnsr_GetMaxTimebase() 96
7.3 PwrSnsr_GetTimebase() 97
7.4 PwrSnsr_GetTimespan() 97
7.5 PwrSnsr_SetTimebase() 98
7.6 PwrSnsr_SetTimespan() 98

8 Marker Funtions 99
8.1 Commands Summary 99
8.2 PwrSnsr_FetchArrayMarkerPower() 101
8.3 PwrSnsr_FetchIntervalAvg() 102
8.4 PwrSnsr_FetchIntervalFilteredMax() 103
8.5 PwrSnsr_FetchIntervalFilteredMin() 103
8.6 PwrSnsr_FetchIntervalMax() 104
8.7 PwrSnsr_FetchIntervalMaxAvg() 104
8.8 PwrSnsr_FetchIntervalMin() 105
8.9 PwrSnsr_FetchIntervalMinAvg() 105
8.10 PwrSnsr_FetchIntervalPkToAvg() 106
8.11 PwrSnsr_FetchMarkerAverage() 107

8.12 PwrSnsr_FetchMarkerDelta() 107
8.13 PwrSnsr_FetchMarkerMax() 108
8.14 PwrSnsr_FetchMarkerMin() 108
8.15 PwrSnsr_FetchMarkerRatio() 109
8.16 PwrSnsr_FetchMarkerRDelta() 109
8.17 PwrSnsr_FetchMarkerRRatio() 110
8.18 PwrSnsr_GetMarkerPixelPosition() 110
8.19 PwrSnsr_GetMarkerTimePosition() 111
8.20 PwrSnsr_ReadArrayMarkerPower() 112
8.21 PwrSnsr_ReadIntervalAvg() 113
8.22 PwrSnsr_ReadIntervalFilteredMax() 113
8.23 PwrSnsr_ReadIntervalFilteredMin() 114
8.24 PwrSnsr_ReadIntervalMax() 115
8.25 PwrSnsr_ReadIntervalMaxAvg() 115
8.26 PwrSnsr_ReadIntervalMin() 116
8.27 PwrSnsr_ReadIntervalMinAvg() 116
8.28 PwrSnsr_ReadIntervalPkToAvg() 117
8.29 PwrSnsr_ReadMarkerAverage() 117
8.30 PwrSnsr_ReadMarkerDelta() 118
8.31 PwrSnsr_ReadMarkerMax() 118
8.32 PwrSnsr_ReadMarkerMin() 119
8.33 PwrSnsr_ReadMarkerRatio() 120
8.34 PwrSnsr_ReadMarkerRDelta() 120
8.35 PwrSnsr_ReadMarkerRRatio() 121
8.36 PwrSnsr_SetMarkerPixelPosition() 121
8.37 PwrSnsr_SetMarkerTimePosition() 122

9 Display Functions 123
9.1 Commands Summary 123
9.2 PwrSnsr_GetVerticalCenter() 123
9.3 PwrSnsr_GetVerticalScale() 124
9.4 PwrSnsr_SetVerticalCenter() 124
9.5 PwrSnsr_SetVerticalScale() 125

10 Statistical Mode 126
10.1 Commands Summary 126
10.2 PwrSnsr_FetchCCDFPercent() 128
10.3 PwrSnsr_FetchCCDFPower() 128
10.4 PwrSnsr_FetchCCDFTrace() 129
10.5 PwrSnsr_FetchCursorPercent() 130
10.6 PwrSnsr_FetchCursorPower() 130
10.7 PwrSnsr_FetchStatMeasurementArray() 131
10.8 PwrSnsr_GetAcqStatusArray() 133
10.9 PwrSnsr_GetCapture() 134
10.10 PwrSnsr_GetCCDFTraceCount() 134
10.11 PwrSnsr_GetDiagStatusArray() 135
10.12 PwrSnsr_GetGating() 136
10.13 PwrSnsr_GetHorizontalOffset() 136
10.14 PwrSnsr_GetHorizontalScale() 137
10.15 PwrSnsr_GetPercentPosition() 137
10.16 PwrSnsr_GetPowerPosition() 138
10.17 PwrSnsr_GetTermAction() 138
10.18 PwrSnsr_GetTermCount() 139
10.19 PwrSnsr_GetTermTime() 139
10.20 PwrSnsr_SetCapture() 140
10.21 PwrSnsr_SetCCDFTraceCount() 140
10.22 PwrSnsr_SetGating() 141
10.23 PwrSnsr_SetHorizontalOffset() 141

10.24 PwrSnsr_SetHorizontalScale() 142
10.25 PwrSnsr_SetPercentPosition() 142
10.26 PwrSnsr_SetPowerPosition() 143
10.27 PwrSnsr_SetTermAction() 143
10.28 PwrSnsr_SetTermCount() 144
10.29 PwrSnsr_SetTermTime() 144
10.30 PwrSnsr_StatModeReset() 145

11 Sensor Info 146
11.1 Commands Summary 146
11.2 PwrSnsr_GetAttenuation() 147
11.3 PwrSnsr_GetFactoryCalDate() 147
11.4 PwrSnsr_GetFpgaVersion() 148
11.5 PwrSnsr_GetImpedance() 148
11.6 PwrSnsr_GetManufactureDate() 149
11.7 PwrSnsr_GetMaxFreqHighBandwidth() 149
11.8 PwrSnsr_GetMaxFreqLowBandwidth() 150
11.9 PwrSnsr_GetMinFreqHighBandwidth() 150
11.10 PwrSnsr_GetMinFreqLowBandwidth() 151
11.11 PwrSnsr_GetMinimumTrig() 151
11.12 PwrSnsr_GetPeakPowerMax() 152
11.13 PwrSnsr_GetPeakPowerMin() 152

12 User Calibration 153
12.1 Commands Summary 153
12.2 PwrSnsr_ClearUserCal() 154
12.3 PwrSnsr_GetExternalSkew() 154
12.4 PwrSnsr_GetInternalSkew() 155
12.5 PwrSnsr_GetSlaveSkew() 155
12.6 PwrSnsr_SaveUserCal() 156
12.7 PwrSnsr_SetExternalSkew(156
12.8 PwrSnsr_SetInternalSkew() 157
12.9 PwrSnsr_SetSlaveSkew() 157
12.10 PwrSnsr_Zero() 158
12.11 PwrSnsr_ZeroQuery() 158

13 Trace Function 159
13.1 Commands Summary 159
13.2 PwrSnsr_FetchDistal() 159
13.3 PwrSnsr_FetchMesial() 160
13.4 PwrSnsr_FetchProximal() 160
13.5 PwrSnsr_GetChanTraceCount() 161
13.6 PwrSnsr_GetSweepTime() 161
13.7 PwrSnsr_GetTimePerPoint() 162
13.8 PwrSnsr_GetTraceStartTime() 162

14 Multiple Pulse 163
14.1 Commands Summary 163
14.2 PwrSnsr_FetchAllMultiPulse() 163
14.2.1 Pulse Info 163

15 Memory Channels 165
15.1 Commands Summary 165
15.2 PwrSnsr_GetMemChanArchive() 165
15.3 PwrSnsr_LoadMemChanFromArchive() 166
15.4 PwrSnsr_SaveToMemoryChannel() 166

16 Modulated Measurements 167
16.1 Command Summary 167
16.2 PwrSnsr_GetIsAvailable() 167
16.3 PwrSnsr_GetIsRunning() 168
16.4 PwrSnsr_GetReadingPeriod() 168

17 Measurement Buffer 169
17.1 Measurement Buffer 169
17.2 PwrSnsr_AcquireMeasurements() 171
17.3 PwrSnsr_AdvanceReadIndex() 171
17.4 PwrSnsr_ClearBuffer() 171
17.5 PwrSnsr_ClearMeasurements() 172
17.6 PwrSnsr_GetBufferedAverageMeasurements() 173
17.7 PwrSnsr_GetBufferedMeasurementsAvailable() 173
17.8 PwrSnsr_GetContinuousCapture() 174
17.9 PwrSnsr_GetDuration() 174
17.10 PwrSnsr_GetDurations() 175
17.11 PwrSnsr_GetEndDelay() 175
17.12 PwrSnsr_GetEndQual() 176
17.13 PwrSnsr_GetGateMode() 176
17.14 PwrSnsr_GetMaxMeasurements() 177
17.15 PwrSnsr_GetMeasBuffEnabled() 177
17.16 PwrSnsr_GetMeasurementsAvailable() 178
17.17 PwrSnsr_GetMinMeasurements() 178
17.18 PwrSnsr_GetOverRan() 179
17.19 PwrSnsr_GetPeriod() 179
17.20 PwrSnsr_GetRdgsEnableFlag() 180
17.21 PwrSnsr_GetReturnCount() 180
17.22 PwrSnsr_GetSequenceNumbers() 181
17.23 PwrSnsr_GetSessionCount() 181
17.24 PwrSnsr_GetStartDelay() 182
17.25 PwrSnsr_GetStartMode() 182
17.26 PwrSnsr_GetStartQual() 183
17.27 PwrSnsr_GetStartTimes() 183
17.28 PwrSnsr_GetTimedOut() 184
17.29 PwrSnsr_GetWriteProtection() 184
17.30 PwrSnsr_QueryAverageMeasurements() 185
17.31 PwrSnsr_QueryDurations() 185
17.32 PwrSnsr_QueryMaxMeasurements() 186
17.33 PwrSnsr_QueryMinMeasurements() 187
17.34 PwrSnsr_QuerySequenceNumbers() 187
17.35 PwrSnsr_QueryStartTimes() 188
17.36 PwrSnsr_ResetContinuousCapture() 189
17.37 PwrSnsr_SetContinuousCapture() 189
17.38 PwrSnsr_SetDuration() 190
17.39 PwrSnsr_SetEndDelay() 190
17.40 PwrSnsr_SetEndQual() 191
17.41 PwrSnsr_SetGateMode() 191
17.42 PwrSnsr_SetMeasBuffEnabled() 192
17.43 PwrSnsr_SetPeriod() 192
17.44 PwrSnsr_SetRdgsEnableFlag() 193
17.45 PwrSnsr_SetReturnCount() 193
17.46 PwrSnsr_SetSessionCount() 194
17.47 PwrSnsr_SetSessionTimeout() 194
17.48 PwrSnsr_SetStartDelay() 195
17.49 PwrSnsr_SetStartMode() 195
17.50 PwrSnsr_SetStartQual() 196

17.51 PwrSnsr_SetWriteProtection() 196
17.52 PwrSnsr_StartAcquisition() 197
17.53 PwrSnsr_StopAcquisition() 197

18 Sensor RawIO 198
18.1 Command Summary 198
18.2 PwrSnsr_ReadByteArray() 198
18.3 PwrSnsr_ReadControl() 199
18.4 PwrSnsr_Write() 200

19 License Functions 201
19.1 Commands Summary 201
19.2 PwrSnsr_GetDongleSerialNumber() 201
19.3 PwrSnsr_GetExpirationDate() 201
19.4 PwrSnsr_GetNumberOfCals() 202
19.5 PwrSnsr_IsLicenseDongleConnected() 202

20 Sensor Simulation 203
20.1 Commands Summary 203
20.2 PwrSnsrSignalUnits 203
20.3 PwrSnsrSimSignalType 203
20.4 PwrSnsr_GetSimSignalAmplitude() 204
20.5 PwrSnsr_GetSimSignalCompression() 204
20.6 PwrSnsr_GetSimSignalDuty() 205
20.7 PwrSnsr_GetSimSignalModulation() 205
20.8 PwrSnsr_GetSimSignalPRF() 206
20.9 PwrSnsr_GetSimSignalType() 206
20.10 PwrSnsr_OpenSimMeter() 207
20.11 PwrSnsr_SetSimSignalAmplitude() 207
20.12 PwrSnsr_SetSimSignalCompression() 208
20.13 PwrSnsr_SetSimSignalDuty() 208
20.14 PwrSnsr_SetSimSignalModulation() 209
20.15 PwrSnsr_SetSimSignalPRF() 209
20.16 PwrSnsr_SetSimSignalType() 210

SCPI Functions

1.1 Command Summary

PwrSnsr_ReadSCPI()

PwrSnsr_ReadSCPIBytes()

PwrSnsr_ReadSCPIFromNamedParser()

PwrSnsr_SendSCPIBytes()

PwrSnsr_SendSCPICommand()

PwrSnsr_SendSCPIToNamedParser()

1.2 PwrSnsr_ReadSCPI()

EXPORT int

PwrSnsr_ReadSCPI (

SessionID	Vi,
int	ValueBufferSize,
$long^*$	ValueActualSize,
char	Value[],
int	Timeout

)

Read a SCPI string response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
ValueBufferSize	Number of elements in Value.
Value	Number of elements actually written to Value.
ValueActualSize	The string returned from the instrument SCPI interface.
Timeout	Time out in milliseconds for the read operation. Use -1 for infinite and -2 to use the existing time out value.

Returns

Success (0) or error code

1.3 PwrSnsr_ReadSCPIBytes()

EXPORT int

PwrSnsr_ReadSCPIBytes(

SessionID	Vi,
int	ValueBufferSize,
char	ValueActualSize,
$long^*$	Value[],
int	Timeout

)

Read a SCPI byte array response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
name	Name of the parser.If parser doesn't exist, return PWR_SNSR_ERROR_NULL_POINTER.
ValueBufferSize	Number of elements in Value.
Value	Number of elements actually written to Value.
ValueActualSize	The string returned from the instrument SCPI interface.
Timeout	Time out in milliseconds for the read operation. Use -1 for infinite and -2 to use the existing time out value.

Returns

Success (0) or error code

1.4 PwrSnsr_ReadSCPIFromNamedParser()

EXPORT int

PwrSnsr_ReadSCPIFromNamedParser(

SessionID	Vi,
const char*	name,
int	ValueBufferSize,
$long^*$	ValueActualSize,
char	Value[],
int	Timeout

)

Read a SCPI string response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
name	Name of the parser.If parser doesn't exist, return PWR_SNSR_ERROR_NULL_POINTER.
ValueBufferSize	Number of elements in Value.
Value	Number of elements actually written to Value.
ValueActualSize	The string returned from the instrument SCPI interface.
Timeout	Time out in milliseconds for the read operation. Use -1 for infinite and -2 to use the existing time out value.

Returns

Success (0) or error code

1.5 PwrSnsr_SendSCPIBytes()

EXPORT int

PwrSnsr_SendSCPIBytes(

SessionID	Vi,
int	CommandBufferSize,
char	Command[]

)

Send a SCPI command as a byte array.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
CommandBufferSize	Number of elements in Command.
Command	Command to send.

Returns

Success (0) or error code

1.6 PwrSnsr_SendSCPICommand()

EXPORT int

PwrSnsr_SendSCPICommand(

SessionID	Vi,
const char*	Command

)

Send a SCPI command to the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Command	SCPI command.

Returns

Success (0) or error code

1.7 PwrSnsr_SendSCPIToNamedParser()

EXPORT int

PwrSnsr_SendSCPIToNamedParser(

SessionID	Vi,
const char*	name,
const char*	Command

)

Send a SCPI command to the instrument using a named SCPI parser.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
name	Name of the parser. Creates a new parser if the name is not already used.
Command	SCPI command.

Returns

Success (0) or error code

Session Management

2.1 Commands Summary

PwrSnsr_ClearError()

PwrSnsr_close()

PwrSnsr_FindResources()

PwrSnsr_GetFetcchLatency()

PwrSnsr_GetMinimumSupportedFirmware()

PwrSnsr_GetTimeOut()

PwrSnsr_init()

PwrSnsr_reset()

PwrSnsr_self_test()

PwrSnsr_SetFetchLatency()

PwrSnsr_SetTimeOut()

2.2 PwrSnsr_ClearError()

EXPORT int

PwrSnsr_ClearError(

SessionID Vi

)

This function clears the error code and error description for the given session.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

2.3 PwrSnsr_close()

EXPORT int

PwrSnsr_close(

SessionID Vi

)

Closes the I/O session to the instrument. Driver methods and properties that access the instrument are not accessible after Close is called.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

2.4 PwrSnsr_FindResources()

EXPORT int

PwrSnsr_FindResources(

const char*	Delimiter,
int	ValueBufferSize,
char	Value[]

)

Returns a delimited string of available resources. These strings can be used in the initialize function to open a session to an instrument.

Parameters

Delimiter	The string used to delimit the list of resources ie. "|", " ", ";", etc.
ValueBufferSize	Number of elements in Value.
Value	The retrun string.

Returns

Success (0) or error code

2.5 PwrSnsr_GetFetchLatency()

EXPORT int

PwrSnsr_GetFetchLatency(

SessionID	Vi,
int	Latency

Get the period the library waits to update fetch measurements in ms.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Latency	Fetch latency in ms.

Returns

Success (0) or error code

2.6 PwrSnsr_GetMinimumSupportedFirmware()

EXPORT int

PwrSnsr_GetMinimumSupportedFirmware(

int	Version

)

Gets the minimum supported firmware as an integer. Format is YYYYMMDD.

Returns

Success (0) or error code

2.7 PwrSnsr_GetTimeOut()

EXPORT int

PwrSnsr_GetTimeOut()

SessionID	Vi,
$long^*$	Val

)

Returns the time out value for I/O in milliseconds.

Parameters

Vi

Val	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timeout	Time out in milliseconds. -1 denote infinite time out.

Returns

Success (0) or error code

2.8 PwrSnsr_init()

EXPORT int

PwrSnsr_init()

char	ResourceName,
SessionID	Vi

)

Initialize a communication session with a supported USB power sensor.

Parameters

ResourceName	Name of the resource. The resource descriptor is in the following format: USB::[VID]::[PID]::[Serial Number]::SNSR Where serial number is the USB power meter's serial number in decimal format, and the VID and PID are in hexidecimal format. e.g. For serial number 1234, VID of 0x1bfe and PID of 0x5500: USB::0x1BFE::0x5500::1234::SNSR Multiple channel synthetic meters can be defined by combining more than one descriptor separated by a semicolon. Channel assignment is determined by the order in the list, in other words CH1 would be the first listed resource, CH2 the second resource, etc. e.g. Define a synthetic peak power meter using serial number 1234 for CH1 and serial number 4242 for CH2: USB::0x1BFE::0x5500::1234::SNSR;USB::0x1BFE::0x5500::4242::SNSR
Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

2.9 PwrSnsr_reset()

EXPORT int

PwrSnsr_reset(

SessionID Vi

)

Places the instrument in a known state.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

2.10 PwrSnsr_self_test()

EXPORT int

PwrSnsr_self_test(

SessionID Vi,

int TestResult

)

Performs an instrument self test, waits for the instrument to complete the test, and queries the instrument for the results. If the instrument passes the test, TestResult is 0.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
TestResult	Error or success code.

Returns

Success (0) or error code

2.11 PwrSnsr_SetFetchLatency()

EXPORT int

PwrSnsr_SetFetchLatency(

SessionID	Vi,
int	Latency

)

Set the period the library waits to update fetch measurements in ms.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Latency	Fetch latency in ms.

Returns

Success (0) or error code

2.12 PwrSnsr_SetTimeOut()

EXPORT int

PwrSnsr_SetTimeOut(

SessionID	Vi,
long	Milliseconds

Sets the time out in milliseconds for I/O.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Milliseconds	Time out in milliseonds. Use -1 for infinite time out.

Returns

Success (0) or error code

Measurements

3.1 Commands Summary

PwrSnsr_FetchCWArray()

PwrSnsr_FetchCWPower()

PwrSnsr_FetchDutyCycle()

PwrSnsr_FetchEdgeDelay()

PwrSnsr_FetchFallTime()

PwrSnsr_FetchIEEEBottom()

PwrSnsr_FetchIEEETop()

PwrSnsr_FetchOfftime()

PwrSnsr_FetchOvershoot()

PwrSnsr_FetchPeriod()

PwrSnsr_FetchPowerArray()

PwrSnsr_FetchPRF()

PwrSnsr_FetchPulseCycleAvg()

PwrSnsr_FetchPulseOnAverage()

PwrSnsr_FetchPulsePeak()

PwrSnsr_FetchRiseTime()

PwrSnsr_FetchTimeArray()

PwrSnsr_FetchWaveform()

PwrSnsr_FetchWaveformMinMax()

PwrSnsr_FetchWidth()

PwrSnsr_MeasurePower()

PwrSnsr_MeasuredVoltage()

PwrSnsr_ReadCWArray()

PwrSnsr_ReadCWPower()

PwrSnsr_ReadDutyCycle()

PwrSnsr_ReadEdgeDelay()

PwrSnsr_ReadFallTime()

PwrSnsr_ReadIEEEBottom()

PwrSnsr_ReadIEEETop()

PwrSnsr_ReadOfftime()

PwrSnsr_ReadOvershoot()

PwrSnsr_ReadPeriod()

PwrSnsr_ReadPowerArray()

PwrSnsr_ReadPRF()

PwrSnsr_ReadPulseCycleAvg()

PwrSnsr_ReadPulseOnAverage()

PwrSnsr_ReadPulsePeak()

PwrSnsr_ReadRiseTime()

PwrSnsr_ReadTimeArray()

PwrSnsr_ReadWaveform()

PwrSnsr_ReadWaveformMinMac()

PwrSnsr_ReadWidth()

3.2 PwrSnsr_FetchCWArray()

EXPORT int

PwrSnsr_FetchCWArray(

SessionID	Vi,
const char*	Channel,
float*	PeakAverage,
PwrSnsrCondCodeEnum*	PeakAverageValid,
float*	PeakMax,
PwrSnsrCondCodeEnum*	PeakMaxValid,
float*	PeakMin,
PwrSnsrCondCodeEnum*	PeakMinValid,
float*	PeakToAvgRatio,
PwrSnsrCondCodeEnum*	PeakToAvgRatioValid

)

Returns the current average, maximum, minimum powers or voltages and the peak-to-average ratio of the specified channel. Units are the same as the channel units. Note the peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PeakAverage	PeakAverage Average power of the peak power envelope.
PeakAverageValid	Condition code.
PeakMax	maximum power of the peak power envelope.
PeakMaxValid	Condition code.
PeakMin	Minimum power of the peak power envelope.
PeakMinValid	Condition code.
PeakToAvgRatio	Peak to average ratio.
PeakToAvgRatioValid	Condition code.

Returns

Success (0) or error code

3.3 PwrSnsr_FetchCWPower()

EXPORT int

PwrSnsr_FetchCWPower(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the most recently acquired CW power.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	CW power in channel units.

Returns

Success (0) or error code

3.4 PwrSnsr_FetchDutyCycle()

EXPORT int

PwrSnsr_FetchDutyCycle(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the ratio of the pulse on-time to off-time.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.5 PwrSnsr_FetchEdgeDelay()

EXPORT int

PwrSnsr_FetchEdgeDelay(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.6 PwrSnsr_FetchFallTime()

EXPORT int

PwrSnsr_FetchFallTime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the interval between the last signal crossing of the distal line to the last signalcrossing of the proximal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.7 PwrSnsr_FetchIEEEBottom()

EXPORT int

PwrSnsr_FetchIEEEBottom(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.8 PwrSnsr_FetchIEEETop()

EXPORT int

PwrSnsr_FetchIEEETop(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.9 PwrSnsr_FetchOfftime()

EXPORT int

PwrSnsr_FetchOfftime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulsewidth).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.10 PwrSnsr_FetchOvershoot()

EXPORT int

PwrSnsr_FetchOvershoot(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.11 PwrSnsr_FetchPeriod()

EXPORT int

PwrSnsr_FetchPeriod(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the interval between two successive pulses. (Reciprocal of the Pulse RepetitionFrequency)

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.12 PwrSnsr_FetchPowerArray()

EXPORT int

PwrSnsr_FetchPowerArray(

SessionID	Vi,
const char*	Channel,
float*	PulsePeak,
PwrSnsrCondCodeEnum*	PulsePeakValid,
float*	PulseCycleAvg,
PwrSnsrCondCodeEnum*	PulseCycleAvgValid,
float*	PulseOnAvg,
PwrSnsrCondCodeEnum*	PulseOnValid,
float*	IEEETop,
PwrSnsrCondCodeEnum*	IEEETopValid,
float*	IEEEBottom,
PwrSnsrCondCodeEnum*	IEEEBottomValid,
float*	Overshoot,
PwrSnsrCondCodeEnum*	OvershootValid,
float*	Droop,
PwrSnsrCondCodeEnum*	DroopValid

)

Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform.

Measurements performed are: peak amplitude during the pulse, average amplitude over a full cycle of the pulse waveform, average amplitude during the pulse, IEEE top amplitude, IEEE bottom amplitude, and overshoot. Units are the same as the channel's units.

Note:

The pulse overshoot is returned in dB for logarithmic channel units, and percent for all other units. Also, the pulse ?ON interval used for peak and average calculations is defined by the SENSe:PULSe:STARTGT and :ENDGT time gating settings.

A full pulse (rise and fall) must be visible on the display to make average and peak pulse power measurements, and a full cycle of the waveform must be visible to calculate average cycle amplitude.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PulsePeak	The peak amplitude during the pulse.
PulsePeakValid	Condition code.
PulseCycleAvg	Average cycle amplitude.
PulseCycleAvgValid	Condition code.
PulseOnAvg	Average power of the ON portion of the pulse.
PulseOnValid	Condition code.
IEEETop	The IEEE-defined top line, i.e. the portion of a pulse waveform, which represents the second nominal state of a pulse.
IEEETopValid	Condition code.
IEEEBottom	The IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.
IEEEBottomValid	Condition code.
Overshoot	The difference between the distortion following a major transition and the IEEE top line in dB or percent,depending on the channel units.
OvershootValid	Condition code.
Droop	Pulse droop.
DroopValid	Condition code.

Returns

Success (0) or error code

3.13 PwrSnsr_FetchPRF()

EXPORT int

PwrSnsr_FetchPRF(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.14 PwrSnsr_FetchPulseCycleAvg()

EXPORT int

PwrSnsr_FetchPulseCycleAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum	IsValid,
float*	Val

)

Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.15 PwrSnsr_FetchPulseOnAverage()

EXPORT int

PwrSnsr_FetchPulseOnAverage(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Average power of the ON portion of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.16 PwrSnsr_FetchPulsePeak()

EXPORT int

PwrSnsr_FetchPulsePeak()

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the peak amplitude during the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.17 PwrSnsr_FetchRiseTime()

EXPORT int

PwrSnsr_FetchRiseTime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.18 PwrSnsr_FetchTimeArray()

EXPORT int

PwrSnsr_FetchTimeArray(

SessionID	Vi,
const char*	Channel,
float*	Frequency,
PwrSnsrCondCodeEnum*	FrequencyValid,
float*	Period,
PwrSnsrCondCodeEnum*	PeriodValid,
float*	Width,
PwrSnsrCondCodeEnum*	WidthValid,
float*	Offtime,
PwrSnsrCondCodeEnum*	OfftimeValid,
float*	DutyCycle,
PwrSnsrCondCodeEnum*	DutyCycleValid,
float*	Risetime,
PwrSnsrCondCodeEnum*	RisetimeValid,
float*	Falltime,
PwrSnsrCondCodeEnum*	FalltimeValid,
float*	EdgeDelay,
PwrSnsrCondCodeEnum*	EdgeDelayValid,
float*	Skew,
PwrSnsrCondCodeEnum*	SkewValid

)

Returns an array of the current automatic timing measurements performed on a periodic pulse waveform.

Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window.

Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions.

It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements. Note that the timing of the edge transitions is defined by the settings of the SENSe:PULSe:DISTal, :MESIal and :PROXimal settings; see the descriptions Forthose commands. Units are the same as the channel's units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Frequency	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid	Condition code.
Period	The interval between two successive pulses.
PeriodValid	Condition code.
Width	The interval between the first and second signal crossings of the mesial line.
WidthValid	Condition code.
Offtime	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid	Condition code.
DutyCycle	The ratio of the pulse on-time to period.
DutyCycleValid	Condition code.
Risetime	The interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.
RisetimeValid	Condition code.
Falltime	The interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.
FalltimeValid	Condition code.
EdgeDelay	Time offset from the trigger reference to the first mesial transition level of either slope on the waveform.
EdgeDelayValid	Condition code.
Skew	The trigger offset between the assigned trigger channel and this channel.
SkewValid	Condition code.

Returns

Success (0) or error code

3.19 PwrSnsr_FetchWaveform()

EXPORT int

PwrSnsr_FetchWaveform(

SessionID	Vi,
const char*	Channel,
int	WaveformArrayBufferSize,
float	WaveformArray[],
init*	WaveformArrayActualSize

)

Returns a previously acquired waveform for this channel. The acquisition must be made prior to calling this method. Call this method separately for each channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
WaveformArrayBufferSize	Size in bytes of the Waveform buffer.
WaveformArray	The array contains the average waveform. Units for the individual array elements are in the channel units setting.
WaveformArrayActualSize	Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code

3.20 PwrSnsr_FetchWaveformMinMax()

EXPORT int

PwrSnsr_FetchWaveformMinMax(

SessionID	VI,
const char*	Channel,
int	MinWaveformBufferSize,
float	MinWaveform[],
init*	MinWaveformActualSize,
int	MaxWaveformBufferSize,
float	MaxWaveform[],
init*	MaxWaveformActualSize,
int	WaveformArrayBufferSize,
float	WaveformArray[],
init*	WaveformArrayActualSize,

)

Returns the previously acquired minimum and maximum waveforms for this specified channel. The acquisition must be made prior to calling this method. Call this method separately for each channel.

Parameters

VI	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinWaveformBufferSize	Size in bytes of the MinWaveform buffer.
MinWaveform[]	This array contains the min waveform. Units for the individual array elements are in the channel units setting.
MinWaveformActualSize	Size in bytes of the data written to MinWaveform.
MaxWaveformBufferSize	Size in bytes of the MaxWaveform buffer.
MaxWaveform[]	This array contains the max waveform. Units for the individual array elements are in the channel units setting
MaxWaveformActualSize	Size in bytes of the data written to MaxWaveform.
WaveformArrayBufferSize	Size in bytes of the Waveform buffer.
WaveformArray[]	The array contains the average waveform. Units for the individual array elements are in the channel units setting.
WaveformArrayActualSize	Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code

3.21 PwrSnsr_FetchWidth()

EXPORT int

PwrSnsr_FetchWidth(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.22 PwrSnsr_MeasurePower()

EXPORT int

PwrSnsr_MeasurePower(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Return average power using a default instrument configuration in Modulated Mode and dBm units. Instrument remains stopped in Modulated Mode after a measurement.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Average power in dBm.

Returns

Success (0) or error code

3.23 PwrSnsr_MeasureVoltage()

EXPORT int

PwrSnsr_MeasureVoltage(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Return average voltage using a default instrument configuration in Modulated Mode and volts units. Instrument remains stopped in Modulated Mode after a measurement.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Average voltage in linear volts.

Returns

Success (0) or error code

3.24 PwrSnsr_ReadCWArray()

EXPORT int

PwrSnsr_ReadCWArray(

SessionID	Vi,
const char*	Channel,
float*	PeakAvearege,
PwrSnsrCondCodeEnum*	PeakAverageValid,
float*	PeakMax,
PwrSnsrCondCodeEnum*	PeakMaxValid,
float*	PeakMin,
PwrSnsrCondCodeEnum*	PeakMinValid,
float*	PeakToAvgRatio,
PwrSnsrCondCodeEnum*	PeakToAvgRatioValid

)

Returns the current average, maximum, minimum powers or voltages and the peak-toaverage ratio of the specified channel. Units are the same as the channel's units.

Note:

The peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
PeakAvearege,	Average power of the peak power envelope.
PeakAverageValid,	Condition code.
PeakMax,	Maximum power of the peak power envelope.
PeakMaxValid,	Condition code.
PeakMin,	Minimum power of the peak power envelope.
PeakMinValid,	Condition code.
PeakToAvgRatio,	Peak to average ratio.
PeakToAvgRatioValid	Condition code.

Returns

Success (0) or error code

3.25 PwrSnsr_ReadCWPower()

EXPORT int

PwrSnsr_ReadCWPower()

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Initiates a CW power acquisition and returns the result in channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.26 PwrSnsr_ReadDutyCycle()

EXPORT int

PwrSnsr_ReadDutyCycle()

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the ratio of the pulse on-time to off-time.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.27 PwrSnsr_ReadEdgeDelay()

EXPORT int

PwrSnsr_ReadEdgeDelay(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.28 PwrSnsr_ReadFallTime()

EXPORT int

PwrSnsr_ReadFallTime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.29 PwrSnsr_ReadIEEEBottom()

EXPORT int

PwrSnsr_ReadIEEEBottom(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.30 PwrSnsr_ReadIEEETop()

EXPORT int

PwrSnsr_ReadIEEETop(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.31 PwrSnsr_ReadOfftime()

EXPORT int

PwrSnsr_ReadOfftime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.32 PwrSnsr_ReadOvershoot()

EXPORT int

PwrSnsr_ReadOvershoot(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.33 PwrSnsr_ReadPeriod()

EXPORT int

PwrSnsr_ReadPeriod(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the interval between two successive pulses.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.34 PwrSnsr_ReadPowerArray()

EXPORT int

PwrSnsr_ReadPowerArray(

SessionID	Vi,
const char*	Channel,
float*	PulsePeak,
PwrSnsrCondCodeEnum*	PulsePeakValid,
float*	PulseCycleAvg,
PwrSnsrCondCodeEnum*	PulseCycleAgvValid,
float*	PulseOnAvg,
PwrSnsrCondCodeEnum*	PulseOnValid,
float*	IEEETop,
PwrSnsrCondCodeEnum*	IEEETopValid,
float*	IEEEBottom,
PwrSnsrCondCodeEnum*	IEEEBottomValid,
float*	Overshoot,
PwrSnsrCondCodeEnum*	OvershootValid,
float*	Droop,
PwrSnsrCondCodeEnum*	DroopValid

)

Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform.

Measurements performed are: peak amplitude during the pulse, average amplitude over a full cycle of thepulse waveform, average amplitude during the pulse, IEEE top amplitude, IEEE bottom amplitude, and overshoot.

Units are the same as the channel's units. Note the pulse overshoot is returned in dB for logarithmic channel units, and percent for all other units. Also, the pulse ON interval used for peak and average calculations is defined by the SENSe:PULSe:STARTGT and :ENDGT time gating settings.

A full pulse (rise and fall) must be visible on the display to make average and peak pulse power measurements, and a full cycle of the waveform must be visible to calculate average cycle amplitude.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
PulsePeak,	The peak amplitude during the pulse.
PulsePeakValid,	Condition code.
PulseCycleAvg,	Average cycle amplitude.
PulseCycleAgvValid,	Condition code.
PulseOnAvg,	Average power of the ON portion of the pulse.
PulseOnValid,	Condition code.
IEEETop,	The IEEE-defined top line, i.e. the portion of a pulse waveform, which represents the second nominal state of a pulse.
IEEETopValid,	Condition code.
IEEEBottom,	The IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.
IEEEBottomValid,	Condition code.
Overshoot,	The difference between the distortion following a major transition and the IEEE top line in dB or percent,depending on the channel units.
OvershootValid,	Condition code.
Droop,	Pulse droop.
DroopValid	Condition code.

Returns

Success (0) or error code

3.35 PwrSnsr_ReadPRF()

EXPORT int

PwrSnsr_ReadPRF(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.36 PwrSnsr_ReadPulseCycleAvg()

EXPORT int

PwrSnsr_ReadPulseCycleAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the average power of the entire pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.37 PwrSnsr_ReadPulseOnAverage()

EXPORT int

PwrSnsr_ReadPulseOnAverage(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Average power of the ON portion of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.38 PwrSnsr_ReadPulsePeak()

EXPORT int

PwrSnsr_ReadPulsePeak(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the peak amplitude during the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.39 PwrSnsr_ReadRiseTime()

EXPORT int

PwrSnsr_ReadRiseTime(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

3.40 PwrSnsr_ReadTimeArray()

EXPORT int

PwrSnsr_ReadTimeArray(

SessionID	Vi,
const char*	Channel,
float*	Frequency,
PwrSnsrCondCodeEnum*	FrequencyValid,
float*	Period,
PwrSnsrCondCodeEnum*	PeriodValid,
float*	Width,
PwrSnsrCondCodeEnum*	WidthValid,
float*	Offtime,
PwrSnsrCondCodeEnum*	OfftimeValid,
float*	DutyCycle,
PwrSnsrCondCodeEnum*	DutyCycleValid,
float*	Risetime,
PwrSnsrCondCodeEnum*	RisetimeValid,
float*	Falltime,
PwrSnsrCondCodeEnum*	FalltimeValid,
float*	EdgeDelay,
PwrSnsrCondCodeEnum*	EdgeDelayValid,
float*	Skew,
PwrSnsrCondCodeEnum*	SkewValid

)

Returns an array of the current automatic timing measurements performed on a periodic pulse waveform. Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window.

Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions. It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements.

Note:

The timing of the edge transitions is defined by the settings of the SENSe:PULSe:DISTal, :MESIal and :PROXimal settings; see the descriptions For those commands. Units are the same as the channel's units.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
Frequency,	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid,	Condition code.
Period,	The interval between two successive pulses.
PeriodValid,	Condition code.
Width,	The interval between the first and second signal crossings of the mesial line.
WidthValid,	Condition code.
Offtime,	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid,	Condition code.
DutyCycle,	The ratio of the pulse on-time to period.
DutyCycleValid,	Condition code.
Risetime,	The interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.
RisetimeValid,	Condition code.
Falltime,	The interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.
FalltimeValid,	Condition code.
EdgeDelay,	Time offset from the trigger reference to the first mesial transition level of either slope on the waveform.
EdgeDelayValid,	Condition code.
Skew,	The trigger offset between the assigned trigger channel and this channel.
SkewValid	Condition code

Returns

Success (0) or error code

3.41 PwrSnsr_ReadWaveform()

EXPORT int

PwrSnsr_ReadWaveform(

SessionID	Vi,
const char*	Channel,
init*	WaveformArrayBufferSize,
float	WaveformArray[],
init*	WaveformArrayActualSize

)

Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the waveform for this channel. Call FetchWaveform to obtain the waveforms for other channels.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
WaveformArrayBufferSize	Size in bytes of the Waveform buffer.e.
WaveformArray[]	The array contains the average waveform. Units for the individual array elements are in the channel units setting.
WaveformArrayActualSize	Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code

3.42 PwrSnsr_ReadWaveformMinMax()

EXPORT int

PwrSnsr_ReadWaveformMinMax(

SessionID	Vi,
const char*	Channel,
int	MinWaveformBufferSize,
float	MinWaveform[],
init*	MinWaveformActualSize,
int	MaxWaveformBufferSize,
float	MaxWaveform[],
init*	MaxWaveformActualSize,
int	WaveformArrayBufferSize,
float	WaveformArray[],
init*	WaveformArrayActualSize,

)

Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the min/max waveforms for this channel. Call FetchMinMaxWaveform to obtain the min/max waveforms for other channels.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinWaveformBufferSize	Size in bytes of the MinWaveform buffer.
MinWaveform[]	This array contains the min waveform. Units for the individual array elements are in the channel units setting.
MinWaveformActualSize	Size in bytes of the data written to MinWaveform.
MaxWaveformBufferSize	Size in bytes of the MaxWaveform buffer.
MaxWaveform[]	This array contains the max waveform. Units for the individual array elements are in the channel units setting.
MaxWaveformActualSize	Size in bytes of the data written to MaxWaveform.
WaveformArrayBufferSize	Size in bytes of the Waveform buffer.
WaveformArray[]	The array contains the average waveform. Units for the individual array elements are in the channel units setting.
WaveformArrayActualSize	Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code

3.43 PwrSnsr_ReadWidth()

EXPORT int

PwrSnsr_ReadWidth(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code

Trigger

4.1 Commands Summary

PwrSnsr_GetTrigDelay()

PwrSnsr_GetTrigHoldoff()

PwrSnsr_GetTrigHoldoffMode()

PwrSnsr_GetTrigLevel()

PwrSnsr_GetTrigMode()

PwrSnsr_GetTrigPosition()

PwrSnsr_GetTrigSlope()

PwrSnsr_GetTrigSource()

PwrSnsr_GetTrigStatus()

PwrSnsr_GetTrigVernier()

PwrSnsr_SetTrigDelay()

PwrSnsr_SetTrigHoldoff()

PwrSnsr_SetTrigHoldoffMode()

PwrSnsr_SetTrigLevel()

PwrSnsr_SetTrigMode()

PwrSnsr_SetTrigOutMode()

PwrSnsr_SetTrigPosition()

PwrSnsr_SetTrigSlope()

PwrSnsr_SetTrigSource()

PwrSnsr_SetTrigVernier()

4.2 PwrSnsr_GetTrigDelay()

EXPORT int

PwrSnsr_GetTrigDelay(

SessionID	Vi,
float*	Delay

)

Return the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position.

Positive values cause the actual trigger to occur after the trigger condition is met. This places the trigger event to the left of the trigger point on the display, and is useful for viewing events during a pulse, some fixed delay time after the rising edge trigger. Negative trigger delay places the trigger event to the right of the trigger point on the display, and is useful for looking at events before the trigger edge.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Delay	Delay return value.

Returns

Success (0) or error code

4.3 PwrSnsr_GetTrigHoldoff()

EXPORT int

PwrSnsr_GetTrigHoldoff(

SessionID	Vi,
float*	Holdoff

)

Return the trigger holdoff time in seconds.

Trigger holdoff is used to disable the trigger for a specified amount of time after each trigger event. The holdoff time starts immediately after each valid trigger edge, and will not permit any new triggers until the time has expired. When the holdoff time is up, the trigger re-arms, and the next valid trigger event (edge) will cause a new sweep.

This feature is used to help synchronize the power meter with burst waveforms such as a TDMA or GSM frame. The trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Holdoff	Holdoff trigger return value.

Returns

Success (0) or error code

4.4 PwrSnsr_GetTrigHoldoffMode()

EXPORT int

PwrSnsr_GetTrigHoldoffMode(

SessionID	Vi,
PwrSnsrHoldoffModeEnum*	HoldoffMode

)

Returns the holdoff mode to normal or gap holdoff.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
HoldoffMode	Holdoff mode .

Returns

Success (0) or error code

4.5 PwrSnsr_GetTrigLevel()

EXPORT int

PwrSnsr_GetTrigLevel(

SessionID	Vi,
float*	Level

)

Return the trigger level for synchronizing data acquisition with a pulsed input signal.

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Level	Trigger level in dBm.

Returns

Success (0) or error code

4.6 PwrSnsr_GetTrigMode()

EXPORT int

PwrSnsr_GetTrigMode(

SessionID	Vi,
PwrSnsrTriggerModeEnum*	Mode

)

Return the trigger mode for synchronizing data acquisition with pulsed signals.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Mode	Trigger mode.

Returns

Success (0) or error code

4.7 PwrSnsr_GetTrigPosition()

EXPORT int

PwrSnsr_GetTrigPosition(

SessionID	Vi,
PwrSnsrTriggerPositionEnum	Position

)

Return the position of the trigger event on displayed sweep.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Position	Trigger position.

Returns

Success (0) or error code

4.8 PwrSnsr_GetTrigSlope()

EXPORT int

PwrSnsr_GetTrigSlope(

SessionID	Vi,
PwrSnsrTriggerSlopEnum*	Slope

)

Return the trigger slope or polarity.

When set to positive, trigger events will be generated when a signal rising edge crosses the trigger level threshold. When negative, trigger events are generated on the falling edge of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Slope	Returns trigger slope.

Returns

Success (0) or error code

4.9 PwrSnsr_GetTrigSource()

EXPORT int

PwrSnsr_GetTrigSource(

SessionID	Vi,
PwrSnsrTriggerSourceEnum*	Source

)

Set the signal the power meter monitors for a trigger. It can be channel external input, or independent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Source	Trigger source.

Returns

Success (0) or error code

4.10 PwrSnsr_GetTrigStatus()

EXPORT int

PwrSnsr_GetTrigStatus(

SessionID	Vi,
PwrSnsrTriggerStatusEnum*	Status

)

The status of the triggering system. Update rate is controlled by FetchLatency setting.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Status	Status of the trigger.

Returns

Success (0) or error code

4.11 PwrSnsr_GetTrigVernier()

EXPORT int

PwrSnsr_GetTrigVernier(

SessionID	Vi,
float*	Vernier

)

Return the fine position of the trigger event on the power sweep.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Vernier	Trigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code

4.12 PwrSnsr_SetTrigDelay()

EXPORT int

PwrSnsr_SetTrigDelay(

SessionID	Vi,
float*	Delay

)

Sets the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position.

Positive values cause the actual trigger to occur after the trigger condition is met. This places the trigger event to the left of the trigger point on the display, and is useful for viewing events during a pulse, some fixed delay time after the rising edge trigger. Negative trigger delay places the trigger event to the right of the trigger point on the display, and is useful for looking at events before the trigger edge

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Delay	Trigger delay in seconds.

Returns

Success (0) or error code

4.13 PwrSnsr_SetTrigHoldoff()

EXPORT int

PwrSnsr_SetTrigHoldoff(

SessionID	Vi,
float*	Holdoff

)

Sets the trigger holdoff time in seconds.

Trigger holdoff is used to disable the trigger for a specified amount of time after each trigger event. The holdoff time starts immediately after each valid trigger edge, and will not permit any new triggers until the time has expired. When the holdoff time is up, the trigger re-arms, and the next valid trigger event (edge) will cause a new sweep. This feature is used to help synchronize the power meter with burst waveforms such as a TDMA or GSM frame. The trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Holdoff	Trigger holdoff in seconds.

Returns

Success (0) or error code

4.14 PwrSnsr_SetTrigHoldoffMode()

EXPORT int

PwrSnsr_SetTrigHoldoffMode(

SessionID	Vi,
PwrSnsrHoldoffModeEnum	HoldoffMode

)

Sets the holdoff mode to normal or gap holdoff.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
HoldoffMode	Holdoff mode.

Returns

Success (0) or error code

4.15 PwrSnsr_SetTrigLevel()

EXPORT int

PwrSnsr_SetTrigLevel(

SessionID	Vi,
float*	Level

)

Set the trigger level for synchronizing data acquisition with a pulsed input signal.

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Level	Trigger level in dBm.

Returns

Success (0) or error code

4.16 PwrSnsr_SetTrigMode()

EXPORT int

PwrSnsr_SetTrigMode(

SessionID	Vi,
PwrSnsrTriggerModeEnum	Mode

)

Set the trigger mode for synchronizing data acquisition with pulsed signals.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Mode	Trigger mode.

Returns

Success (0) or error code

4.17 PwrSnsr_SetTrigOutMode()

EXPORT int

PwrSnsr_SetTrigOutMode(

SessionID	Vi,
const char*	Channel,
int	Mode

)

Sets the trigger out/mult io mode. Setting trigger mode overrides this command.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Mode	Trigger out/multi IO mode.

Returns

Success (0) or error code

4.18 PwrSnsr_SetTrigPosition()

EXPORT int

PwrSnsr_SetTrigPosition(

SessionID Vi,
PwrSnsrTriggerPositionEnum Position

)

Set the position of the trigger event on displayed sweep.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Position	Trigger position.

Returns

Success (0) or error code

4.19 PwrSnsr_SetTrigSlope()

EXPORT int

PwrSnsr_SetTrigSlope(

SessionID Vi,
PwrSnsrTriggerSlopeEnum Slope

)

Sets the trigger slope or polarity.

When set to positive, trigger events will be generated when a signal rising edge crosses the trigger level threshold. When negative, trigger events are generated on the falling edge of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Slope	Trigger slop or polarity.

Returns

Success (0) or error code

4.20 PwrSnsr_SetTrigSource()

EXPORT int

PwrSnsr_SetTrigSource(

SessionID	Vi,
PwrSnsrTriggerSourceEnum	Source

)

Get the signal the power meter monitors for a trigger. It can be channel external input, or independent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Source	External or independent trigger signal.

Returns

Success (0) or error code

4.21 PwrSnsr_SetTrigVernier()

EXPORT int

PwrSnsr_SetTrigVernier(

SessionID	Vi,
float*	Vernier

)

Set the fine position of the trigger event on the power sweep.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Vernier	rigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code

Acquisition

5.1 Commands Summary

PwrSnsr_Abort()

PwrSnsr_Clear()

PwrSnsr_EnableCapturePriority()

PwrSnsr_FetchExtendedWaveform()

PwrSnsr_GetInitiateContinuous()

PwrSnsr_InitiateAquisition()

PwrSnsr_SetInitiateContinuous()

PwrSnsr_Status()

5.2 PwrSnsr_Abort()

EXPORT int

PwrSnsr_Abort(

SessionID Vi,

)

Terminates any measurement in progress and resets the state of the trigger system.

Note:

Abort will leave the measurement in a stopped condition with all current measurements cleared.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

5.3 PwrSnsr_Clear()

EXPORT int

PwrSnsr_Clear(

SessionID	Vi,

)

Clear all data buffers. Clears averaging filters to empty.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

5.4 PwrSnsr_EnableCapturePriority()

EXPORT int

PwrSnsr_EnableCapturePriority(

SessionID	Vi,
const char*	Channel
int	Enabled

)

Sets the 55 series power meter to a buffered capture mode and disables real time processing.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Enabled	Set to 1, enables buffered mode. Set to zero, disables capturepriority(default).

Returns

Success (0) or error code

5.5 PwrSnsr_FetchExtendedWaveform()

EXPORT int

PwrSnsr_FetchExtendedWaveform(

SessionID	Vi,
const char*	Channel,
int	WaveformArrayBufferSize,
float	WaveformArray[],
int*	WaveformArrayActualSize,
int	Count

)

When capture priority is enabled, returns up to 100000 points of trace data based on the current timebase starting at the current trigger delay point

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
WaveformArrayBufferSize,	Number of elements in the WaveformArray buffer.
WaveformArray[],	Waveform buffer.
WaveformArrayActualSize,	Number of elements updated with data.
Count	Number of points to capture

Returns

Success (0) or error code

5.6 PwrSnsr_GetInitiateContinuous()

EXPORT int

PwrSnsr_GetInitiateContinuous(

SessionID	Vi,
int*	InitiateContinuous

)

Get the data acquisition mode for single or free-run measurements.

If INITiate:CONTinuous is set to ON, the instrument immediately begins taking measurements (Modulated, CW and Statistical Modes), or arms its trigger and takes a measurement each time a trigger occurs (Pulse Mode). If set to OFF, the measurement will begin (or be armed) as soon as the INITiate command is issued, and will stop once the measurement criteria (averaging, filtering or sample count) has been satisfied. Note that INITiate:IMMediate and READ commands are invalid when INITiate:CONTinuous is set to ON; however, by convention this situation does not result in a SCPI error.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
InitiateContinuous	Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code

5.7 PwrSnsr_InitiateAquisition()

EXPORT int

PwrSnsr_InitiateAquisition(

SessionID	Vi,

)

Starts a single measurement cycle when INITiate:CONTinuous is set to OFF.

In Modulated Mode, the measurement will complete once the power has been integrated for the full FILTer time. In Pulse Mode, enough trace sweeps must be triggered to satisfy the AVERaging setting. In Statistical Mode, acquisition stops once the terminal condition(s) are met. In each case, no reading will be returned until the measurement is complete. This command is not valid when INITiate:CONTinuous is ON, however, by convention this situation does not result in a SCPI error

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

5.8 PwrSnsr_SetInitiateContinuous()

EXPORT int

PwrSnsr_SetInitiateContinuous(

SessionID	Vi,
int	InitiateContinuous

)

Set the data acquisition mode for single or free-run measurements.

If INITiate:CONTinuous is set to ON, the instrument immediately begins taking measurements (Modulated, CW and Statistical Modes), or arms its trigger and takes a measurement each time a trigger occurs (Pulse Mode). If set to OFF, the measurement will begin (or be armed) as soon as the INITiate command is issued, and will stop once the measurement criteria (averaging, filtering or sample count) has been satisfied. Note that INITiate:IMMediate and READ commands are invalid when INITiate:CONTinuous is set to ON; however, by convention this situation does not result in a SCPI error.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
InitiateContinuous	Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code

5.9 PwrSnsr_Status()

EXPORT int

PwrSnsr_Status(

SessionID	Vi,
PwrSnsrAcquisitionStatusEnum*	Val

)

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Val	Status out parameter.

Returns

Success (0) or error code

Channel Functions

6.1 Commands Summary

PwrSnsr_GetAverage()

PwrSnsr_GetBandwidth()

PwrSnsr_GetCallFactor()

PwrSnsr_GetCallFactors()

PwrSnsr_GetChannelByIndex()

PwrSnsr_GetChannelCount()

PwrSnsr_GetCurrentTemp()

PwrSnsr_GetDistal()

PwrSnsr_GetEnabled()

PwrSnsr_GetEndGate()

PwrSnsr_GetFilterState()

PwrSnsr_GetFilterTime()

PwrSnsr_GetFirmwareVersion()

PwrSnsr_GetFrequency()

PwrSnsr_GetIsAvgSensor()

PwrSnsr_GetMesial()

PwrSnsr_GetModel()

PwrSnsr_GetOffsetdB()

PwrSnsr_GetPeakHoldDecay()

PwrSnsr_GetPeakHoldTracking()

PwrSnsr_GEtProximal()

PwrSnsr_GetPulseUnits()

PwrSnsr_GetSerialNumber()

PwrSnsr_GetStartGate()

PwrSnsr_GetTempComp()

PwrSnsr_GetUnits()

PwrSnsr_SetAverage()

PwrSnsr_SetBandwidth()

PwrSnsr_SetCalFactor()

PwrSnsr_SetDistal()

PwrSnsr_SetEnabled()

PwrSnsr_SetEndGate()

PwrSnsr_SetFilterState()

PwrSnsr_SetFilterTime()

PwrSnsr_SetFrequency()

PwrSnsr_SetMesial()

PwrSnsr_SetOffsetdB()

PwrSnsr_SetPeaskHoldDecay()

PwrSnsr_SetPeakHoldTracking()

PwrSnsr_SetProximal()

PwrSnsr_SetPulseUnits()

PwrSnsr_SetStartGate()

PwrSnsr_SetTempComp()

PwrSnsr_SetUnits()

6.2 PwrSnsr_GetAverage()

EXPORT int

PwrSnsr_GetAverage(

SessionID	Vi,
const char*	Channel,
int*	Average

)

Return the average measurements of the traces in the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
channel	Channel number. For single instruments, set this to "CH1".
Average	Trace measurements average value.

Returns

Success (0) or error code

6.3 PwrSnsr_GetBandwidth()

EXPORT int

PwrSnsr_GetBandwidth(

SessionID	Vi,
const char*	Channel,
PwrSnsrBandwidthEnum*	Bandwidth

)

Get the sensor video bandwidth for the selected sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Bandwith	Return video bandwith of selected sensro.

Returns

Success (0) or error code

6.4 PwrSnsr_GetCalFactor()

EXPORT int

PwrSnsr_GetCalFactor(

SessionID	Vi,
const char*	Channel,
float*	CalFactor

)

Get the frequency calibration factor currently in use on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CalFactro	Return the frequency calibration factor of selected channel.

Returns

Success (0) or error code

6.5 PwrSnsr_GetCalFactors()

EXPORT int

PwrSnsr_GetCalFactors(

SessionID	Vi,
const char*	Channel,
float*	MaxFrequency,
float*	MinFrequency,
int	FrequencyListBufferSize,
float*	FrequencyList[],
int*	FrequencyListActualSize,
int	CalFactorListBufferSize,
float*	CalFactorList[],
int*	CalFactroListActualSize,
PwrSnsrBandwidthEnum	Bandwidth

)

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
MaxFrequency,	Maximum RF frequency measureable by this channel.
MinFrequency,	Minimum RF frequency measureable by this channel.
FrequencyListBufferSize,	Number of elements in FrequencyList.
FrequencyList[],	List of frequencies correlated to the cal factors.
FrequencyListActualSize,	Actual number of elements returned in FrequencyList.
CalFactorListBufferSize,	Number of elements in CalFactorList.
CalFactorList[],	List of cal factors correlated to the frequencies.
CalFactroListActualSize,	Actual number of elements returned in CalFactorList.
Bandwidth	Bandwidth of interest. Cal factors for low and high bandwidth are different.

Returns

Success (0) or error code

6.6 PwrSnsr_GetChannelByIndex()

EXPORT int

PwrSnsr_GetChannelByIndex(

SessionID	Vi,
int	BuffSize,
char	Channel[],
int	Index

)

Gets the channel name by zero index. Note: SCPI commands use a one-based index.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
BuffSize	Buffer size for Channel
Channel	Channel number buffer.
Index	the index of the channel

Returns

Success (0) or error code

6.7 PwrSnsr_GetChannelCount()

EXPORT int

PwrSnsr_GetChannelCount(

SessionID	Vi,
int*	Count

)

Get the number of channels available.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Count	Numver of channels

Returns

Success (0) or error code

6.8 PwrSnsr_GetCurrentTemp()

EXPORT int

PwrSnsr_GetCurrentTemp()

SessionID	Vi,
const char*	Channel
double*	CurrentTemp

)

Get current sensor internal temperature in degrees C.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CurrentTemp	Sensor internal temperature in celcius.

Returns

Success (0) or error code

6.9 PwrSnsr_GetDistal()

EXPORT int

PwrSnsr_GetDistal(

SessionID	Vi,
const char*	Channel,
float*	Distal

)

Get the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Distal	Pulse amplitude percentage

Returns

Success (0) or error code

6.10 PwrSnsr_GetEnabled()

EXPORT int

PwrSnsr_GetEnabled(

SessionID	Vi,
const char*	Channel,
int*	Enabled

)

Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Enabled	Boolean. 1 for enabled; 0 for disabled.

Returns

Success (0) or error code

6.11 PwrSnsr_GetEndGate()

EXPORT int

PwrSnsr_GetEndGate(

SessionID	Vi,
const char*	Channel,
float*	EndGate

)

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EndGate	Retrun point on pulse used to define the end of the pulse's active interval.

Returns

Success (0) or error code

6.12 PwrSnsr_GetFilterState()

EXPORT int

PwrSnsr_GetFilterState(

SessionID	Vi,
const char*	Channel,
PwrSnsrFilterStateEnum*	FilterState

)

Get the current setting of the integration filter on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FilterState	State of the filter.

Returns

Success (0) or error code

6.13 PwrSnsr_GetFilterTime()

EXPORT int

PwrSnsr_GetFilterTime(

SessionID	Vi,
const char*	Channel,
float*	FilterTiem

)

Get the current length of the integration filter on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FilterTime	Filter time in seconds.

Returns

Success (0) or error code

6.14 PwrSnsr_GetFirmwareVersion()

EXPORT int

PwrSnsr_GetFirmwareVersion(

SessionID	Vi,
const char*	Channel,
int	FirwareVersionBufferSize,
char	FirmwareVersion[]

)

Returns the firmware version of the power meter associated with this channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FirmwareVersionBufferSize	Size of the FirmwareVersion buffer.
FirmwareVersion	Buffer to hold the firmware version.

Returns

Success (0) or error code

6.15 PwrSnsr_GetFrequency()

EXPORT int

PwrSnsr_GetFrequency(

SessionID	Vi,
const char*	Channel,
floatI	Frequency

)

Get the RF frequency for the current sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Frequency	RF Frequency in Hz.

Returns

Success (0) or error code

6.16 PwrSnsr_GetIsAvgSensor()

EXPORT int

PwrSnsr_GetIsAvgSensor(

SessionID	Vi,
const char*	Channel,
int*	IsAvgSensor

)

Retruns true if sensor is average responding (not peak detecting).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsAvgSensor	True if sensor is average responding.

Returns

Success (0) or error code

6.17 PwrSnsr_GetMesial()

EXPORT int

PwrSnsr_GetMesial(

SessionID	Vi,
const char*	Channel,
float*	Mesial

)

Get the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Mesial	Pulse amplitude percentage

Returns

Success (0) or error code

6.18 PwrSnsr_GetModel()

EXPORT int

PwrSnsr_GetModel(

SessionID	Vi,
const char*	Channel,
int	ModelBufferSize,
Char	Model[]

)

Gets the model of the meter connected to the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ModelBufferSize	Size of the buffer.
Model	Buffer where the model is read into.

Returns

Success (0) or error code

6.19 PwrSnsr_GetOffsetdB()

EXPORT int

PwrSnsr_GetOffsetdB(

SessionID	Vi,
const char*	Channel,
float*	OffsetdB

)

Get a measurement offset in dB for the selected sensor.

This setting is used to compensate for external couplers, attenuators or amplifiers in the RF signal path ahead of the power sensor

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
offsetdB	Offset in dB fo the selected sensor.

Returns

Success (0) or error code

6.20 PwrSnsr_GetPeakHoldDecay()

EXPORT int

PwrSnsr_GetPeakHoldDecay(

SessionID	Vi,
const char*	Channel,
int*	EnvelopeAverage

)

Get the number of min/max traces averaged together to form the peak hold measurement results on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeAverage	Min/max averaged togerther to form the peak hold.

Returns

Success (0) or error code

6.21 PwrSnsr_GetPeakHoldTracking()

EXPORT int

PwrSnsr_GetPeakHoldTracking(

SessionID	Vi,
const char*	Channel,
int*	EnvelopeTracking

)

Returns whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is indepedent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeTracking	Out boolean parameter value.

Returns

Success (0) or error code

6.22 PwrSnsr_GetProximal()

EXPORT int

PwrSnsr_GetProximal(

SessionID	Vi,
const char*	Channel,
float*	Proximal

)

Get the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Proximal	Pulse amplitude percentage

Returns

Success (0) or error code

6.23 PwrSnsr_GetPulseUnits()

EXPORT int

PwrSnsr_GetPulseUnits(

SessionID	Vi,
const char*	Channel,
PwrSnsrPulseUnitsEnum	Units

)

Get the units for entering the pulse distal, mesial and proximal levels.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Units	PwrSnsrPulseUnitsEnum Pulse caluclationunits (watts or volts)

Returns

Success (0) or error code

6.24 PwrSnsr_GetSerialNumber()

EXPORT int

PwrSnsr_GetSerialNumber(

SessionID	Vi,
const char*	Channel,
int	SerialNumberBufferSize,
char	SerialNumber[]

)

Gets the serial number of the sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
SerialNumberBufferSize	Size in bytes of Serial number.
SerialNumber	Out parameter. ASCII string serial numnber.

Returns

Success (0) or error code

6.25 PwrSnsr_GetStartGate()

EXPORT int

PwrSnsr_GetStartGate(

SessionID	Vi,
const char*	Channel,
float*	StartGate

)

Get the point on a pulse, which is used to define the beginning of the pulse's active interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
StartGate	Point used to define the beginning of the pulse's active interval.

Returns

Success (0) or error code

6.26 PwrSnsr_GetTempComp()

EXPORT int

PwrSnsr_GetTempComp(

SessionID	Vi,
const char*	Channel,
int*	TempComp

)

Get the state of the peak sensor temperature compensation system.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TempComp	Boolean 1 for on; 0 for off.

Returns

Success (0) or error code

6.27 PwrSnsr_GetUnits()

EXPORT int

PwrSnsr_GetUnits(

SessionID	Vi,
const char*	Channel,
PwrSnsrUnitsEnum	Units

)

Get units for the selected channel.

Voltage is calculated with reference to the sensor input impedance. Note that for ratiometric results, logarithmic units will always return dBr (dB relative) while linear units return percent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Units	Units of the selected channel.

Returns

Success (0) or error code

6.28 PwrSnsr_SetAverage()

EXPORT int

PwrSnsr_SetAverage(

SessionID	Vi,
const char*	Channel,
int	Average

)

Set the number of traces averaged together to form the measurement result on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Average	Number of traces to be average together.

Returns

Success (0) or error code

6.29 PwrSnsr_SetBandwidth()

EXPORT int

PwrSnsr_SetBandwidth(

SessionID	Vi,
const char*	Channel,
PwrSnsrBandwidthEnum	Bandwidth

)

Set the sensor video bandwidth for the selected sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Bandwidth	Bandwidth of the selected sensor.

Returns

Success (0) or error code

6.30 PwrSnsr_SetCalFactor()

EXPORT int

PwrSnsr_SetCalFactor(

SessionID	Vi,
const char*	Channel,
float	CalFactor

)

Set the frequency calibration factor currently in use on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Channel number. For single instruments, set this to "CH1".
Val	Frequency calibration factor.

Returns

Success (0) or error code

6.31 PwrSnsr_SetDistal()

EXPORT int

PwrSnsr_SetDistal(

SessionID	Vi,
const char*	Channel,
float	Distal

)

Set the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
channel	Channel number. For single instruments, set this to "CH1".
Distal	Pulse amplitude percentage.

Returns

Success (0) or error code

6.32 PwrSnsr_SetEnabled()

EXPORT int

PwrSnsr_SetEnabled(

SessionID	Vi,
const char*	Channel,
int	Enabled

)

Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Enabled	Boolean: 1 for enabled; 0 for disabled.

Returns

Success (0) or error code

6.33 PwrSnsr_SetEndGate()

EXPORT int

PwrSnsr_SetEndGate(

SessionID	Vi,
const char*	Channel,
float	EndGate

)

Set the point on a pulse, which is used to define the end of the pulse's active interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EndGate	Point that defines the end of the pulse's active interval.

Returns

Success (0) or error code

6.34 PwrSnsr_SetFilterState()

EXPORT int

PwrSnsr_SetFilterState(

SessionID	Vi,
const char*	Channel,
PwrSnsrFilterStateEnum	FilterState

)

Set the current setting of the integration filter on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FilterState	State of the filter.

Returns

Success (0) or error code

6.35 PwrSnsr_SetFilterTime()

EXPORT int

PwrSnsr_SetFilterTime(

SessionID	Vi,
const char*	Channel,
float	FilterTime

)

Set the current length of the integration filter on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FilterTime	Filter time in seconds.

Returns

Success (0) or error code

6.36 PwrSnsr_SetFrequency()

EXPORT int

PwrSnsr_SetFrequency(

SessionID	Vi,
const char*	Channel,
float	Frequency

)

Set the RF frequency for the current sensor, and apply the appropriate frequency calibration factor from the sensor internal table.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Frequency	RF Frequency in Hz.

Returns

Success (0) or error code

6.37 PwrSnsr_SetMesial()

EXPORT int

PwrSnsr_SetMesial(

SessionID	Vi,
const char*	Channel,
float	Mesial

)

Set the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Mesial	Pulse amplitude percentage.

Returns

Success (0) or error code

6.38 PwrSnsr_SetOffsetdB()

EXPORT int

PwrSnsr_SetOffsetdB(

SessionID	Vi,
const char*	Channel,
float	OffsetdB

)

Set a measurement offset in dB for the selected sensor.

Compensate for external couplers, attenuators or amplifiers in the RF signal path ahead of the power sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
OffsetdB	Offset in dB for the selected sensor.

Returns

Success (0) or error code

6.39 PwrSnsr_SetPeakHoldDecay()

EXPORT int

PwrSnsr_SetPeakHoldDecay(

SessionID	Vi,
const char*	Channel,
int	PeakHoldDecay

)

Set the number of min/max traces averaged together to form the peak hold measurement results on the selected channel

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PeakHoldDecay	Peak hold decay value.

Returns

Success (0) or error code

6.40 PwrSnsr_SetPeakHoldTracking()

EXPORT int

PwrSnsr_SetPeakHoldTracking(

SessionID	Vi,
const char*	Channel,
int	EnvelopeTracking

)

Sets whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is indepedent

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeTracking	Boolean value: True to set peak hold tracking on.

Returns

Success (0) or error code

6.41 PwrSnsr_SetProximal()

EXPORT int

PwrSnsr_SetProximal(

SessionID	Vi,
const char*	Channel,
float	Proximal

)

Set the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Proximal	Pulse amplitude percentage

Returns

Success (0) or error code

6.42 PwrSnsr_SetPulseUnits()

EXPORT int

PwrSnsr_SetPulseUnits(

SessionID	Vi,
const char*	Channel,
PwrSnsrPulseUnitsEnum	PwrSnsrPulseUnitsEnum

)

Set the units for entering the pulse distal, mesial and proximal levels.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PwrSnsrPulseUnitsEnum	Units

Returns

Success (0) or error code

6.43 PwrSnsr_SetStartGate()

EXPORT int

PwrSnsr_SetStartGate(

SessionID	Vi,
const char*	Channel,
float	StartGate

)

Set the state of the peak sensor temperature compensation system.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
StartGate	Pulse point

Returns

Success (0) or error code

6.44 PwrSnsr_SetTempComp()

EXPORT int

PwrSnsr_SetTempComp(

SessionID	Vi,
const char*	Channel,
int	TempComp

)

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TempComp	Boolean: 1 for on; 0 for off.

Returns

Success (0) or error code

6.45 PwrSnsr_SetUnits()

EXPORT int

PwrSnsr_SetUnits(

SessionID	Vi,
const char*	Channel,
PwrSnsrUnitsEnum	Units

)

Set units for the selected channel.

Voltage is calculated with reference to the sensor input impedance.

Note:

Ratiometric results, logarithmic units will always return dBr (dB relative) while linear units return percent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
	Units for selected channel.

Returns

Success (0) or error code

Time Base Functions

7.1 Commands Summary

PwrSnsr_GetMaxTimebase()

PwrSnsr_GetTimebase()

PwrSnsr_GetTimespan()

PwrSnsr_SetTimebase()

PwrSnsr_SetTimespan()

7.2 PwrSnsr_GetMaxTimebase()

EXPORT int

PwrSnsr_GetMaxTimebase(

SessionID	Vi,
float	MaxTimebase

)

Gets the maximum timebase setting available.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MaxTimease	Maximum timebase setting available.

Returns

Success (0) or error code

7.3 PwrSnsr_GetTimebase()

EXPORT int

PwrSnsr_GetTimebase(

SessionID	Vi,
float	Timebase

)

Get the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 (or max timebase) sec in a 1-2-5 sequence.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timebase	Pulse mode timebase in seconds/division

Returns

Success (0) or error code

7.4 PwrSnsr_GetTimespan()

EXPORT int

PwrSnsr_GetTimespan(

SessionID	Vi,
float	Timespan

)

Get the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timespan	Return horizontal time span

Returns

Success (0) or error code

7.5 PwrSnsr_SetTimebase()

EXPORT int

PwrSnsr_SetTimebase(

SessionID	Vi,
float	Timebase

)

Set the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 sec (or max timebase) in a 1-2-5 sequence.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timebase	Pulse mode timebase in seconds/division

Returns

Success (0) or error code

7.6 PwrSnsr_SetTimespan()

EXPORT int

PwrSnsr_SetTimespan(

SessionID	Vi,
float	Timespan

)

Set the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timespan	Horizontal time span of trace ins pulse mode.

Returns

Success (0) or error code

Marker Funtions

8.1 Commands Summary

PwrSnsr_FetchArrayMarkerPower()

PwrSnsr_FetchIntervalAvg()

PwrSnsr_FetchIntervalAvg()

PwrSnsr_FetchIntervalFilteredMin()

PwrSnsr_FetchIntervalMax()

PwrSnsr_FetchIntervalMaxAvg()

PwrSnsr_FetchIntervalMin()

PwrSnsr_FetchIntervalMinAvg()

PwrSnsr_FetchIntervalPkToAvg()

PwrSnsr_FetchMarkerAverage()

PwrSnsr_FetchMarkerDelta()

PwrSnsr_FetchMarkerMax()

PwrSnsr_FetchMarkerMin()

PwrSnsr_FetchMarkerRatio()

PwrSnsr_FetchMarkerRDelta()

PwrSnsr_FetchMarkerRRatio()

PwrSnsr_GetMarkerPixelPosition()

PwrSnsr_GetMarkerTimePosition()

PwrSnsr_ReadArrayMarkerPower()

PwrSnsr_ReadIntervalAvg()

PwrSnsr_ReadIntervalFilteredMax()

PwrSnsr_ReadIntervalFilteredMin()

PwrSnsr_ReadIntervalMax()

PwrSnsr_ReadIntervalMaxAvg()

PwrSnsr_ReadIntervalMin()

PwrSnsr_ReadIntervalMinAvg()

PwrSnsr_ReadIntervalPkToAvg()

PwrSnsr_ReadMarkerAverage()

PwrSnsr_ReadMarkerDelta()

PwrSnsr_ReadMarkerMax()

PwrSnsr_ReadMarkerMin()

PwrSnsr_ReadMarkerRatio()

PwrSnsr_ReadMarkerRDelta()

PwrSnsr_REadMarkerRRatio()

PwrSnsr_SetMarkerPixelPosition()

PwrSnsr_SetMarkerTimePosition()

8.2 PwrSnsr_FetchArrayMarkerPower()

EXPORT int

PwrSnsr_FetchArrayMarkerPower(

SessionID	Vi,
const char*	Channel,
float*	AvgPower,
PwrSnsrCondCodeEnum	AvgPowerCondCode,
float*	MaxPower,
PwrSnsrCondCodeEnum	MaxPowerCondCode,
float*	MinPower,
PwrSnsrCondCodeEnum	MinPowerCondCode,
float*	PkToAvgRatio,
PwrSnsrCondCodeEnum	PkToAvgRatioCondCode,
float*	Marker1Power,
PwrSnsrCondCodeEnum	Marker1PowerCondCode,
float*	Marker2Power,
PwrSnsrCondCodeEnum	Marker2PowerCondCode,
float*	MarkerRatio,
PwrSnsrCondCodeEnum	MarkerRatioCondCode

)

Returns an array of the current marker measurements for the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
AvgPower,	Average power between the markers.
AvgPowerCondCode,	Condition code.
MaxPower,	Maximum power between themarkers.
MaxPowerCondCode,	Condition code.
MinPower,	Minimum power between themarkers.
MinPowerCondCode,	Condition code.

PkToAvgRatio,	The ratio of peak to average power between the markers.
PkToAvgRatioCondCode,	Condition code.
Marker1Power,	The power at Marker 1.
Marker1PowerCondCode,	Condition code.
Marker2Power,	The power at Marker 2.
Marker2PowerCondCode,	Condition code.
MarkerRatio,	Ratio of power at Marker 1 and power at Marker 2.
MarkerRatioCondCode	Condition code.

Returns

Success (0) or error code

8.3 PwrSnsr_FetchIntervalAvg()

EXPORT int

PwrSnsr_FetchIntervalAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondConde,
float*	Val

)

Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Conditioncode for tyhe measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.4 PwrSnsr_FetchIntervalFilteredMax()

EXPORT int

PwrSnsr_FetchIntervalFilteredMax(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.5 PwrSnsr_FetchIntervalFilteredMin()

EXPORT int

PwrSnsr_FetchIntervalFilteredMin(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode
float	Val

)

Return the minmum power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.6 PwrSnsr_FetchIntervalMax()

EXPORT int

PwrSnsr_FetchIntervalMax(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode,	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.7 PwrSnsr_FetchIntervalMaxAvg()

EXPORT int

PwrSnsr_FetchIntervalMaxAvg()

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float	Val

)

Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters

	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Vi	Channel number. For single instruments, set this to "CH1".
	Condition code for the measurement.
Channel	Measurement value.

Returns

Success (0) or error code

8.8 PwrSnsr_FetchIntervalMin()

EXPORT int

PwrSnsr_FetchIntervalMin(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ConCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.9 PwrSnsr_FetchIntervalMinAvg()

EXPORT int

FetchIntervalMinAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.10 PwrSnsr_FetchIntervalPkToAvg()

EXPORT int

PwrSnsr_FetchIntervalPkToAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.11 PwrSnsr_FetchMarkerAverage()

EXPORT int

PwrSnsr_FetchMarkerAverage(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the average power or voltage of the specified marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Marker number.
Val	Measurement value.

Returns

Success (0) or error code

8.12 PwrSnsr_FetchMarkerDelta()

EXPORT int

PwrSnsr_FetchMarkerDelta(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the difference between MK1 and MK2. The units will be the same as marker units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.13 PwrSnsr_FetchMarkerMax()

EXPORT int

PwrSnsr_FetchMarkerMax(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returns the maximum power or voltage of the specified marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Marker number
Val	Measurement value.

Returns

Success (0) or error code

8.14 PwrSnsr_FetchMarkerMin()

EXPORT int

PwrSnsr_FetchMarkerMin(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	IsValid,
float*	Val

)

Returna the minimum power or voltage of the specified marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Marker number.
Val	Measurement value.

Returns

Success (0) or error code

8.15 PwrSnsr_FetchMarkerRatio()

EXPORT int

PwrSnsr_FetchMarkerRatio(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Units for selected channel.Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.16 PwrSnsr_FetchMarkerRDelta()

EXPORT int

PwrSnsr_FetchMarkerRDelta(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode
float*	Val

)

Returns the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.17 PwrSnsr_FetchMarkerRRatio()

EXPORT int

PwrSnsr_FetchMarkerRRatio(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	ConCode,
float*	Val

)

Returns the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.18 PwrSnsr_GetMarkerPixelPosition()

EXPORT int

PwrSnsr_GetMarkerPixelPosition(

SessionID	Vi,
int	MarkerNumber,
int*	PixelPosition

)

Get the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MarkerNumber	Marker number.
PixelPosition	Picel position value.

Returns

Success (0) or error code

8.19 PwrSnsr_GetMarkerTimePosition()

EXPORT int

PwrSnsr_GetMarkerTimePosition(

SessionID	Vi,
int	MarkerNumber,
float*	TimePosition

)

Get the time (x-axis-position) of the selected marker relative to the trigger.

Note:

Time markers must be positioned within the time limits of the trace window in the graph display. If a time outside of the display limits is entered, the marker will be placed at the first or last time position as appropriate.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MarkerNumber	Marker number.
TimePosition	Time position value.

Returns

Success (0) or error code

8.20 PwrSnsr_ReadArrayMarkerPower()

EXPORT int

PwrSnsr_ReadArrayMarkerPower(

SessionID	Vi,
const char*	Channel,
float*	AvgPower,
PwrSnsrCondCodeEnum*	AvgPowerCondCode,
float*	MaxPower,
PwrSnsrCondCodeEnum*	MaxPowerCondCode,
float*	MinPower,
PwrSnsrCondCodeEnum*	MinPowerConCode,
float*	PkToAvgRatio,
PwrSnsrCondCodeEnum*	PkToAvgRatioCondCode,
float*	Marker1Power,
PwrSnsrCondCodeEnum*	Marker1PowerCondCode,
float*	Marker2Power,
PwrSnsrCondCodeEnum*	Marker2PowerCondCode,
float*	MarkerRatio,
PwrSnsrCondCodeEnum*	MarkerRatioCondCode

)

Returns an array of the current marker measurements for the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
AvgPower,	Average power between the markers.
AvgPowerCondCode,	Condition code.
MaxPower,	Maximum power between themarkers.
MaxPowerCondCode,	Condition code.
MinPower,	Minimum power between themarkers.
MinPowerConCode,	Condition code.
PkToAvgRatio,	The ratio of peak to average power between the markers.
PkToAvgRatioCondCode,	Condition code.
Marker1Power,	The power at Marker 1.

Marker1PowerCondCode,	Condition code.
Marker2Power,	The power at Marker 2.
Marker2PowerCondCode,	Condition code.
MarkerRatio,	Ratio of power at Marker 1 and power at Marker 2.
MarkerRatioCondCode	Condition code.

Returns

Success (0) or error code

8.21 PwrSnsr_ReadIntervalAvg()

EXPORT int

PwrSnsr_ReadIntervalAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.22 PwrSnsr_ReadIntervalFilteredMax()

EXPORT int

PwrSnsr_ReadIntervalFilteredMax(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.23 PwrSnsr_ReadIntervalFilteredMin()

EXPORT int

PwrSnsr_ReadIntervalFilteredMin(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the minmum power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement values.

Returns

Success (0) or error code

8.24 PwrSnsr_ReadIntervalMax()

EXPORT int

PwrSnsr_ReadIntervalMax(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.25 PwrSnsr_ReadIntervalMaxAvg()

EXPORT int

PwrSnsr_ReadIntervalMaxAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return maximum of the average power trace between MK1 and MK2.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Conditional code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.26 PwrSnsr_ReadIntervalMin()

EXPORT int

PwrSnsr_ReadIntervalMin(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode
float*	Val

)

Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.27 PwrSnsr_ReadIntervalMinAvg()

EXPORT int

PwrSnsr_ReadIntervalMinAvg()

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return minimum of the average power trace between MK1 and MK2.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.28 PwrSnsr_ReadIntervalPkToAvg()

EXPORT int

PwrSnsr_ReadIntervalPkToAvg(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.29 PwrSnsr_ReadMarkerAverage()

EXPORT int

PwrSnsr_ReadMarkerAverage(

SessionID	Vi,
const char*	Marker,
int	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Forthe specified marker, return the average power or voltage at the marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Marker	Marker number.
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.30 PwrSnsr_ReadMarkerDelta()

EXPORT int

PwrSnsr_ReadMarkerDelta(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the difference between MK1 and MK2. The units will be the same as marker units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.31 PwrSnsr_ReadMarkerMax()

EXPORT int

PwrSnsr_ReadMarkerMax(

SessionID	Vi,
const char*	Channel,
int	Marker,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Forthe specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Marker	Marker number.
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.32 PwrSnsr_ReadMarkerMin()

EXPORT int

PwrSnsr_ReadMarkerMin(

SessionID	Vi,
const char*	Channel,
int	Marker,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Return the minimum power or voltage of the specified marker. The units are the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Marker	Marker number.
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.33 PwrSnsr_ReadMarkerRatio()

EXPORT int

PwrSnsr_ReadMarkerRatio(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCodde,
float*	Val

)

Returns the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.34 PwrSnsr_ReadMarkerRDelta()

EXPORT int

PwrSnsr_ReadMarkerRDelta(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the difference between MK2 and MK1. The units will be the same as marker units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measuremnet.
Val	Measurement value.

Returns

Success (0) or error code

8.35 PwrSnsr_ReadMarkerRRatio()

EXPORT int

PwrSnsr_ReadMarkerRRatio(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Measurement value.

Returns

Success (0) or error code

8.36 PwrSnsr_SetMarkerPixelPosition()

EXPORT int

PwrSnsr_SetMarkerPixelPosition(

SessionID	Vi,
int	MarkerNumber,
int	PixelPosition

)

Set the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Channel number. For single instruments, set this to "CH1".
MarkerNumber	Marker number.
PixelPosition	Picel position.

Returns

Success (0) or error code

8.37 PwrSnsr_SetMarkerTimePosition()

EXPORT int

PwrSnsr_SetMarkerTimePosition(

SessionID	Vi,
int	MarkerNumber,
float	TimePosition

)

Set the time (x-axis-position) of the selected marker relative to the trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Channel number. For single instruments, set this to "CH1".
MarkerNumber	Marker number.
TimePosition	Time position.

Returns

Success (0) or error code

Display Functions

9.1 Commands Summary

PwrSnsr_GetVerticalCenter()

PwrSnsr_GetVerticalScale()

PwrSnsr_SetVerticalCenter()

PwrSnsr_SetVerticalScale()

9.2 PwrSnsr_GetVerticalCenter()

EXPORT int

PwrSnsr_GetVerticalCenter(

SessionID	Vi,
const char*	Channel,
float*	VerticalCenter

)

Gets vertical center based on current units: arg = (range varies by units)

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VerticalCenter	Vertical center in units.

Returns

Success (0) or error code

9.3 PwrSnsr_GetVerticalScale()

EXPORT int

PwrSnsr_GetVerticalScale(

SessionID	Vi,
const char*	Channel,
float*	VerticalScale

)

Gets vertical scale based on current units: arg = (range varies by units)

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VerticalScale	Vertical scale in units.

Returns

Success (0) or error code

9.4 PwrSnsr_SetVerticalCenter()

EXPORT int

PwrSnsr_SetVerticalCenter()

SessionID	Vi,
const char*	Channel,
float*	VertivalCenter

)

Sets vertical center based on current units: arg = (range varies by units)

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VerticalCenter	Vertival center in units.

Returns

Success (0) or error code

9.5 PwrSnsr_SetVerticalScale()

EXPORT int

PwrSnsr_SetVerticalScale(

SessionID	Vi,
const char*	Channel,
float*	VertivalCenter

)

Gets vertical center based on current units: arg = (range varies by units)

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VertivalScale	Vertical scale in units.

Returns

Success (0) or error code

Statistical Mode

10.1 Commands Summary

PwrSnsr_FetchCCDFPercent()

PwrSnsr_FetchCCDFPower()

PwrSnsr_FetchCCDFTrace()

PwrSnsr_FetchCursorPercent()

PwrSnsr_FetchCursorPower()

PwrSnsr_FetchStatMeasurementArray()

PwrSnsr_GetAcqStatusArray()

PwrSnsr_GetCapture()

PwrSnsr_(GetCCDFTraceCount)

PwrSnsr_GetDiagStatusArray()

PwrSnsr_GetGating()

PwrSnsr_GetHorizontalOffset()

PwrSnsr_GetHorizontalScale()

PwrSnsr_GetPercentPosition()

PwrSnsr_GetPowerPosition()

PwrSnsr_GetTermAction()

PwrSnsr_GetTermCount()

PwrSnsr_GetTermTime()

PwrSnsr_SetCapture()

PwrSnsr_SetCCDFTraceCount()

PwrSnsr_SetGating()

PwrSnsr_SetHorizontalOffset()

PwrSnsr_SetHorizontalScale()

PwrSnsr_SetPercentPosition()

PwrSnsr_SetPowerPosition()

PwrSnsr_SetTermAction()

PwrSnsr_SetTermCount()

PwrSnsr_SetTermTime()

PwrSnsr_StatModeReset()

10.2 PwrSnsr_FetchCCDFPercent()

EXPORT int

PwrSnsr_FetchCCDFPercent(

SessionID	Vi,
const char*	Channel,
double	Power,
PwrSnsrCondCodeEnum*	CondCode,
double*	Val

)

Returns relative power (in dB) for a given percent on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Power	Relative power in dB.
CondCode,	Condition code for the measurement.
Val	Percent measurement at power.

Returns

Success (0) or error code

10.3 PwrSnsr_FetchCCDFPower()

EXPORT int

PwrSnsr_FetchCCDFPower(

SessionID	Vi,
const char*	Channel,
double	Percent,
PwrSnsrCondCodeEnum*	CondCode,
double*	Val

)

Return relative power (in dB) for a given percent on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Percent	Statistical percent to retrieve power from.
CondCode	Condition code for the measurement.
Val	Relative power at percent.

Returns

Success (0) or error code

10.4 PwrSnsr_FetchCCDFTrace()

EXPORT int

PwrSnsr_FetchCCDFTrace(

SessionID	Vi,
const char*	Channel,
int	TraceBufferSize,
float	Trace[],
int*	TraceActualSize

)

Returns the points in the CCDF trace.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Channel number. For single instruments, set this to "CH1".
Channel	Trace buffer size.
TraceBufferSize	Trace number.
TraceActualSize	Trace actual size.

Returns

Success (0) or error code

10.5 PwrSnsr_FetchCursorPercent()

EXPORT int

PwrSnsr_FetchCursorPercent(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
double*	Val

)

Returns the percent CCDF at the cursor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Percent CCDF value.

Returns

Success (0) or error code

10.6 PwrSnsr_FetchCursorPower()

EXPORT int

PwrSnsr_FetchCursorPower(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
double*	Val

)

Returns the power CCDF in dB at the cursor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Channel number. For single instruments, set this to "CH1".
Channel	Condition code for the measurement.
VertivalScale	CCDF power value.

Returns

Success (0) or error code

10.7 PwrSnsr_FetchStatMeasurementArray()

EXPORT int

PwrSnsr_FetchStatMeasurementArray(

SessionID	Vi,
const char*	Channel,
double	Pavg,
PwrSnsrCondCodeEnum*	PavgCond,
double*	Ppeak,
PwrSnsrCondCodeEnum*	PpeakCond,
double*	Pmin,
PwrSnsrCondCodeEnum*	PminCond,
double*	PkToAvgRatio,
PwrSnsrCondCodeEnum*	PkToAvgRatioCond,
double*	CursorPwr,
PwrSnsrCondCodeEnum*	CursorPwrCond,
double*	CursorPct,
PwrSnsrCondCodeEnum*	CursorPctCond,
double*	SampleCount,
PwrSnsrCondCodeEnum*	SampleCountCond,
double*	SecondsRun,
PwrSnsrCondCodeEnum*	SecondsRunCond

)

Returns an array of the current automatic statistical measurements performed on a sample population.

Note:

Measurements performed are: long term average, peak and minimum amplitude, peak-toaverage ratio, amplitude at the CCDF percent cursor, statistical percent at the CCDF power cursor, and the sample population size in samples.

The peak-to-average ratio is returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Pavg	Long term average power in channel units.
PavgCond	Condition code for the measurement.
Ppeak	Peak power in channel units.
PpeakCond	Condition code.
Pmin	Minimum power in channel units.
PminCond	Condition code.
PkToAvgRatio	Peak-to-average power in percent or dB.
PkToAvgRatioCond	Condition code.
CursorPwr	Power at the cursor in channel units.
CursorPwrCond	Condition code.
CursorPct	Statistical percent at the cursor.
CursorPctCond	Condition code.
SampleCount	Population size in samples.
SampleCountCond	Condition code.
SecondsRun	Number of seconds the measurement has run.
SecondsRunCond	Condition code.

Returns

Success (0) or error code

10.8 PwrSnsr_GetAcqStatusArray()

EXPORT int

PwrSnsr_GetAcqStatusArray(

SessionID	Vi,
const char*	Channel,
int*	SweepLength,
double*	SampleRate,
double*	SweepRate,
double*	SweepTime,
double*	StartTime,
int*	StatusWord

)

Returns data about the status of the acquisition system.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
SweepLength	Returns the number of points in the trace.
SampleRate	Returns the sample rate.
SweepRate	Returns the number of triggered sweeps per second.
SweepTime	Returns the sweep time for the trace.
StartTime	Returns the start time relative to the trigger.
StatusWord	Internal use - acquisition system status word.

Returns

Success (0) or error code

10.9 PwrSnsr_GetCapture()

EXPORT int

PwrSnsr_GetCapture(

SessionID	Vi,
const char*	Channel,
int*	Capture

)

Get whether statistical capture is enabled.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Capture	Capture

Returns

Success (0) or error code

10.10 PwrSnsr_GetCCDFTraceCount()

EXPORT int

PwrSnsr_GetCCDFTraceCount(

SessionID	Vi,
const char*	Channel,
int*	TraceCount

)

Get the number of points in the CCDF trace plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TraceCount	Trace count.

Returns

Success (0) or error code

10.11 PwrSnsr_GetDiagStatusArray()

EXPORT int

PwrSnsr_GetDiagStatusArray(

SessionID	Vi,
const char*	Channel,
float*	DetectorTemp,
float*	CpuTemp,
float*	MioVoltage,
float*	VccInt10,
float*	VccAux18,
float*	Vcc50,
float*	Vcc25,
float*	Vcc33

)

Returns diagnostic data.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
DetectorTemp	Temperature in degrees C at the RF detector.
CpuTemp	Temperature of the CPU in degrees C.
MioVoltage	Voltage at the Multi I/O port.
VccInt10	Vcc 10 voltage.
VccAux18	Vcc Aux 18 voltage.
Vcc50	Vcc 50 voltage.
Vcc25	Vcc 25 voltage.
Vcc33	Vcc 33 voltage.

Returns

Success (0) or error code

10.12 PwrSnsr_GetGating()

EXPORT int

PwrSnsr_GetGating(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
	Gating

)

Get whether statistical capture is enabled.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Gating	Condition code for the measurement.
	Condition code for gating.

Returns

Success (0) or error code

10.13 PwrSnsr_GetHorizontalOffset()

EXPORT int

PwrSnsr_GetHorizontalOffset(

SessionID	Vi,
const char*	Channel,
double*	HorizontalOffset

)

Get the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
HorizontalOffset	Horizontal offset.

Returns

Success (0) or error code

10.14 PwrSnsr_GetHorizontalScale()

EXPORT int

PwrSnsr_GetHorizontalScale(

SessionID	Vi,
const char*	Channel,
double*	HorizontalScale

)

Get the statistical mode horizontal scale in dB/Div.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
HorizontalScale	Horizontal scale value.

Returns

Success (0) or error code

10.15 PwrSnsr_GetPercentPosition()

EXPORT int

PwrSnsr_GetPercentPosition(

SessionID	Vi,
const char*	Channel,
double*	PercentPosition

)

Get the cursor percent on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PercentPosition	Percent position value.

Returns

Success (0) or error code

10.16 PwrSnsr_GetPowerPosition()

EXPORT int

PwrSnsr_GetPowerPosition(

SessionID	Vi,
const char*	Channel,
double*	PowerPosition

)

Get the cursor power in dB on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VertivalScale	Power position value.

Returns

Success (0) or error code

10.17 PwrSnsr_GetTermAction()

EXPORT int

PwrSnsr_GetTermAction(

SessionID	Vi,
const char*	Channel,
PwrSnsrTermActionEnum *	TermAction

)

Get the termination action for statistical capturing.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
VertivalScale	Termination action for the statistical capturing.

Returns

Success (0) or error code

10.18 PwrSnsr_GetTermCount()

EXPORT int

PwrSnsr_GetTermCount(

SessionID	Vi,
const char*	Channel,
double*	TermCount

)

Get the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TermCount	Term count value.

Returns

Success (0) or error code

10.19 PwrSnsr_GetTermTime()

EXPORT int

PwrSnsr_GetTermTime(

SessionID	Vi,
const char*	Channel,
int*	TermTime

)

Get the termination time in seconds for statistical capturing. After the time has elapsed, the action determined by TermAction is taken.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TermTime	Term time value.

Returns

Success (0) or error code

10.20 PwrSnsr_SetCapture()

EXPORT int

PwrSnsr_SetCapture(

SessionID	Vi,
const char*	Channel,
int	Capture

)

Set whether statistical capture is enabled.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Capture	Capture value.

Returns

Success (0) or error code

10.21 PwrSnsr_SetCCDFTraceCount()

EXPORT int

PwrSnsr_SetCCDFTraceCount(

SessionID	Vi,
const char*	Channel,
int	TraceCount

)

Set the number of points (1 - 16384) in the CCDF trace plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TraceCount	Number of points in the CCDF trace plot.

Returns

Success (0) or error code

10.22 PwrSnsr_SetGating()

EXPORT int

PwrSnsr_SetGating(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	Gating

)

Set whether the statical capture is gated by markers or free-running.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Gating	Statical capture mode.

Returns

Success (0) or error code

10.23 PwrSnsr_SetHorizontalOffset()

EXPORT int

PwrSnsr_SetHorizontalOffset(

SessionID	Vi,
const char*	Channel,
double	HorizontalOffset

)

Set the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
HorizontalOffset	Statistical mode horizontal scale offset in dB.

Returns

Success (0) or error code

10.24 PwrSnsr_SetHorizontalScale()

EXPORT int

PwrSnsr_SetHorizontalScale(

SessionID	Vi,
const char*	Channel,
double	HorizontalScale

)

Set the statistical mode horizontal scale in dB/Div.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
HorizontalScale	Statistical mode horizontal scale in dB/Div.

Returns

Success (0) or error code

10.25 PwrSnsr_SetPercentPosition()

EXPORT int

PwrSnsr_SetPercentPosition(

SessionID	Vi,
const char*	Channel,
double	PercentPosition

)

Set the cursor percent on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PercentPosition	Cursor percent value.

Returns

Success (0) or error code

10.26 PwrSnsr_SetPowerPosition()

EXPORT int

PwrSnsr_SetPowerPosition(

SessionID	Vi,
const char*	Channel,
double	PowerPosition

)

Set the cursor power in dB on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PowerPosition	Cursor power value in dB.

Returns

Success (0) or error code

10.27 PwrSnsr_SetTermAction()

EXPORT int

PwrSnsr_SetTermAction(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	TermAction

)

Set the termination action for statistical capturing.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TermAction	Termination action for statistical capturing.

Returns

Success (0) or error code

10.28 PwrSnsr_SetTermCount()

EXPORT int

PwrSnsr_SetTermCount(

SessionID	Vi,
const char*	Channel,
double	TermCount

)

Set the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TermCount	Termination count value for statistical capturing.

Returns

Success (0) or error code

10.29 PwrSnsr_SetTermTime()

EXPORT int

PwrSnsr_SetTermTime(

SessionID	Vi,
const char*	Channel,
int	TermTime

)

Set the termination time in seconds (1 - 3600) for statistical capturing. After the time has elapsed, the action determined by TermAction is taken.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TermTime	Termination time value in seconds.

Returns

Success (0) or error code

10.30 PwrSnsr_StatModeReset()

EXPORT int

PwrSnsr_StatModeReset(

SessionID Vi,

const char* Channel

)

Resets statistical capturing mode by clearing the buffers and restarting the aquisition timer.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code

Sensor Info

11.1 Commands Summary

PwrSnsr_GetAttenuation()

PwrSnsr_GetFactoryCalDate()

PwrSnsr_GetFpgaVersion()

PwrSnsr_GetImpedance()

PwrSnsr_GetManufactureDate()

PwrSnsr_GetMaxFreqHighBandwidth()

PwrSnsr_GetMaxFreqLowBandwidth()

PwrSnsr_GetMinFreqHighBandwidth()

PwrSnsr_GetMinFreqLowBandwidrht()

PwrSnsr_GetMinimumTrig()

PwrSnsr_GetPeakPowerMax()

PwrSnsr_GetPeakPowerMin()

11.2 PwrSnsr_GetAttenuation()

EXPORT int

PwrSnsr_GetAttenuation(

SessionID	Vi,
const char*	Channel,
float	Attenuation

)

Returns the attenuation in dB of the sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Attenuation	Attenuation value.

Returns

Success (0) or error code

11.3 PwrSnsr_GetFactoryCalDate()

EXPORT int

PwrSnsr_GetFactoryCalDate(

SessionID	Vi,
const char*	Channel,
int	FactoryCalDateBufferSize,
char	FactroyCalDate[]

)

Returns the date (YYYYmmDD) the last time the sensor was calibrated at the factory.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
FactroyCalDateBufferSize	Size of FactoryCalDate in bytes.
FactoryCalDate[]	Factory cal date.

Returns

Success (0) or error code

11.4 PwrSnsr_GetFpgaVersion()

EXPORT int

PwrSnsr_GetFpgaVersion(

SessionID	Vi,
const char*	Channel,
Int	ValBufferSize,
Char	Val[]

)

Get the sensor FPGA version.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size pf Val in bytes.
Val[]	Buffer for storing the version.

Returns

Success (0) or error code

11.5 PwrSnsr_GetImpedance()

EXPORT int

PwrSnsr_GetImpedance(

SessionID	Vi,
const char*	Channel,
float	Impedance

)

Returns the input impedance of the sensor.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Impedance	Imput impedance value.

Returns

Success (0) or error code

11.6 PwrSnsr_GetManufactureDate()

EXPORT int

PwrSnsr_GetManufactureDate(

SessionID	Vi,
const char*	Channel,
int	ManufacureDateBufferSize,
char	ManufactureDate[]

)

Returns date the sensor was manufactured in the following format YYYYmmDD.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ManufacureDateBufferSize	Size of the ManufactureDate in bytes.
ManufactureDate[]	Return value, manufacture date.

Returns

Success (0) or error code

11.7 PwrSnsr_GetMaxFreqHighBandwidth()

EXPORT int

PwrSnsr_GetMaxFreqHighBandwidth(

SessionID	Vi,
const char*	Channel,
float	MaxFreqHighBandwidth

)

Returns maximum frequency carrier the sensor can measure in high bandwidth.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MaxFreqHighBandwith	Maximum frequency value in high bandwith.

Returns

Success (0) or error code

11.8 PwrSnsr_GetMaxFreqLowBandwidth()

EXPORT int

PwrSnsr_GetMaxFreqLowBandwidth(

SessionID	Vi,
const char*	Channel,
float	MaxFreqLowBandwidth

)

Returns maximum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MaxFreqLowBandwidth	Macimum frequency value in low bandwith.

Returns

Success (0) or error code

11.9 PwrSnsr_GetMinFreqHighBandwidth()

EXPORT int

PwrSnsr_GetMinFreqHighBandwidth(

SessionID	Vi,
const char*	Channel,
float	MinFreqHighBandwidth

)

Returns minimum frequency of RF the sensor can measure in high bandwidth.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinFreqHighBandwidth	Minimum frequency value in high bandwidth.

Returns

Success (0) or error code

11.10 PwrSnsr_GetMinFreqLowBandwidth()

EXPORT int

PwrSnsr_GetMinFreqLowBandwidth(

SessionID	Vi,
const char*	Channel,
float	MinFreqLowBandwidth

)

Returns minimum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinFreqLowBandwidth	Minimum frequency value in low bandwidth.

Returns

Success (0) or error code

11.11 PwrSnsr_GetMinimumTrig()

EXPORT int

PwrSnsr_GetMinimumTrig(

SessionID	Vi,
const char*	Channel,
float	Minimumtrig

)

Returns minimum internal trigger level in dBm.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinimumTrig	Minimum internal trigger level in dBm.

Returns

Success (0) or error code

11.12 PwrSnsr_GetPeakPowerMax()

EXPORT int

PwrSnsr_GetPeakPowerMax(

SessionID	Vi,
const char*	Channel,
float	PeakPowerMax

)

Returns the maximum power level the sensor can measure.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PeakPowerMax	Maximum power level value.

Returns

Success (0) or error code

11.13 PwrSnsr_GetPeakPowerMin()

EXPORT int

PwrSnsr_GetPeakPowerMin(

SessionID	Vi,
const char*	Channel,
float	PeakPowerMin

)

Minimum power level the sensor can measure.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PeakPowerMin	Minimum power value.

Returns

Success (0) or error code

User Calibration

12.1 Commands Summary

PwrSnsr_ClearUserCal()

PwrSnsr_GetExternalSkew()

PwrSnsr_GetInternalSkew()

PwrSnsr_GetSlaveSkew()

PwrSnsr_SaveUserCal()

PwrSnsr_SetExternalSkew()

PwrSnsr_SetInternalSkew()

PwrSnsr_SetSlaveSkew()

PwrSnsr_Zero()

PwrSnsr_ZeroQuery()

12.2 PwrSnsr_ClearUserCal()

EXPORT int

PwrSnsr_ClearUserCal(

SessionID	Vi,
const char*	Channel,

)

Resets the value of fixed cal, zero, and skew to factory defaults.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code

12.3 PwrSnsr_GetExternalSkew()

EXPORT int

PwrSnsr_GetExternalSkew(

SessionID	Vi,
const char*	Channel,
float	Externa

)

Returns the skew in seconds for the external trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
External	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.4 PwrSnsr_GetInternalSkew()

EXPORT int

PwrSnsr_GetInternalSkew(

SessionID	Vi,
const char*	Channel,
float	InteranlSkew

)

Gets the skew in seconds for the internal trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
InternalSkew	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.5 PwrSnsr_GetSlaveSkew()

EXPORT int

PwrSnsr_GetSlaveSkew(

SessionID	Vi,
const char*	Channel,
float	SlaveSkew

)

Gets the skew in seconds for the slave trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
SlaveSkew	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.6 PwrSnsr_SaveUserCal()

EXPORT int

PwrSnsr_SaveUserCal(

SessionID	Vi,
const char*	Channel,

)

Instructs power meter to save the value of fixed cal, zero, and skew values.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code

12.7 PwrSnsr_SetExternalSkew(

EXPORT int

PwrSnsr_SetExternalSkew

SessionID	Vi,
const char*	Channel,
float	External

)

Sets the skew in seconds for the external trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
External	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.8 PwrSnsr_SetInternalSkew()

EXPORT int

PwrSnsr_SetInternalSkew(

SessionID	Vi,
const char*	Channel,
float	InternalSkew

)

Sets the skew in seconds for the internal trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
InternalSkew	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.9 PwrSnsr_SetSlaveSkew()

EXPORT int

PwrSnsr_SetSlaveSkew(

SessionID	Vi,
const char*	Channel,
float	SlaveSkew

)

Sets the skew in seconds for the slave trigger.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
SlaveSkew	Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code

12.10 PwrSnsr_Zero()

EXPORT int

PwrSnsr_Zero(

SessionID	Vi,
const char*	Channel,

)

Performs a zero offset null adjustment.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code

12.11 PwrSnsr_ZeroQuery()

EXPORT int

PwrSnsr_ZeroQuery(

SessionID	Vi,
const char*	Channel,
int*	Val

)

Performs a zero offset null adjustment and returns true if successful.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Val	Boolean value for operation success or failure.

Returns

Success (0) or error code

Trace Function

13.1 Commands Summary

PwrSnsr_FetchDistal()

PwrSnsr_FetchMesial()

PwrSnsr_FetchProximal()

PwrSnsr_GetChanTraceCount()

PwrSnsr_GetSweepTime()

PwrSnsr_GetTimePerPoint()

PwrSnsr_GetTraceStartTime()

13.2 PwrSnsr_FetchDistal()

EXPORT int

PwrSnsr_FetchDistal(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the actual detected power of the distal level in the current channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Detected power of lthe distal level in the current channel units.

Returns

Success (0) or error code

13.3 PwrSnsr_FetchMesial()

EXPORT int

PwrSnsr_FetchMesial(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	Val

)

Returns the actual detected power of the mesial level in the current channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Detected power of lthe mesial level in the current channel.

Returns

Success (0) or error code

13.4 PwrSnsr_FetchProximal()

EXPORT int

PwrSnsr_FetchProximal(

SessionID	Vi,
const char*	Channel,
PwrSnsrCondCodeEnum*	CondCode,
float*	

)

Returns the actual detected power of the proximal level in the current channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.

Returns

Success (0) or error code

13.5 PwrSnsr_GetChanTraceCount()

EXPORT int

PwrSnsr_GetChanTraceCount(

SessionID	Vi,
const char*	Channel,
int*	TraceCount

)

Get the number of points in the CCDF trace plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TraceCount	The number of points in the CCDF trace plot

Returns

Success (0) or error code

13.6 PwrSnsr_GetSweepTime()

EXPORT int

PwrSnsr_GetSweepTime(

SessionID	Vi,
const char*	Channel,
float*	SweepTime

)

Get sweep time for the trace in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
SweepTime	Sweep time for the trace in seconds.

Returns

Success (0) or error code

13.7 PwrSnsr_GetTimePerPoint()

EXPORT int

PwrSnsr_GetTimePerPoint(

SessionID	Vi,
const char*	Channel,
float*	TimePerPoint

)

Get time spacing for each waveform point in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TimePerPoint	Tiem spacing for each waveform point in seconds.

Returns

Success (0) or error code

13.8 PwrSnsr_GetTraceStartTime()

EXPORT int

PwrSnsr_GetTraceStartTime(

SessionID	Vi,
const char*	Channel,
float*	TraceStartTime

)

Get time offset (start time) of the trace in seconds. May be negative, indicating pre-trigger information.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TraceStartTim	Time offset of the trace in seconds. May be negative, indicating pretrigger information

Returns

Success (0) or error code

Multiple Pulse

14.1 Commands Summary

PwrSnsr_FetchAllMultiPulse()

- Pulse Info

14.2 PwrSnsr_FetchAllMultiPulse()

EXPORT int

PwrSnsr_FetchAllMultiPulse(

SessionID	Vi,
const char*	Channel,
int	PulseInfosSize,
PulseInfo	PulseInfos[],
int*	PulseInfosActualSize

)

Return all previously acquired multiple pulse measurements. The elements in the PulseInfos array correspond to pulses on the current trace from left to right (ascending time order).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PulseInfosSize	PulseInfosSize Number of elements in PulseInfos array.
PulseInfos[]	Array to fill with multi pulse information.
PulseInfosActualSize	Actual number of valid elements in PulseInfos array.

Returns

Success (0) or error code

14.2.1 Pulse Info

FallDistal

float FallDistal Position in time for the distal crossing on the falling edge of the pulse.

FallProximal

float FallProximal Position in time for the proximal crossing on the falling edge of the pulse.

FallTime

float FallTime Fall time of the pulse.

Min

float Min Minimum instanaeous power measurement.

Peak

float Peak Peak (max instanaeous) power measurement.

Position

float Position Time position corresponding to the mesial crossing of the rising edge for the pulse.

PulseAvg

float PulseAvg Average power measurement for the pulse.

RiseDistal

float RiseDistal Position in time for the distal crossing on the rising edge of the pulse.

RiseProximal

float RiseProximal Position in time for the proximal crossing on the rising edge of the pulse.

RiseTime

float RiseTime Rise time of the pulse.

Width

float Width Pulse width is defined as the interval between the first and second signal crossings of the mesial line.

Memory Channels

15.1 Commands Summary

PwrSnsr_GetMemChanArchive()

PwrSnsr_LoadMemChanFromArchive()

PwrSnsr_SaveToMemoryChannel()

15.2 PwrSnsr_GetMemChanArchive()

EXPORT int

PwrSnsr_GetMemChanArchive(

SessionID	Vi,
const char*	MemChan,
int	ValBufferSize,
char	Val[]

)

Returns an XML document containing settings and readings obtained using the SaveToMemoryChannel method.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MemChan	The name of the memory channel to get the archive form.
ValBufferSize	
Val	XML document containing settings and readings.

Returns

Success (0) or error code

15.3 PwrSnsr_LoadMemChanFromArchive()

EXPORT int

PwrSnsr_LoadMemChanFromArchive(

SessionID	Vi,
const char*	MemChan,
const char*	ArchiveContent

)

Loads the named memory channel using the given archive. If the memory channel does not exist, one is created.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MemChan	Memory channel name. Must have the form MEM1...n, where n is the number of measurement channels. In single channel configurations, this parameter should always be "MEM1".
ArchiveContent	An xml document containing settings and readings obtained using the SaveToMemoryChannel method. An archive can be obtained using the GetMemChanArchive method.

Returns

Success (0) or error code

15.4 PwrSnsr_SaveToMemoryChannel()

EXPORT int

PwrSnsr_SaveToMemoryChannel(

SessionID	Vi,
const char*	MemChan,
const char*	ChannelName

)

Saves the given channel to a memory channel. If the memory channel does not exist, a new one is created.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MemChannel	Memory channel name. Must have the form MEM1...n, where n is the number of measurement channels. In single channel configurations, this parameter should always be "MEM1".
ChannelName	The channel name to copy from.

Returns

Success (0) or error code

Modulated Measurements

16.1 Command Summary

PwrSnsr_GetIsAvailable()

PwrSnsr_GetIsRunning()

PwrSnsr_GetReadingPeriod()

16.2 PwrSnsr_GetIsAvailable()

EXPORT int

PwrSnsr_GetIsAvailable(

SessionID	Vi,
const char*	MemChan,
int*	IsAvailable

)

Returns true if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsAvailable	True if modulate/CW measurement system is available. Will always return false if measurement buffer is enabled.

Returns

Success (0) or error code

16.3 PwrSnsr_GetIsRunning()

EXPORT int

PwrSnsr_GetIsRunning(

SessionID	Vi,
const char*	MemChan,
int*	IsRunning

)

Returns true if modulated/CW measuremnts are actively running.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsRunning	True if modulated/CW measurements are actively running.

Returns

Success (0) or error code

16.4 PwrSnsr_GetReadingPeriod()

EXPORT int

PwrSnsr_GetReadingPeriod(

SessionID	Vi,
const char*	MemChan,
float*	ReadingPeriod

)

Returns the period (rate) in seconds per new filtered reading.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel,	Channel number. For single instruments, set this to "CH1".
ReadingPeriod	The period(rate) in seconds per new filtered reading.

Returns

Success (0) or error code

Measurement Buffer

17.1 Measurement Buffer

PwrSnsr_AcquireMeasurements()

PwrSnsr_AdvanceReadIndex()

PwrSnsr_ClearBuffer()

PwrSnsr_ClearMeasurements()

PwrSnsr_GetBufferedAverageMeasurements()

PwrSnsr_GetBufferedMeasurementsAvailable()

PwrSnsr_GetContinuousCapture()

PwrSnsr_GetDuration()

PwrSnsr_GetDurations()

PwrSnsr_GetEndDelay()

PwrSnsr_GetEndQual()

PwrSnsr_GetGateMode()

PwrSnsr_GetMaxMeasurements()

PwrSnsr_GetMeasBuffEnabled()

PwrSnsr_GetMeasurementsAvailable()

PwrSnsr_GetMinMeasurements()

PwrSnsr_GetOverRan()

PwrSnsr_GetPeriod()

PwrSnsr_GetRdgsEnableFlag()

PwrSnsr_GetReturnCount()

PwrSnsr_GetSequenceNumbers()

PwrSnsr_GetSessionCount()

PwrSnsr_GetStartDelay()

PwrSnsr_GetStartMode()

PwrSnsr_GetStartQual()

PwrSnsr_GetStartTimes()

PwrSnsr_GetTimedOut()

PwrSnsr_GetWriteProtection()

PwrSnsr_QueryAverageMeasurements()

PwrSnsr_QueryDurations()

PwrSnsr_QueryMaxMeasurements()

PwrSnsr_QueryMinMeasurements()

PwrSnsr_QuerySequenceNumbers()

PwrSnsr_QueryStartTimes()

PwrSnsr_ResetContinuousCapture()

PwrSnsr_SetDuration()

PwrSnsr_SetEndDelay()

PwrSnsr_SetEndQual()

PwrSnsr_SetGateMode()

PwrSnsr_SetMeasBuffEnabled()

PwrSnsr_SetPeriod()

PwrSnsr_SetRdgsEnableFlag()

PwrSnsr_SetReturnCount()

PwrSnsr_SetSessionCount()

PwrSnsr_SetSessionTimeout()

PwrSnsr_SetStartDelay()

PwrSnsr_SetStartMode()

PwrSnsr_SetStartQual()

PwrSnsr_SetWriteProtection()

PwrSnsr_StartAcquisition()

PwrSnsr_StopAcquisition()

17.2 PwrSnsr_AcquireMeasurements()

EXPORT int

PwrSnsr_AcquireMeasurements()

SessionID	Vi,
double	Timeout,
int	Count,
PwrSnsrMeasBuffStopReasonEnum *	StopReason,
int*	Val

)

Initiates new acquisition from the measurement buffer system (if acquisition is in the stopped state). Blocks until the number of measurements for each enabled channel is equal to count, or a time out has occurred.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timeout	Maximum time in seconds to continue acquiring samples. Negative values will be treated as infinite.
Count	Number of samples to acquire.
StopReason	Reason aqcuisition stopped.
Val	Number of samples acquired.

Returns

Success (0) or error code

17.3 PwrSnsr_AdvanceReadIndex()

EXPORT int

PwrSnsr_AdvanceReadIndex(

SessionID	Vi,

)

Send a command to the meter to notify it the user is done reading and to advance the read index.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

17.4 PwrSnsr_ClearBuffer()

EXPORT int

PwrSnsr_ClearBuffer(

SessionID Vi,

)

Sends a command to the power meter to clear all buffered readings. This method does not clear cached measurements accessible through GetAverageMeasurements, etc.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

17.5 PwrSnsr_ClearMeasurements()

EXPORT int

PwrSnsr_ClearMeasurements(

SessionID Vi,

)

Clears cached average, min, max, duration, start time, and sequence number measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

17.6 PwrSnsr_GetBufferedAverageMeasurements()

EXPORT int

PwrSnsr_GetBuffered AverageMeasurements(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Get the average power measurements that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1"
ValBufferSize	Buffer size of Val.
Val[]	Array of average measurements.
ValActualSize	Actual size of Val.

Returns

Success (0) or error code

17.7 PwrSnsr_GetBufferedMeasurementsAvailable()

EXPORT int

PwrSnsr_GetBuffered MeasurementsAvailable(

SessionID	Vi,
int*	MeasurementsAvailable

)

Gets the number of measurements available in the power meter's internal buffer.

Note:

The number of readings that have been acquired may be more or less.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MeasurementsAvailable	The number of measurements available in the internal buffer.

Returns

Success (0) or error code

17.8 PwrSnsr_GetContinuousCapture()

EXPORT int

PwrSnsr_GetContinuousCapture(

SessionID	Vi,
int*	ContinuousCapture

)

Get whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
ContinuousCapture	True if AcquireMeasurements will stop the measurement buffer session or continue captruing measurement buffer entries after being called.

Returns

Success (0) or error code

17.9 PwrSnsr_GetDuration()

EXPORT int

PwrSnsr_GetDuration(

SessionID	Vi,
float*	Duration

)

Get the time duration samples are captured during each timed mode acquisiton.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Duration	The duration in seconds samples are captured during each timed mode acquisitionl.

Returns

Success (0) or error code

17.10 PwrSnsr_GetDurations()

EXPORT int

PwrSnsr_GetDurations(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Get the duration entries in seconds that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of measurement duration in seconds.
ValActualSize	Actual size of the returned buffer.

Returns

Success (0) or error code

17.11 PwrSnsr_GetEndDelay()

EXPORT int

PwrSnsr_GetEndDelay(

SessionID	Vi,
float*	EndDelay

)

Get delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
EndDelay	The dealy time added to the detected end of a burst for analysis.

Returns

Success (0) or error code

17.12 PwrSnsr_GetEndQual()

EXPORT int

PwrSnsr_GetEndQual(

SessionID	Vi,
float*	EndQual

)

Get the minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
EndQual	The minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Returns

Success (0) or error code

17.13 PwrSnsr_GetGateMode()

EXPORT int

PwrSnsr_GetGateMode(

SessionID	Vi,
PwrSnsrMeasBuffGateEnum *	GateMode

)

Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. The gate signal may be internally or externally generated in several different ways.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
GateMode	Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code

17.14 PwrSnsr_GetMaxMeasurements()

EXPORT int

PwrSnsr_GetMaxMeasurements(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Get the maximum power measurements that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timeout	Channel number. For single instruments, set this to "CH1".
Count	Size of the buffer.
StopReason	Array of max measurements.
Val	Actual size of the returned array in the elements.

Returns

Success (0) or error code

17.15 PwrSnsr_GetMeasBuffEnabled()

EXPORT int

PwrSnsr_GetMeasBuffEnabled(

SessionID	Vi,
int*	MeasBuffEnabled

)

Get whether the measurement buffer has been enabled.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MeasBuffEnabled	True if measurement buffer is enabled.

Returns

Success (0) or error code

17.16 PwrSnsr_GetMeasurementsAvailable()

EXPORT int

PwrSnsr_GetMeasurementsAvailable(

SessionID	Vi,
Const char*	Channel,
int*	Val

)

Get the number of measurement entries available that were captured during AcquireMeasurements()

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Val	Number of measurement entries available.

Returns

Success (0) or error code

17.17 PwrSnsr_GetMinMeasurements()

EXPORT int

PwrSnsr_GetMinMeasurements(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Initiates new acquisition from the measurement buffer system (if acquisition is in the stopped state). Blocks until the number of measurements for each enabled channel is equal to count, or a time out has occurred.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val[]	Array of min measurements.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.18 PwrSnsr_GetOverRan()

EXPORT int

PwrSnsr_GetOverRan(

SessionID	Vi,
int*	OverRan

)

Get flag indicating whether the power meter's internal buffer filled up before being emptied.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
OverRan	True if the pwoer meter's internal buffer filled up before being emptied.

Returns

Success (0) or error code

17.19 PwrSnsr_GetPeriod()

EXPORT int

PwrSnsr_GetPeriod(

SessionID	Vi,
float*	Period

)

Get the period each timed mode acquisition (measurement buffer) is started.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Period	The period in seconds each tied mode acquisition is started.

Returns

Success (0) or error code

17.20 PwrSnsr_GetRdgsEnableFlag()

EXPORT int

PwrSnsr_GetRdgsEnableFlag(

SessionID	Vi,
int*	Flag

)

Get the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Flag	Bit masked value indicating which measurement arrays will be queried (see PwrSnsrRdgsEnabledFlag).

Returns

Success (0) or error code

17.21 PwrSnsr_GetReturnCount()

EXPORT int

PwrSnsr_GetReturnCount(

SessionID	Vi,
int*	ReturnCount

)

Get the return count for each measurement query.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
RetrunCount	The return count for each measurement query.

Returns

Success (0) or error code

17.22 PwrSnsr_GetSequenceNumbers()

EXPORT int

PwrSnsr_GetSequenceNumbers(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
unsigned int	Val[],
int*	ValActualSize

)

Get the sequence number entries that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of sequence numbers.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.23 PwrSnsr_GetSessionCount()

EXPORT int

PwrSnsr_GetSessionCount(

SessionID	Vi,
int*	SessionCount

)

Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
SessionCount	Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Returns

Success (0) or error code

17.24 PwrSnsr_GetStartDelay()

EXPORT int

PwrSnsr_GetStartDelay(

SessionID	Vi,
float*	StartDelay

)

Returns delay time added to the detected beginning of a burst for analysis. Used to exclude the rising edge of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartDelay	Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code

17.25 PwrSnsr_GetStartMode()

EXPORT int

PwrSnsr_GetStartMode(

SessionID	Vi,
PwrSnsrMeasBuffStartModeEnum*	StartMode

)

Get the mode used to start acquisition of buffer entries.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartMode	Mode used to start acquisition of buffer entries.

Returns

Success (0) or error code

17.26 PwrSnsr_GetStartQual()

EXPORT int

PwrSnsr_GetStartQual(

SessionID	Vi,
float*	StartQual

)

Returns the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartQual	The minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Returns

Success (0) or error code

17.27 PwrSnsr_GetStartTimes()

EXPORT int

PwrSnsr_GetStartTimes(

SessionID	Vi,
const char*	Channel,
int	ValbufferSize,
double	Val[],
int*	ValActualSize

)

Get the start time entries in seconds that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of start times.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.28 PwrSnsr_GetTimedOut()

EXPORT int

PwrSnsr_GetTimedOut(

SessionID	Vi,
int*	TimedOut

)

Check if the last measurement buffer session timed out.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Timeout	True if the last measurement buffer session timed out.

Returns

Success (0) or error code

17.29 PwrSnsr_GetWriteProtection()

EXPORT int

PwrSnsr_GetWriteProtection(

SessionID	Vi,
int*	WriteProtection

)

Get whether the measurement buffer is set to overwrite members that have not been read by the user.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
WriteProtection	Returns true if the measurement buffer is allowed to overwrite members that have not been read by the user.

Returns

Success (0) or error code

17.30 PwrSnsr_QueryAverageMeasurements()

EXPORT int

PwrSnsr_QueryAverageMeasurements(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Query the power meter for all buffered average power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer in elements.
Val	Array of average power measurements.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.31 PwrSnsr_QueryDurations()

EXPORT int

PwrSnsr_QueryDurations(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
float	Val,
int*	ValActualSize

)

Query the power meter for all buffered measurement durations in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, setthis to "CH1".
Count	Size of the buffer.
StopReason	Array of buffered measurement durations.
Val	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.32 PwrSnsr_QueryMaxMeasurements()

EXPORT int

PwrSnsr_QueryMaxMeasurements(

SessionID	Vi,
const charI	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Query the power meter for all buffered maximum power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of max measurements.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.33 PwrSnsr_QueryMinMeasurements()

EXPORT int

PwrSnsr_QueryMinMeasurements(

SessionID	Vi,
const charl	Channel,
int	ValBufferSize,
float	Val[],
int*	ValActualSize

)

Query the power meter for all buffered minimum power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val[]	Array of min measurements.
ValActtualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.34 PwrSnsr_QuerySequenceNumbers()

EXPORT int

PwrSnsr_QuerySequenceNumbers(

SessionID	Vi,
const char*	Channel,
int	ValBufferSize,
unsigned int	Val[],
int*	ValActualSize

)

Query the power meter for all buffered sequence numbers.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of sequence numbers.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.35 PwrSnsr_QueryStartTimes()

EXPORT int

PwrSnsr_QueryStartTimes(

	Vi,
SessionID	Channel,
const char*	ValBufferSize,
int	Val,
float	ValActualSize

)

Query the power meter for all buffered start times in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of start times in seconds.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code

17.36 PwrSnsr_ResetContinuousCapture()

EXPORT int

PwrSnsr_ResetContinuousCapture(

SessionID Vi,

)

Sets a flag indicating to restart continuous capture. This method allows the user to restart continuous acquisition. Has no effect if ContinuousCapture is set to false.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

17.37 PwrSnsr_SetContinuousCapture()

EXPORT int

PwrSnsr_SetContinuousCapture(

SessionID Vi,

int ContinuousCapture

)

Set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
ContinuousCapture	True to set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Returns

Success (0) or error code

17.38 PwrSnsr_SetDuration()

EXPORT int

PwrSnsr_SetDuration(

SessionID Vi,

float Duration

)

Set the duration samples are captured during each timed mode acquisition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Duration	The duration samples are captured during each timed mode acquistion in seconds.

Returns

Success (0) or error code

17.39 PwrSnsr_SetEndDelay()

EXPORT int

PwrSnsr_SetEndDelay(

SessionID Vi,

float EndDelay

)

Set delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
EndDelay	Delay time addded to the detected end of a burst for analysis.

Returns

Success (0) or error code

17.40 PwrSnsr_SetEndQual()

EXPORT int

PwrSnsr_SetEndQual(

SessionID	Vi,
float	EndQual

)

Set the minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
EndQual	The minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Returns

Success (0) or error code

17.41 PwrSnsr_SetGateMode()

EXPORT int

PwrSnsr_SetGateMode(

SessionID	Vi,
PwrSnsrMeasBuffGateEnum	GateMode

)

Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
GateMode	Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code

17.42 PwrSnsr_SetMeasBuffEnabled()

EXPORT int

PwrSnsr_SetMeasBuffEnabled(

SessionID	Vi,
int	MeasBuffEnabled

)

Enable or disable the measurement buffer. Disabling the measurement buffer enables modulated/CW measurements. Conversely, enabling it disables modulated/CW measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MeasBuffEnabled	True to enable measurement buffer, false to disable.

Returns

Success (0) or error code

17.43 PwrSnsr_SetPeriod()

EXPORT int

PwrSnsr_SetPeriod(

SessionID	Vi,
float	Period

)

Set the period each timed mode acquisition (measurement buffer) is started.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Period	The period in seconds each timed mode acquisition is started.

Returns

Success (0) or error code

17.44 PwrSnsr_SetRdgsEnableFlag()

EXPORT int

PwrSnsr_SetRdgsEnableFlag(

SessionID	Vi,
int	Flag

)

Set the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Flag	Bit masked value indicating which measurement array will be queried (see PwrSnsrRdgsEnabledFlag).

Returns

Success (0) or error code

17.45 PwrSnsr_SetReturnCount()

EXPORT int

PwrSnsr_SetReturnCount(

SessionID	Vi,
int	ReturnCount

)

Set the return count for each measurement query.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
ReturnCount	The return count for each measurement query.

Returns

Success (0) or error code

17.46 PwrSnsr_SetSessionCount()

EXPORT int

PwrSnsr_SetSessionCount(

SessionID	Vi,
int	SessionCount

)

Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
SessionCount	Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Returns

Success (0) or error code

17.47 PwrSnsr_SetSessionTimeout()

EXPORT int

PwrSnsr_SetSessionTimeout(

SessionID	Vi,
float	Seconds

)

Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Seconds	Set the time out value. Values less than or equal to 0 will be treated as infinite. Valid range: 0.001 to 1000

Returns

Success (0) or error code

17.48 PwrSnsr_SetStartDelay()

EXPORT int

PwrSnsr_SetStartDelay(

SessionID	Vi,
float	StartDelay

)

Set delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartDelay	Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code

17.49 PwrSnsr_SetStartMode()

EXPORT int

PwrSnsr_SetStartMode(

SessionID	Vi,
PwrSnsrMeasBuffStartModeEnum	StartMode

)

Set the mode used to start acquisition of buffer entries.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartMode	Mode used to start acquistion of buffer entries.

Returns

Success (0) or error code

17.50 PwrSnsr_SetStartQual()

EXPORT int

PwrSnsr_SetStartQual(

SessionID	Vi,
float	StartQual

)

Set the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
StartQual	The minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Returns

Success (0) or error code

17.51 PwrSnsr_SetWriteProtection()

EXPORT int

PwrSnsr_SetWriteProtection(

SessionID	Vi,
int	WriteProtection

)

Set whether to allow the measurement buffer to overwrite entries that have not been read by the user.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
WriteProtection	Set false to allow the measurement buffer to overwrite entries that have not been read by the user.

Returns

Success (0) or error code

17.52 PwrSnsr_StartAcquisition()

EXPORT int

PwrSnsr_StartAcquisition(

SessionID Vi,

)

Starts measurement buffer acquisition. This method allows the user to send a command to the power meter to begin buffering measurements without waiting for all measurements to be completed. Alternately, you can call the AcquireReadings method to start buffering measurements and wait for them to be read from the meter.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

17.53 PwrSnsr_StopAcquisition()

EXPORT int

PwrSnsr_StopAcquisition()

SessionID Vi,

)

Sends a command to stop the measurement buffer from acquiring readings.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code

Sensor RawIO

18.1 Command Summary

PwrSnsr_ReadByteArray()

PwrSnsr_ReadControl()

PwrSnsr_Write()

18.2 PwrSnsr_ReadByteArray()

EXPORT int

PwrSnsr_ReadByteArray(

SessionID	Vi,
const char*	Channel,
int	Count,
int	ValBufferSize,
unsigned char	Val[],
int*	ValActualSize

)

Reads byte array from the meter.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Count	Maximum count of bytes to return.
ValBufferSize	Size of the buffer.
Val[]	Byte array from the USB.
ValActualSize	Actual size of the returned array in bytes.

Returns

Success (0) or error code

18.3 PwrSnsr_ReadControl()

EXPORT int

PwrSnsr_ReadControl(

SessionID	Vi,
const char*	Channel,
int	Count,
int	ValBufferSize,
unsigned char	Val[],
int*	ValActualSize

)

Reads a control transfer on the USB.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Count	Maximum count to return.
ValBufferSize	Size of the buffer.
Val[]	Byte array from a USB control transfer.
ValActualSize	Actual size of the returned array in bytes.

Returns

Success (0) or error code

18.4 PwrSnsr_Write()

EXPORT int

PwrSnsr_Write(

SessionID	Vi,
const char*	Channel,
int	DataBufferSize,
unsigned char	Data[]

)

Write a byte array to the meter.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
DataBufferSize	Size of the buffer in bytes.
Data	Data to send.

Returns

Success (0) or error code

License Functions

19.1 Commands Summary

PwrSnsr_GetDongleSerialNumber()

PwrSnsr_GetExpirationDate()

PwrSnsr_GetNumberOfCals()

PwrSnsr_IsLicenseDongleConnected()

19.2 PwrSnsr_GetDongleSerialNumber()

EXPORT int

PwrSnsr_GetDongleSerialNumber(

long* val

)

Get the hardware license serial number.

Parameters

val	Serial number of the license dongle.

Returns

Success (0) or error code

19.3 PwrSnsr_GetExpirationDate()

EXPORT int

PwrSnsr_GetExpirationDate(

int* Date

)

Get the hardware license expiration date.

Parameters

Date	Expiration date in the format YYYYMMDD

Returns

Success (0) or error code

19.4 PwrSnsr_GetNumberOfCals()

EXPORT int

PwrSnsr_GetNumberOfCals(

long* val

)

Get the number of calibrations left on the license.

Parameters

val	Number of cals left.

Returns

Success (0) or error code

19.5 PwrSnsr_IsLicenseDongleConnected()

EXPORT int

PwrSnsr_IsLicenseDongleConnected(

int* val

)

Get whether the hardware license dongle is connected.

Parameters

val	Boolean. 1 for connected or 0 for not connected.

Returns

Success (0) or error code

Sensor Simulation

20.1 Commands Summary

PwrSnsrSignalUnits

PwrSnsrSimSignalTye

PwrSnsr_GetSimSignalAmplitude()

PwrSnsr_GetSimSignalCompression()

PwrSnsr_GetSimSignalDuty()

PwrSnsr_GetSimSignalModulation()

PwrSnsr_GetSimSignalPRF()

PwrSnsr_GetSimSignalType()

PwrSnsr_OpenSimMeter()

PwrSnsr_SetSimSignalAmplitude()

PwrSnsr_SetSimSignalCompression()

PwrSnsr_SetSimSignalDuty()

PwrSnsr_SetSimSignalModulation()

PwrSnsr_SetSimSignalPRF()

PwrSnsr_SetSimSignalType()

20.2 PwrSnsrSignalUnits

Unit selector for watts or dBm.

PwrSnsrSignalUnitsdBm	Selects dBm.
PwrSnsrSignalUnitsWatts	Selects watts.

20.3 PwrSnsrSimSignalType

Select simulated signal type.

PwrSnsrSimSignalPeriodic	Periodic waveform defined by PRF and duty cycle.
PwrSnsrSimSignalBurst	IFF-like burst signal.

20.4 PwrSnsr_GetSimSignalAmplitude()

EXPORT int

PwrSnsr_GetSimSignalAmplitude(

SessionID	Vi,
const char *	Channel,
float	Amplitude,
PwrSnsrSignalUnits	Units

)

Get the amplitude for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Amplitude	The simulated signal level (return value).
Units	Units of amplitude in dBm or Watts

Returns

Success (0) or error code

20.5 PwrSnsr_GetSimSignalCompression()

EXPORT int

PwrSnsr_GetSimSignalCompression(

SessionID	Vi,
const char *	Channel,
float	Percent

)

Get the percent compression for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Percent	The percent compression (return value).

Returns

Success (0) or error code

20.6 PwrSnsr_GetSimSignalDuty()

EXPORT int

PwrSnsr_GetSimSignalDuty(

SessionID	Vi,
const char *	Channel,
float	Duty

)

Get the simulated signal duty cycle in percent. Affects Periodic only.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Duty	Duty cycle in percent (retrun value).

Returns

Success (0) or error code

20.7 PwrSnsr_GetSimSignalModulation()

EXPORT int

PwrSnsr_GetSimSignalModulation(

SessionID	Vi,
const char *	Channel,
float	Percent

)

Get the percent modulation for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Percent	The percent modulation (return value).

Returns

Success (0) or error code

20.8 PwrSnsr_GetSimSignalPRF()

EXPORT int

PwrSnsr_GetSimSignalPRF(

SessionID	Vi,
const char *	Channel,
float *	PRF

)

Get the simulated signal PRF. Valid for Periodic and Burst.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
PRF	Pulse repetition frequency in Hz (return value).

Returns

Success (0) or error code

20.9 PwrSnsr_GetSimSignalType()

EXPORT int

PwrSnsr_GetSimSignalType(

SessionID	Vi,
const char *	Channel,
PwrSnsrSimSignalType*	SingalType

)

Get the simulated channel signal type.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
SingalType	The simulated signal type (return value).

Returns

Success (0) or error code

20.10 PwrSnsr_OpenSimMeter()

EXPORT int

PwrSnsr_OpenSimMeter(

int	NumChans,
char *	ResourceBuff,
int	ResourceBuffSize,
SessionID*	Vi

)

Open a simulated power meter session.

Parameters

NumChans	Number of channels
ResourceBuff	Buffer to read back the session name, can be NULL
ResourceBuffSize	Size of ResourceBuff in characters
Vi	SessionID handle (out parameter)

Returns

Success (0) or error code

20.11 PwrSnsr_SetSimSignalAmplitude()

EXPORT int

PwrSnsr_SetSimSignalAmplitude(

SessionID	Vi,
const char *	Channel,
float	Amplitude,
PwrSnsrSignalUnits	Units

)

Set the amplitude for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Amplitude	The simulated signal level (return value).
Units	Units of amplitude in dBm or Watts

Returns

Success (0) or error code

20.12 PwrSnsr_SetSimSignalCompression()

EXPORT int

PwrSnsr_SetSimSignalCompression(

SessionID	Vi,
const char *	Channel,
float	Percent

)

Set the percent compression for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Percent	The percent compression.

Returns

Success (0) or error code

20.13 PwrSnsr_SetSimSignalDuty()

EXPORT int

PwrSnsr_SetSimSignalDuty(

SessionID	Vi,
const char *	Channel,
float	Duty

)

Set the simulated signal duty cycle in percent. Affects Periodic only.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Duty	Duty cycle in percent (<99.0).

Returns

Success (0) or error code

20.14 PwrSnsr_SetSimSignalModulation()

EXPORT int

PwrSnsr_SetSimSignalModulation(

SessionID	Vi,
const char *	Channel,
float	Percent

)

Set the percent modulation for the signal on a simulation channel.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
Percent	The percent modulation.

Returns

Success (0) or error code

20.15 PwrSnsr_SetSimSignalPRF()

EXPORT int

PwrSnsr_SetSimSignalPRF(

	Vi,
SessionID	Channel,
const char *	PRF

)

Set the simulated signal PRF. Valid for Periodic and Burst.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
PRF	Pulse repetition frequency in Hz.

Returns

Success (0) or error code

20.16 PwrSnsr_SetSimSignalType()

EXPORT int

PwrSnsr_SetSimSignalType(

SessionID	Vi,
const char *	Channel,
PwrSnsrSimSignalType	SignalType

)

Set the simulated channel signal type.

Parameters

Vi,	The SessionID handle that you obtain from the PwrSnsr_OpenSimMeter function. The handle identifies a particular instrument session.
Channel	Channel Channel number. For single instruments, set this to "CH1".
SignalType	The simulated signal type.

Returns

Success (0) or error code