Programming Manual

MA244xxA Peak Power Sensors

Anritsu

Anritsu Company
490 Jarvis Drive
Morgan Hill, CA 95037-2809
USA

Part Number: 10585-00034
Revision: B
Published: February 2021
© Copyright 2021 Anritsu Company

Table of Contents

Chapter 1—General Information

1-1 Data Structures 1-1
PulseInfo Struct Reference 1-1
1-2 Macros 1-1

Chapter 2—Power Sensors Enumerations

2-1 General Purpose Enumerations 2-1
Acquisition Status: PwrSnsrAcquisitionStatusEnum 2-1
Trigger Mode for Pulsed Signals: PwrSnsrTriggerModeEnum 2-1
Values for Edge Trigger Slope: PwrSnsrTriggerSlopeEnum 2-1
Trigger Event Position on Displayed Sweep: PwrSnsrTriggerPositionEnum 2-1
Units Returned by Channel Measurements: PwrSnsrUnitsEnum 2-2
Marker Number: PwrSnsrMarkerNumberEnum 2-2
Video Bandwidth: PwrSnsrBandwidthEnum 2-2
Filter State: PwrSnsrFilterStateEnum 2-2
Pulse Calculation Units: PwrSnsrPulseUnitsEnum 2-2
Measurement Validity Condition Code: PwrSnsrCondCodeEnum 2-3
Acquisition System Trigger Status: PwrSnsrTriggerStatusEnum 2-3
Terminal Action: PwrSnsrTermActionEnum 2-3
Trigger Holdoff Mode: PwrSnsrHoldoffModeEnum 2-3
Statistical Capture Gating Value: PwrSnsrStatGatingEnum 2-4
Trigger Source for Synchronizing Data Acquisition: PwrSnsrTriggerSourceEnum 2-4
Multi IO Trigger Out Modes: PwrSnsrTrigOutModeEnum 2-4
Measurement Buffer Gate Modes: PwrSnsrMeasBuffGateEnum 2-5
Measurement Buffer Start Modes: PwrSnsrMeasBuffStartModeEnum 2-5
Measurement Buffer Stop Reason: PwrSnsrMeasBuffStopReasonEnum 2-5
Readings Enablement: PwrSnsrRdgsEnableFlag 2-5
2-2 ERROR_BASE Enumerations (0xBFFA0000L) 2-6
Error codes: PwrSnsrErrorCodesEnum 2-6
2-3 LIBUSB Error Enumerations 2-6

Chapter 3—Function Documentation

PwrSnsr_Abort() 3-1
PwrSnsr_AcquireMeasurements 3-1
PwrSnsr_AdvanceReadIndex() 3-2
PwrSnsr_Clear() 3-2
PwrSnsr_ClearBuffer() 3-2
PwrSnsr_ClearError() 3-3
PwrSnsr_ClearMeasurements() 3-3
PwrSnsr_ClearUserCal() 3-3
PwrSnsr_close() 3-4
PwrSnsr_EnableCapturePriority() 3-4
PwrSnsr_FetchAllMultiPulse() 3-4
PwrSnsr_FetchArrayMarkerPower() 3-5
PwrSnsr_FetchCCDFPercent() 3-6

PwrSnsr_FetchCCDFPower() . . . 3-7
PwrSnsr_FetchCCDFTrace() . . . 3-7
PwrSnsr_FetchCursorPercent() . . . 3-8
PwrSnsr_FetchCursorPower() . . . 3-8
PwrSnsr_FetchCWArray() . . . 3-9
PwrSnsr_FetchCWPower() . . . 3-9
PwrSnsr_FetchDistal() . . . 3-10
PwrSnsr_FetchDutyCycle() . . . 3-10
PwrSnsr_FetchEdgeDelay() . . . 3-11
PwrSnsr_FetExtendedWaveform() . . . 3-11
PwrSnsr_FetchFallTime() . . . 3-12
PwrSnsr_FetchIEEEBottom() . . . 3-12
PwrSnsr_FetchIEEETop() . . . 3-13
PwrSnsr_FetchIntervalAvg() . . . 3-13
PwrSnsr_FetchIntervalFilteredMax() . . . 3-14
PwrSnsr_FetchIntervalFilteredMin() . . . 3-14
PwrSnsr_FetchIntervalMax() . . . 3-15
PwrSnsr_FetchIntervalMaxAvg() . . . 3-15
PwrSnsr_FetchIntervalMin() . . . 3-16
PwrSnsr_FetchIntervalMinAvg() . . . 3-17
PwrSnsr_FetchIntervalPkToAvg() . . . 3-17
PwrSnsr_FetchMarkerAverage() . . . 3-18
PwrSnsr_FetchMarkerDelta() . . . 3-18
PwrSnsr_FetchMarkerMax() . . . 3-19
PwrSnsr_FetchMarkerMin() . . . 3-19
PwrSnsr_FetchMarkerRatio() . . . 3-20
PwrSnsr_FetchMarkerRDelta() . . . 3-20
PwrSnsr_FetchMarkerRRatio() . . . 3-21
PwrSnsr_FetchMesial() . . . 3-21
PwrSnsr_FetchOfftime() . . . 3-22
PwrSnsr_FetchOvershoot() . . . 3-22
PwrSnsr_FetchPeriod() . . . 3-22
PwrSnsr_FetchPowerArray() . . . 3-23
PwrSnsr_FetchPRF() . . . 3-24
PwrSnsr_FetchProximal() . . . 3-25
PwrSnsr_FetchPulseCycleAvg() . . . 3-25
PwrSnsr_FetchPulseOnAverage() . . . 3-26
PwrSnsr_FetchPulsePeak() . . . 3-26
PwrSnsr_FetchRiseTime() . . . 3-27
PwrSnsr_FetchStatMeasurementArray() . . . 3-27
PwrSnsr_FetchTimeArray() . . . 3-28
PwrSnsr_FetchWaveform() . . . 3-30
PwrSnsr_FetchWaveformMinMax() . . . 3-31
PwrSnsr_FetchWidth() . . . 3-32
PwrSnsr_FindResources() . . . 3-32
PwrSnsr_GetAcqStatusArray() . . . 3-33
PwrSnsr_GetAttenuation() . . . 3-33

PwrSnsr_GetAverage() . . . 3-34
PwrSnsr_GetBandwidth() . . . 3-34
PwrSnsr_GetBufferedAverageMeasurements() . . . 3-34
PwrSnsr_GetBufferedMeasurementsAvailable() . . . 3-35
PwrSnsr_GetCalFactor() . . . 3-35
PwrSnsr_GetCalFactors() . . . 3-36
PwrSnsr_GetCapture() . . . 3-37
PwrSnsr_GetCCDFTraceCount() . . . 3-37
PwrSnsr_GetChannelByIndex() . . . 3-38
PwrSnsr_GetChannelCount() . . . 3-38
PwrSnsr_GetChanTraceCount() . . . 3-38
PwrSnsr_GetContinuousCapture() . . . 3-39
PwrSnsr_GetCurrentTemp() . . . 3-39
PwrSnsr_GetDiagStatusArray() . . . 3-40
PwrSnsr_GetDistal() . . . 3-40
PwrSnsr_GetDongleSerialNumber() . . . 3-41
PwrSnsr_GetDuration() . . . 3-41
PwrSnsr_GetDurations() . . . 3-41
PwrSnsr_GetEnabled() . . . 3-42
PwrSnsr_GetEndDelay() . . . 3-42
PwrSnsr_GetEndGate() . . . 3-43
PwrSnsr_GetEndQual() . . . 3-43
PwrSnsr_GetError() . . . 3-44
PwrSnsr_GetExpirationDate() . . . 3-44
PwrSnsr_GetExternalSkew() . . . 3-44
PwrSnsr_GetFactoryCalDate() . . . 3-45
PwrSnsr_GetFetchLatency() . . . 3-45
PwrSnsr_GetFilterState() . . . 3-46
PwrSnsr_GetFilterTime() . . . 3-46
PwrSnsr_GetFirmwareVersion() . . . 3-46
PwrSnsr_GetFpgaVersion() . . . 3-47
PwrSnsr_GetFrequency() . . . 3-47
PwrSnsr_GetGateMode() . . . 3-48
PwrSnsr_GetGating() . . . 3-48
PwrSnsr_GetImpedance() . . . 3-49
PwrSnsr_GetInitiateContinuous() . . . 3-50
PwrSnsr_GetInternalSkew() . . . 3-50
PwrSnsr_GetIsAvailable() . . . 3-51
PwrSnsr_GetIsAvgSensor() . . . 3-51
PwrSnsr_GetIsRunning() . . . 3-52
PwrSnsr_GetManufactureDate() . . . 3-52
PwrSnsr_GetMarkerPixelPosition() . . . 3-53
PwrSnsr_GetMarkerTimePosition() . . . 3-53
PwrSnsr_GetMaxFreqHighBandwidth() . . . 3-53
PwrSnsr_GetMaxFreqLowBandwidth() . . . 3-54
PwrSnsr_GetMaxMeasurements() . . . 3-54
PwrSnsr_GetMaxTimebase() . . . 3-55

PwrSnsr_GetMeasBuffEnabled() . . . 3-55
PwrSnsr_GetMeasurementsAvailable() . . . 3-56
PwrSnsr_GetMemChanArchive() . . . 3-56
PwrSnsr_GetMesial() . . . 3-56
PwrSnsr_GetMinFreqHighBandwidth() . . . 3-57
PwrSnsr_GetMinFreqLowBandwidth() . . . 3-57
PwrSnsr_GetMinimumSupportedFirmware() . . . 3-58
PwrSnsr_GetMinimumTrig() . . . 3-58
PwrSnsr_GetMinMeasurements() . . . 3-59
PwrSnsr_GetModel() . . . 3-59
PwrSnsr_GetNumberOfCals() . . . 3-60
PwrSnsr_GetOffsetdB() . . . 3-60
PwrSnsr_GetOverRan() . . . 3-60
PwrSnsr_GetPeakHoldDecay() . . . 3-61
PwrSnsr_GetPeakHoldTracking() . . . 3-61
PwrSnsr_GetPeakPowerMax() . . . 3-62
PwrSnsr_GetPeakPowerMin() . . . 3-62
PwrSnsr_GetPercentPosition() . . . 3-62
PwrSnsr_GetPeriod() . . . 3-63
PwrSnsr_GetPowerPosition() . . . 3-63
PwrSnsr_GetProximal() . . . 3-64
PwrSnsr_GetPulseUnits() . . . 3-64
PwrSnsr_GetRdgsEnableFlag() . . . 3-64
PwrSnsr_GetReadingPeriod() . . . 3-65
PwrSnsr_GetReturnCount() . . . 3-65
PwrSnsr_GetSequenceNumbers() . . . 3-66
PwrSnsr_GetSerialNumber() . . . 3-66
PwrSnsr_GetSessionCount() . . . 3-67
PwrSnsr_GetSlaveSkew() . . . 3-67
PwrSnsr_GetStartDelay() . . . 3-67
PwrSnsr_GetStartGate() . . . 3-68
PwrSnsr_GetStartMode() . . . 3-68
PwrSnsr_GetStartQual() . . . 3-69
PwrSnsr_GetStartTimes() . . . 3-69
PwrSnsr_GetSweepTime() . . . 3-70
PwrSnsr_GetTempComp() . . . 3-70
PwrSnsr_GetTermAction() . . . 3-70
PwrSnsr_GetTermCount() . . . 3-71
PwrSnsr_GetTermTime() . . . 3-71
PwrSnsr_GetTimebase() . . . 3-72
PwrSnsr_GetTimedOut() . . . 3-72
PwrSnsr_GetTimeOut() . . . 3-72
PwrSnsr_GetTimePerPoint() . . . 3-73
PwrSnsr_GetTimespan() . . . 3-73
PwrSnsr_GetTraceStartTime() . . . 3-74
PwrSnsr_GetTrigDelay() . . . 3-74
PwrSnsr_GetTrigHoldoff() . . . 3-74

PwrSnsr_GetTrigHoldoffMode() . . . 3-75
PwrSnsr_GetTrigLevel() . . . 3-75
PwrSnsr_GetTrigMode() . . . 3-76
PwrSnsr_GetTrigPosition() . . . 3-76
PwrSnsr_GetTrigSlope() . . . 3-77
PwrSnsr_GetTrigSource() . . . 3-77
PwrSnsr_GetTrigStatus() . . . 3-77
PwrSnsr_GetTrigVernier() . . . 3-78
PwrSnsr_GetUnits() . . . 3-78
PwrSnsr_GetVerticalCenter() . . . 3-78
PwrSnsr_GetVerticalScale() . . . 3-79
PwrSnsr_GetWriteProtection() . . . 3-79
PwrSnsr_init() . . . 3-80
PwrSnsr_InitiateAquisition() . . . 3-80
PwrSnsr_IsLicenseDongleConnected() . . . 3-81
PwrSnsr_LoadMemChanFromArchive() . . . 3-81
PwrSnsr_MeasurePower() . . . 3-82
PwrSnsr_MeasureVoltage() . . . 3-82
PwrSnsr_QueryAverageMeasurements() . . . 3-83
PwrSnsr_QueryDurations() . . . 3-83
PwrSnsr_QueryMaxMeasurements() . . . 3-84
PwrSnsr_QueryMinMeasurements() . . . 3-84
PwrSnsr_QuerySequenceNumbers() . . . 3-85
PwrSnsr_QueryStartTimes() . . . 3-85
PwrSnsr_ReadArrayMarkerPower() . . . 3-86
PwrSnsr_ReadByteArray() . . . 3-87
PwrSnsr_ReadControl() . . . 3-87
PwrSnsr_ReadCWArray() . . . 3-88
PwrSnsr_ReadCWPower() . . . 3-89
PwrSnsr_ReadDutyCycle() . . . 3-89
PwrSnsr_ReadEdgeDelay() . . . 3-90
PwrSnsr_ReadFallTime() . . . 3-90
PwrSnsr_ReadIEEEBottom() . . . 3-91
PwrSnsr_ReadIEEETop() . . . 3-91
PwrSnsr_ReadIntervalAvg() . . . 3-92
PwrSnsr_ReadIntervalFilteredMax() . . . 3-92
PwrSnsr_ReadIntervalFilteredMin() . . . 3-93
PwrSnsr_ReadIntervalMax() . . . 3-93
PwrSnsr_ReadIntervalMaxAvg() . . . 3-94
PwrSnsr_ReadIntervalMin() . . . 3-94
PwrSnsr_ReadIntervalMinAvg() . . . 3-95
PwrSnsr_ReadIntervalPkToAvg() . . . 3-95
PwrSnsr_ReadMarkerAverage() . . . 3-96
PwrSnsr_ReadMarkerDelta() . . . 3-96
PwrSnsr_ReadMarkerMax() . . . 3-97
PwrSnsr_ReadMarkerMin() . . . 3-97
PwrSnsr_ReadMarkerRatio() . . . 3-98

PwrSnsr_ReadMarkerRDelta() . . . 3-98
PwrSnsr_ReadMarkerRRatio() . . . 3-99
PwrSnsr_ReadOfftime() . . . 3-99
PwrSnsr_ReadOvershoot() . . . 3-100
PwrSnsr_ReadPeriod() . . . 3-100
PwrSnsr_ReadPowerArray() . . . 3-100
PwrSnsr_ReadPRF() . . . 3-102
PwrSnsr_ReadPulseCycleAvg() . . . 3-102
PwrSnsr_ReadPulseOnAverage() . . . 3-103
PwrSnsr_ReadPulsePeak() . . . 3-103
PwrSnsr_ReadRiseTime() . . . 3-104
PwrSnsr_ReadSCPI() . . . 3-104
PwrSnsr_ReadSCPIBytes() . . . 3-105
PwrSnsr_ReadSCPIFromNamedParser() . . . 3-105
PwrSnsr_ReadTimeArray() . . . 3-106
PwrSnsr_ReadWaveform() . . . 3-107
PwrSnsr_ReadWaveformMinMax() . . . 3-108
PwrSnsr_ReadWidth() . . . 3-109
PwrSnsr_reset() . . . 3-109
PwrSnsr_ResetContinuousCapture() . . . 3-110
PwrSnsr_SaveToMemoryChannel() . . . 3-110
PwrSnsr_SaveUserCal() . . . 3-111
PwrSnsr_self_test() . . . 3-111
PwrSnsr_SendSCPIBytes() . . . 3-111
PwrSnsr_SendSCPICommand() . . . 3-112
PwrSnsr_SendSCPIToNamedParser() . . . 3-112
PwrSnsr_SetAverage() . . . 3-112
PwrSnsr_SetBandwidth() . . . 3-113
PwrSnsr_SetCalFactor() . . . 3-113
PwrSnsr_SetCapture() . . . 3-114
PwrSnsr_SetCCDFTraceCount() . . . 3-114
PwrSnsr_SetContinuousCapture() . . . 3-115
PwrSnsr_SetDistal() . . . 3-115
PwrSnsr_SetDuration() . . . 3-115
PwrSnsr_SetEnabled() . . . 3-116
PwrSnsr_SetEndDelay() . . . 3-116
PwrSnsr_SetEndGate() . . . 3-117
PwrSnsr_SetEndQual() . . . 3-117
PwrSnsr_SetExternalSkew() . . . 3-117
PwrSnsr_SetFetchLatency() . . . 3-118
PwrSnsr_SetFilterState() . . . 3-118
PwrSnsr_SetFilterTime() . . . 3-119
PwrSnsr_SetFrequency() . . . 3-119
PwrSnsr_SetGateMode() . . . 3-119
PwrSnsr_SetGating() . . . 3-120
PwrSnsr_SetHorizontalOffset() . . . 3-120
PwrSnsr_SetHorizontalScale() . . . 3-121
PwrSnsr_SetInitiateContinuous() . . . 3-121

PwrSnsr_SetInternalSkew() 3-122
PwrSnsr_SetMarkerPixelPosition() 3-122
PwrSnsr_SetMarkerTimePosition() 3-122
PwrSnsr_SetMeasBuffEnabled() 3-123
PwrSnsr_SetMesial() 3-123
PwrSnsr_SetOffsetdB() 3-124
PwrSnsr_SetPeakHoldDecay() 3-124
PwrSnsr_SetPeakHoldTracking() 3-125
PwrSnsr_SetPercentPosition() 3-125
PwrSnsr_SetPeriod() 3-125
PwrSnsr_SetPowerPosition() 3-126
PwrSnsr_SetProximal() 3-126
PwrSnsr_SetPulseUnits() 3-127
PwrSnsr_SetRdgsEnableFlag() 3-127
PwrSnsr_SetReturnCount() 3-128
PwrSnsr_SetSessionCount() 3-128
PwrSnsr_SetSessionTimeout() 3-128
PwrSnsr_SetSlaveSkew() 3-129
PwrSnsr_SetStartDelay() 3-129
PwrSnsr_SetStartGate() 3-130
PwrSnsr_SetStartMode() 3-130
PwrSnsr_SetStartQual() 3-130
PwrSnsr_SetTempComp() 3-131
PwrSnsr_SetTermAction() 3-131
PwrSnsr_SetTermCount() 3-132
PwrSnsr_SetTermTime() 3-132
PwrSnsr_SetTimebase() 3-132
PwrSnsr_SetTimeOut() 3-133
PwrSnsr_SetTimespan() 3-133
PwSrnsr_SetTrigDelay() 3-134
PwrSnsr_SetTrigHoldoff() 3-134
PwrSnsr_SetTrigHoldoffMode() 3-135
PwrSnsr_SetTrigLevel() 3-135
PwrSnsr_SetTrigMode() 3-135
PwrSnsr_SetTrigOutMode() 3-136
PwrSnsr_SetTrigPosition() 3-136
PwrSnsr_SetTrigSlope() 3-136
PwrSnsr_SetTrigSource() 3-137
PwrSnsr_SetTrigVernier() 3-137
PwrSnsr_SetUnits() 3-138
PwrSnsr_SetVerticalCenter() 3-138
PwrSnsr_SetVerticalScale() 3-138
PwrSnsr_SetWriteProtection() 3-139
PwrSnsr_StartAcquisition() 3-139
PwrSnsr_StatModeReset() 3-140
PwrSnsr_Status() 3-140
PwrSnsr_StopAcquisition() 3-140
PwrSnsr_Write() 3-141

PwrSnsr_Zero() 3-141
PwrSnsr_ZeroQuery() 3-142

Chapter 1 — General Information

The data in this Reference were imported from PwrSnsrLib.h.

1-1 Data Structures

This section contains brief descriptions of the data structures.

PulseInfo Struct Reference

Data structure containing pulse information.

Data Type	Name	Description
float	Width	Pulse width is defined as the interval between the first and second signal crossings of the mesial line.
float	Peak	Peak (max instantaneous) power measurement.
float	Min	Minimum instantaneous power measurement.
float	PulseAvg	Average power measurement for the pulse.
float	Position	Time position corresponding to the mesial crossing of the rising edge for the pulse.
float	RiseProximal	Position in time for the proximal crossing on the rising edge of the pulse.
float	RiseDistal	Position in time for the distal crossing on the rising edge of the pulse.
float	RiseTime	Rise time of the pulse.
float	FallProximal	Position in time for the proximal crossing on the falling edge of the pulse.
float	FallDistal	Position in time for the distal crossing on the falling edge of the pulse.
float	FallTime	Fall time of the pulse.

1-2 Macros

This section lists the macros available for the Power Sensors.

#define SUCCESS (0L)

#define CURRENT_TIMEOUT (-2)

#define EXPORT

#define ERROR_BASE (0xBFFA0000L)

Chapter 2 — Power Sensors Enumerations

2-1 General Purpose Enumerations

This section lists and describes the Enumerations.

Acquisition Status: PwrSnsrAcquisitionStatusEnum

Status and Value	Description
PwrSnsrAcqComplete = 1	The meter has completed the acquisition.
PwrSnsrAcqInProgress = 0	The meter is still acquiring data.
PwrSnsrAcqStatusUnknown = -1	The meter cannot determine the status of the acquisition.

Trigger Mode for Pulsed Signals: PwrSnsrTriggerModeEnum

Mode	Description
PwrSnsrTriggerModeNormal = 1	The power meter causes a sweep to be triggered each time the power level crosses the preset trigger level in the direction specified by the slope.
PwrSnsrTriggerModeAuto = 2	The power meter automatically triggers if the configured trigger does not occur within the meter's timeout period.
PwrSnsrTriggerModeAutoLevel = 3	The power meter automatically adjusts the trigger level the trigger level to halfway between the highest and lowest power levels detected.
PwrSnsrTriggerModeFreerun = 4	The power meter forces traces at a high rate to assist in locating the signal.

Values for Edge Trigger Slope: PwrSnsrTriggerSlopeEnum

Trigger Slope	Description
PwrSnsrTriggerSlopePositive = 1	A negative (falling) edge passing through the trigger level triggers the power meter.
PwrSnsrTriggerSlopeNegative = 0	A positive (rising) edge passing through the trigger level triggers the power meter.

Trigger Event Position on Displayed Sweep: PwrSnsrTriggerPositionEnum

Trigger Position	Description
PwrSnsrTriggerPositionLeft = 0	Left trigger position.
PwrSnsrTriggerPositionMiddle = 1	Middle trigger position.
PwrSnsrTriggerPositionRight = 2	Right trigger position.

Units Returned by Channel Measurements: PwrSnsrUnitsEnum

Units Returned	Description
PwrSnsrUnitsdBm = 0	dBm
PwrSnsrUnitswatts = 1	Watts
PwrSnsrUnitsvolts = 2	Volts
PwrSnsrUnitsDBV = 3	dBV
PwrSnsrUnitsDBMV = 4	dBmV
PwrSnsrUnitsDBUV = 5	dBuV

Marker Number: PwrSnsrMarkerNumberEnum

Marker	Description
PwrSnsrMarkerNumberMarker1 = 1	Marker 1
PwrSnsrMarkerNumberMarker2 = 2	Marker 2

Video Bandwidth: PwrSnsrBandwidthEnum

Bandwidth	Description
PwrSnsrBandwidthHigh = 0	High bandwidth.
PwrSnsrBandwidthLow = 1	Low bandwidth.

Filter State: PwrSnsrFilterStateEnum

Filter State	Description
PwrSnsrFilterStateOff = 0	Filter off.
PwrSnsrFilterStateOn = 1	Filter on.
PwrSnsrFilterStateAuto = 2	Automatically calculated filter.

Pulse Calculation Units: PwrSnsrPulseUnitsEnum

Calculation Units	Description
PwrSnsrPulseUnitsWatts = 0	Calculates distal, mesial, and proximal using watts.
PwrSnsrPulseUnitsVolts = 1	Calculates distal, mesial, and proximal using volts.

Measurement Validity Condition Code: PwrSnsrCondCodeEnum

Measurement Validity	Description
PwrSnsrCondCodeMeasurementStopped = -1	Measurement is STOPPED. Value returned is not updated.
PwrSnsrCondCodeError = 0	Error return. Measurement is not valid.
PwrSnsrCondCodeUnderrange = 2	An Over-range condition exists.
PwrSnsrCondCodeOverrange = 3	An Under-range condition exists.
PwrSnsrCondCodeNormal = 1	Normal return. No error.

Acquisition System Trigger Status: PwrSnsrTriggerStatusEnum

Trigger Status	Description
PwrSnsrTriggerStatusStopped = 0	Acquisition is stopped.
PwrSnsrTriggerStatusPretrig = 1	Acquiring data and waiting for the pre-trigger to be satisfied.
PwrSnsrTriggerStatusWaiting = 2	Meter is armed and waiting for trigger event.
PwrSnsrTriggerStatusAcquiringNew = 3	Acquiring new data.
PwrSnsrTriggerStatusAutoTrig = 4	Meter is autotriggering.
PwrSnsrTriggerStatusFreerun = 5	Trigger is in free-run mode.
PwrSnsrTriggerStatusTriggered = 6	Meter is currently triggered.
PwrSnsrTriggerStatusRunning = 7	Acquisition is running.

Terminal Action: PwrSnsrTermActionEnum

Select the action to take when either the statistical terminal count is reached or the terminal time has elapsed.

Terminal Action	Description
PwrSnsrTermActionStop = 0	Stop accumulating samples and hold the result.
PwrSnsrTermActionRestart = 1	Clear the CCDF and begin a new one.
PwrSnsrTermActionDecimate = 2	Divide all sample bins by 2 and continue.

Trigger Holdoff Mode: PwrSnsrHoldoffModeEnum

Holdoff Mode	Description
PwrSnsrHoldoffModeNormal = 1	Trigger will not arm again after the trigger conditions and its inverse are satisfied and then the amount of time set for trigger holdoff.
PwrSnsrHoldoffModeGap = 2	Trigger will not arm again after the trigger conditions are satisfied and then the amount of time set for trigger holdoff.

Statistical Capture Gating Value: PwrSnsrStatGatingEnum

Gating Value	Description
PwrSnsrStatGatingFreeRun = 0	No gating.
PwrSnsrStatGatingMarkers = 1	Gating is constrained to the portion of the trace between the markers.

Trigger Source for Synchronizing Data Acquisition: PwrSnsrTriggerSourceEnum

Trigger Source	Description
PwrSnsrTriggerSourceChannel1 = 0	Channel 1
PwrSnsrTriggerSourceExternal = 2	EXT setting uses the signal applied to the rear MULTI IO connector.
PwrSnsrTriggerSourceChannel2 = 1	Channel 2
PwrSnsrTriggerSourceChannel3 = 3	Channel 3
PwrSnsrTriggerSourceChannel4 = 4	Channel 4
PwrSnsrTriggerSourceChannel5 = 5	Channel 5
PwrSnsrTriggerSourceChannel6 = 6	Channel 6
PwrSnsrTriggerSourceChannel7 = 7	Channel 7
PwrSnsrTriggerSourceChannel8 = 8	Channel 8
PwrSnsrTriggerSourceChannel9 = 9	Channel 9
PwrSnsrTriggerSourceChannel10 = 10	Channel 10
PwrSnsrTriggerSourceChannel11 = 11	Channel 11
PwrSnsrTriggerSourceChannel12 = 12	Channel 12
PwrSnsrTriggerSourceChannel13 = 13	Channel 13
PwrSnsrTriggerSourceChannel14 = 14	Channel 14
PwrSnsrTriggerSourceChannel15 = 15	Channel 15
PwrSnsrTriggerSourceChannel16 = 16	Channel 16
PwrSnsrTriggerSourceIndependent = 17	Sets each sensor in a measurement group to use its own internal trigger.

Multi IO Trigger Out Modes: PwrSnsrTrigOutModeEnum

Trigger Mode	Description
PwrSnsrTrigOutModeMioOff = 0	
PwrSnsrTrigOutModeMioPullUp = 1	
PwrSnsrTrigOutModeMioTtl0 = 2	
PwrSnsrTrigOutModeMioTbRef = 3	
PwrSnsrTrigOutModeMioSweepHigh = 4	
PwrSnsrTrigOutModeMioSweepLow = 5	
PwrSnsrTrigOutModeMioTrigHigh = 6	
PwrSnsrTrigOutModeMioTrigLow = 7	
PwrSnsrTrigOutModeMioMaster = 8	
PwrSnsrTrigOutModeMioSlave = 9	

Measurement Buffer Gate Modes: PwrSnsrMeasBuffGateEnum

Gate Mode	Description
PwrSnsrMeasBuffGateBurst = 0	
PwrSnsrMeasBuffGateMarker = 1	
PwrSnsrMeasBuffGateExtGate = 2	
PwrSnsrMeasBuffGatePeriodic = 3	
PwrSnsrMeasBuffGateExtTrig = 4	

Measurement Buffer Start Modes: PwrSnsrMeasBuffStartModeEnum

Start Mode	Description
PwrSnsrMeasBuffStartModeImmediate = 1	
PwrSnsrMeasBuffStartModeExternalEnable = 2	
PwrSnsrMeasBuffStartModeExternalStart = 3	

Measurement Buffer Stop Reason: PwrSnsrMeasBuffStopReasonEnum

Reason	Description
PwrSnsrMeasBuffStopReasonNone = 0	
PwrSnsrMeasBuffStopReasonCountReached = 1	
PwrSnsrMeasBuffStopReasonTimedOut = 2	
PwrSnsrMeasBuffStopReasonBufferOverran = 3	

Readings Enablement: PwrSnsrRdgsEnableFlag

Reading	Description
PwrSnsrSequenceEnable = 1	Enable sequence array capture.
PwrSnsrStartTimeEnable = 2	Enable start time array capture.
PwrSnsrDurationEnable = 4	Enable duration array capture.
PwrSnsrMinEnable = 8	Enable min measurement array capture.
PwrSnsrAvgEnable = 16	Enable average measurement capture.
PwrSnsrMaxEnable = 32	Enable max measurement capture.

2-2 ERROR_BASE Enumerations (0xBFFA0000L)

Error codes: PwrSnsrErrorCodesEnum

Error Code	Description
PWR_SNSR_IO_GENERAL = -2147204588	IO error.
PWR_SNSR_IO_TIMEOUT = -2147204587	IO timeout error.
PWR_SNSR_MODEL_NOT_SUPPORTED = -2147204586	Instrument model does not support this feature.
PWR_SNSR_INV_PARAMETER = -1073807240	Invalid parameter value.
PWR_SNSR_ERROR_INVALID_SESSION_HANDLE = -1074130544	Session ID invalid.
PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE = -1074134947	Status not available.
PWR_SNSR_ERROR_RESET_FAILED = -1074134945	Reset failed.
PWR_SNSR_ERROR_RESOURCE_UNKNOWN = -1074134944	Unknown resource descriptor.
PWR_SNSR_ERROR_ALREADY_INITIALIZED = -1074134943	Session already initialized.
PWR_SNSR_ERROR_OUT_OF_MEMORY = -1074134954	Out of memory.
PWR_SNSR_ERROR_OPERATION_PENDING = -1074134953	Operation pending.
PWR_SNSR_ERROR_NULL_POINTER = -1074134952	Null pointer not allowed.
PWR_SNSR_ERROR_UNEXPECTED_RESPONSE = -1074134951	Unexpected response from the instrument.
PWR_SNSR_ERROR_NOT_INITIALIZED = -1074135011	Session not initialized.

2-3 LIBUSB Error Enumerations

Error Code	Description
PWR_SNSR_LIBUSB_ERROR_IO = -1,	Input-output error
PWR_SNSR_LIBUSB_ERROR_INVALID_PARAM = -2,	Invalid parameter
PWR_SNSR_LIBUSB_ERROR_ACCESS = -3,	Access denied (insufficient permissions)
PWR_SNSR_LIBUSB_ERROR_NO_DEVICE = -4,	No such device (it may have been disconnected)
PWR_SNSR_LIBUSB_ERROR_NOT_FOUND = -5,	Entity not found
PWR_SNSR_LIBUSB_ERROR_BUSY = -6,	Resource busy
PWR_SNSR_LIBUSB_ERROR_TIMEOUT = -7,	Operation timed out
PWR_SNSR_LIBUSB_ERROR_OVERFLOW = -8,	Overflow
PWR_SNSR_LIBUSB_ERROR_PIPE = -9,	Pipe error
PWR_SNSR_LIBUSB_ERROR_INTERRUPTED = -10,	System call interrupted (perhaps due to signal)
PWR_SNSR_LIBUSB_ERROR_NO_MEM = -11,	Insufficient memory
PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED = -12,	Operation not supported or unimplemented on this platform
PWR_SNSR_LIBUSB_ERROR_OTHER = -99,	Other error

Chapter 3 — Function Documentation

This chapter describes the function of and the expected parameters for the Power Sensor functions.

PwrSnsr_Abort()

Terminates any measurement in progress and resets the state of the trigger system. Note that Abort will leave the measurement in a stopped condition with all current measurements cleared.

```
EXPORT int PwrSnsr_Abort
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_AcquireMeasurements

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.

```
EXPORT int PwrSnsr_AcquireMeasurements
(
SessionID Vi,
double Timeout,
int Count,
PwrSnsrMeasBuffSt opReasonEnum *StopReason,
int *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_AdvanceReadIndex()

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

```
EXPORT int PwrSnsr_Adv anceReadIndex
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_Clear()

Clear all data buffers. Clears averaging filters to empty.

```
EXPORT int PwrSnsr_Clear
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_ClearBuffer()

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

```
EXPORT int PwrSnsr_ClearBuffer
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_ClearError()

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

```
EXPORT int PwrSnsr_ClearError
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_ClearMeasurements()

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

```
EXPORT int PwrSnsr_ClearMeasurement
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_ClearUserCal()

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

```
EXPORT int PwrSnsr_ClearUserCal
(
SessionID Vi, const char *Channel
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_close()

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

```
EXPORT int PwrSnsr_close
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_EnableCapturePriority()

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

```
EXPORT int PwrSnsr_EnableCapturePriority
(
SessionID Vi,
const char *Channel,
int Enabled
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Enabled: If set to 1, enables buffered mode. If set to zero, disables capture priority(default).

Returns

Success (0) or error code.

PwrSnsr_FetchAllMultiPulse()

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).

```
EXPORT int PwrSnsr_FetchAllMultiPulse
(SessionID Vi,
const char *Channel,
int PulseInfosSize,
PulseInfo PulseInfos[],
```

```
int *PulseInfosActualSize
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

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.

PwrSnsr_FetchArrayMarkerPower()

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

```
EXPORT int PwrSnsr_FetchArrayMarkerPower
(
SessionID Vi,
const char *Channel,
float *AvgPower,
PwrSnsrCondCode
Enum *AvgPowerCondCode,
float *MaxPower,
PwrSnsrCondCode
Enum *MaxPowerCondCode,
float *MinPower,
PwrSnsrCondCode
Enum *MinPowerCondCode,
float *PkToAvgRatio,
PwrSnsrCondCode Enum *PkToAvgRatioCondC ode,
float *Marker1Power,
PwrSnsrCondCode Enum *Marker1PowerCondC ode,
float *Marker2Power,
PwrSnsrCondCode Enum *Marker2PowerCondC ode,
float *MarkerRatio,
PwrSnsrCondCode Enum *MarkerRatioCondCode
)
```

Parameters

Vi: The SessionID handle returned by 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 the markers.

MaxPowerCondCode: Condition code.

MinPower: Minimum power between the markers.

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.

PwrSnsr_FetchCCDFPercent()

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

```
EXPORT int PwrSnsr_FetchCCDFPercent
(
SessionID Vi,
const char *Channel,
double Power,
PwrSnsrCondCodeEnum *CondCode,
double *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchCCDFPower()

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

```
EXPORT int PwrSnsr_FetchCCDFPower
(
SessionID Vi,
const char *Channel,
double Percent,
PwrSnsrCondCodeEnum *CondCode,
double *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchCCDFTrace()

Returns the points in the CCDF trace.

```
EXPORT int PwrSnsr_FetchCCDFTrace
(
SessionID Vi,
const char *Channel,
int TraceBufferSize,
float Trace[],
int *TraceActualSize
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TraceBufferSize
Trace: TraceActualSize

Returns

Success (0) or error code.

PwrSnsr_FetchCursorPercent()

Returns the percent CCDF at the cursor.

```
EXPORT int PwrSnsr_FetchCursorPercent
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
double *Val
)
```

Parameters

Vi: The SessionID handle returned by 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:

Returns

Success (0) or error code.

PwrSnsr_FetchCursorPower()

Returns the power CCDF in dB at the cursor.

```
EXPORT int PwrSnsr_FetchCursorPower
(
SessionID
Vi const char *Channel,
PwrSnsr CondCode
Enum *CondCode,
double *Val
)
```

Parameters

Vi: The SessionID handle returned by 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:

Returns

Success (0) or error code.

PwrSnsr_FetchCWArray()

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.

```
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
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

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.

PwrSnsr_FetchCWPower()

Returns the most recently acquired CW power.

```
EXPORT int PwrSnsr_FetchCWPower
(
SessionID Vi,
```

```
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchDistal()

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

```
EXPORT int PwrSnsr_FetchDistal
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCode Enum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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 the distal level in the current channel units.

Returns

Success (0) or error code.

PwrSnsr_FetchDutyCycle()

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

```
EXPORT int PwrSnsr_FetchDutyCycle
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCode
Enum *IsValid,
```

```
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchEdgeDelay()

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

```
EXPORT int PwrSnsr_FetchEdgeDelay
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

IsValid Val:

Returns

Success (0) or error code.

PwrSnsr_FetExtendedWaveform()

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.

```
EXPORT int PwrSnsr_FetchExtendedWaveform
(
SessionID Vi,
const char *Channel, WaveformArrayBuffer
int Size,
float WaveformArray[],
WaveformArrayActual int *Size,
```

```
int Count
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

WaveformArrayBuffer Size: Number of elements in the WaveformArray buffer

WaveformArray: Waveform buffer.

WaveformArrayActual Size: Number of elements updated with data.

Count: Number of points to capture.

Returns

Success (0) or error code.

PwrSnsr_FetchFallTime()

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

```
EXPORT int PwrSnsr_FetchFallTime
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchIEEEBottom()

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.

```
EXPORT int PwrSnsr_FetchIEEEBottom
(
SessionID Vi,
```

```
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchIEEETop()

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

```
EXPORT int PwrSnsr_FetchIEEETop
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchIntervalAvg()

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.

```
EXPORT int PwrSnsr_FetchIntervalAvg
(
SessionID Vi,
const char *Channel,
```

```
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalFilteredMax()

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.

```
EXPORT int PwrSnsr_FetchIntervalFilteredMax
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code.

Val:

Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalFilteredMin()

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

```
EXPORT int PwrSnsr_FetchIntervalFilteredMin
(
```

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

Parameters

Vi: The SessionID handle returned by 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.

Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalMax()

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.

```
EXPORT int PwrSnsr_FetchIntervalMax
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalMaxAvg()

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

```
EXPORT int PwrSnsr_FetchIntervalMaxAvg
```

```
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalMin()

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.

```
EXPORT int PwrSnsr_FetchIntervalMin
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCode Enum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalMinAvg()

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

```
EXPORT int PwrSnsr_FetchIntervalMinAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchIntervalPkToAvg()

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.

```
EXPORT int PwrSnsr_FetchIntervalPkToAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerAverage()

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

```
EXPORT int PwrSnsr_FetchMarkerAverage
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Marker: Marker number.

IsValid: Condition code.

Val: Measurement value

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerDelta()

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

```
EXPORT int PwrSnsr_FetchMarkerDelta
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchMarkerMax()

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

```
EXPORT int PwrSnsr_FetchMarkerMax
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Marker: Marker number.

IsValid:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerMin()

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

```
EXPORT int PwrSnsr_FetchMarkerMin
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Marker: Marker number.

IsValid:

Val: measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerRatio()

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

```
EXPORT int PwrSnsr_FetchMarkerRatio
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerRDelta()

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

```
EXPORT int PwrSnsr_FetchMarkerRDelta
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *condCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMarkerRRatio()

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

```
EXPORT int PwrSnsr_FetchMarkerRRatio
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned byreturned by 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.

Condition code:

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_FetchMesial()

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

```
EXPORT int PwrSnsr_FetchMesial
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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 the mesial level in the current channel units.

Returns

Success (0) or error code.

PwrSnsr_FetchOfftime()

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

```
EXPORT int PwrSnsr_FetchOfftime
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchOvershoot()

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

```
EXPORT int PwrSnsr_FetchOvershoot
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchPeriod()

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

```
EXPORT int PwrSnsr_FetchPeriod
(
```

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

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchPowerArray()

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.

```
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
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchPRF()

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

```
EXPORT int PwrSnsr_FetchPRF
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchProximal()

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

```
EXPORT int PwrSnsr_FetchProximal
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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 the proximal level in the current channel units.

Returns

Success (0) or error code.

PwrSnsr_FetchPulseCycleAvg()

Returns the average power of the entire pulse.

```
EXPORT int PwrSnsr_FetchPulseCycleAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchPulseOnAverage()

Average power of the ON portion of the pulse.

```
EXPORT int PwrSnsr_FetchPulseOnAverage
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCode Enum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchPulsePeak()

Returns the peak amplitude during the pulse.

```
EXPORT int PwrSnsr_FetchPulsePeak
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchRiseTime()

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

```
EXPORT int PwrSnsr_FetchRiseTime
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchStatMeasurementArray()

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

Measurements performed are: long term average, peak and minimum amplitude, peak-to- average ratio, amplitude at the CCDF percent cursor, statistical percent at the CCDF power cursor, and the sample population size in samples. Note the peak-to-average ratio is returned in dB for logarithmic channel units, and percent for all other channel units.

```
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
)
```

Parameters

Vi:	The SessionID handle returned by 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.
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.

PwrSnsr_FetchTimeArray()

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 for those commands. Units are the same as the channel's units.

```
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
)
```

Parameters

Vi: The SessionID handle returned by 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.

OfftimeValid: 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.

PwrSnsr_FetchWaveform()

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.

```
EXPORT int PwrSnsr_FetchWaveform
(
SessionID Vi,
const char *Channel,
int WaveformArrayBufferSize,
float WaveformArray[],
int *WaveformArrayActualSize
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

WaveformArrayBuffer Size: 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.

WaveformArrayActual Size: Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

PwrSnsr_FetchWaveformMinMax()

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.

```
EXPORT int PwrSnsr_FetchWaveformMinMax
(
```

```
SessionID Vi,
const char *Channel,
int MinWaveformBufferSize,
float MinWaveform[],
int *MinWaveformActualSize
int MaxWaveformBufferSize,
float MaxWaveform[],
int *MaxWaveformActualSize
int WaveformArrayBuffer Size,

float WaveformArray[],
int *WaveformArrayActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FetchWidth()

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

```
EXPORT int PwrSnsr_FetchWidth
(
```

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

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_FindResources()

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

```
EXPORT int PwrSnsr_FindResources
(
const char *Delimiter,
int ValBufferSize,
char Val[]
)
```

Parameters

Delimiter: The string used to delimit the list of resources i.e., "|", " ", ";", etc.

ValBufferSize: Number of elements in Val.

Val: The return string.

Returns

Success (0) or error code.

PwrSnsr_GetAcqStatusArray()

Returns data about the status of the acquisition system.

```
EXPORT int PwrSnsr_GetAcqStatusArray
(
SessionID Vi,
const char *Channel,
int *SweepLength,
double *SampleRate,
```

```
double *SweepRate,
double *SweepTime,
double *StartTime,
int *StatusWord
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetAttenuation()

Attenuation in dB for the sensor.

```
EXPORT int PwrSnsr_GetAttenuation
(
SessionID Vi,
const char *Channel,
float *Attenuation
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Attenuation:

Returns

Success (0) or error code.

PwrSnsr_GetAverage()

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

```
EXPORT int PwrSnsr_GetAverage
(
SessionID *Vi,
```

```
const char *Channel,
int Average
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Average:

Returns

Success (0) or error code.

PwrSnsr_GetBandwidth()

Get the sensor video bandwidth for the selected sensor.

```
EXPORT int PwrSnsr_GetBandwidth
(
SessionID Vi,
const char *Channel,
PwrSnsrBandwidthEnum *Bandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Bandwidth:

Returns

Success (0) or error code.

PwrSnsr_GetBufferedAverageMeasurements()

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

```
EXPORT int PwrSnsr_GetBuffered AverageMeasurements
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetBufferedMeasurementsAvailable()

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.

```
EXPORT int PwrSnsr_GetBufferedMeasurementsAvailable
(
SessionID Vi,
int *MeasurementsAvailable
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MeasurementsAvailable: 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.

Returns

Success (0) or error code.

PwrSnsr_GetCalFactor()

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

```
EXPORT int PwrSnsr_GetCalFactor
(
SessionID Vi,
const char *Channel,
float *CalFactor
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

CalFactor:

Returns

Success (0) or error code.

PwrSnsr_GetCalFactors()

Query information associated with calibration factors.

```
EXPORT int PwrSnsr_GetCalFactors
(
SessionID Vi,
const char *Channel,
float *MaxFrequency,
float *MinFrequency,
int FrequencyListBufferSize,
float FrequencyList[],
int *FrequencyListActualSize
int CalFactorListBufferSize,
float CalFactorList[],
int *CalFactorListActualSize,
PwrSnsrBandwidthEnum Bandwidth
)
```

Parameters

Vi:	The SessionID handle returned by 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.
CalFactorListActualSize:	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.

PwrSnsr_GetCapture()

Get whether statistical capture is enabled.

```
EXPORT int PwrSnsr_GetCapture
(
SessionID Vi.
const char *Channel,
int *Capture
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Capture:

Returns

Success (0) or error code.

PwrSnsr_GetCCDFTraceCount()

Get the number of points in the CCDF trace plot.

```
EXPORT int PwrSnsr_GetCCDFTraceCount
(
SessionID Vi,
const char *Channel,
int *TraceCount
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TraceCount:

Returns

Success (0) or error code.

PwrSnsr_GetChannelByIndex()

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

```
EXPORT int PwrSnsr_GetChannelByIndex
(
SessionID Vi,
int BuffSize,
char Channel[],
int Index
```

```
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Buffer size for Channel.

Channel: Channel number buffer.

Index: The index of the channel.

Returns

Success (0) or error code.

PwrSnsr_GetChannelCount()

Get number of channels.

```
EXPORT int PwrSnsr_GetChannelCount
(
SessionID Vi,
int *Count
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Count: Number of channels

Returns

Success (0) or error code.

PwrSnsr_GetChanTraceCount()

Get the number of points in the CCDF trace plot.

```
EXPORT int PwrSnsr_GetChanTraceCount
(
SessionID Vi,
const char *Channel,
int TraceCount
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetContinuousCapture()

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

```
EXPORT int PwrSnsr_GetContinuousCapture
(
SessionID Vi,
int *ContinuousCapture
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

ContinuousCapture: True if AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Returns

Success (0) or error code.

PwrSnsr_GetCurrentTemp()

Get current sensor internal temperature in degrees C.

```
EXPORT int PwrSnsr_GetCurrentTemp
(
SessionID Vi,
const char * Channel,
double* CurrentTemp
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

CurrentTemp:

Returns

Success (0) or error code.

PwrSnsr_GetDiagStatusArray()

Returns diagnostic data.

```
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
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetDistal()

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

```
(
SessionID Vi,
const char * Channel,
float* Distal
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Distal:

Returns

Success (0) or error code.

PwrSnsr_GetDongleSerialNumber()

Get the hardware license serial number.

```
EXPORT int PwrSnsr_Get DongleSerialNumber
(
long *val
)
```

Parameters

Val: Serial number of the license dongle

Returns

Success (0) or error code.

PwrSnsr_GetDuration()

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

```
EXPORT int PwrSnsr_GetDuration
(
SessionID Vi,
float *Duration
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Duration: The duration in seconds samples are captured during each timed mode acquisiton.

Returns

Success (0) or error code.

PwrSnsr_GetDurations()

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

```
EXPORT int PwrSnsr_GetDurations
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
```

```
)
```

Parameters

Vi: The SessionID handle returned by 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 durations in seconds.

ValActualSize: Actual size of the returned buffer.

Returns

Success (0) or error code.

PwrSnsr_GetEnabled()

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.

```
EXPORT int PwrSnsr_GetEnabled
(
SessionID Vi,
const char *Channel,
int *Enabled
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetEndDelay()

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.

```
EXPORT int PwrSnsr_GetEndDelay
(
SessionID Vi,
float *EndDelay
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

EndDelay: The delay time added to the detected end of a burst for analysis.

Returns

Success (0) or error code.

PwrSnsr_GetEndGate()

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

```
(
SessionID Vi,
const char * Channel,
float* EndGate
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

EndGate:

Returns

Success (0) or error code.

PwrSnsr_GetEndQual()

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

```
EXPORT int PwrSnsr_GetEndQual
(
SessionID Vi,
float * EndQual
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetError()

This function retrieves and then clears the error information for the session. Normally, the error information describes the first error that occurred since the user last called the Get Error or Clear Error function.

```
EXPORT int PwrSnsr_GetError
(
```

```
SessionID Vi,
int *ErrorCode,
int ErrorDescriptionBufferSize,
char ErrorDescription[]
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

ErrorCode:

ErrorDescriptionBufferSize:

ErrorDescription:

Returns

Success (0) or error code.

PwrSnsr_GetExpirationDate()

Get the hardware license expiration date.

```
EXPORT int PwrSnsr_GetExpirationDate
(
int *Date
)
```

Parameters

Date: expiration date in the format YYYYMMDD

Returns

Success (0) or error code.

PwrSnsr_GetExternalSkew()

Gets the skew in seconds for the external trigger.

```
EXPORT int PwrSnsr_GetExternalSkew
(
SessionID Vi,
const char *Channel,
float *External
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetFactoryCalDate()

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

```
EXPORT int PwrSnsr_GetFactoryCalDate
(
SessionID Vi,
const char *Channel, FactoryCalDateBuffer
int Size,
char FactoryCalDate[]
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

FactoryCalDateBufferSize

Size of FactoryCalDate in bytes.

FactoryCalDate:

Returns: Success (0) or error code.

PwrSnsr_GetFetchLatency()

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

```
EXPORT int PwrSnsr_GetFetchLatency
(
SessionID Vi,
int *Latency
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Latency: Fetch latency in ms.

Returns

Success (0) or error code.

PwrSnsr_GetFilterState()

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

```
EXPORT int PwrSnsr_GetFilterState
(
SessionID Vi,
const char *Channel,
PwrSnsrFilterStateEnum *FilterState
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

ManufactureDateBufferSize:

ManufactureDate:

Returns

Success (0) or error code.

PwrSnsr_GetFilterTime()

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

```
EXPORT int PwrSnsr_SetFilterState
(
SessionID Vi,
const char * Channel,
PwrSnsrFilterStateEnum FilterState
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

FilterTime:

Returns

Success (0) or error code.

PwrSnsr_GetFirmwareVersion()

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

```
EXPORT int PwrSnsr_GetFirmwareVersion
(
SessionID Vi,
const char *Channel,
FirmwareVersionBufferSize int,
```

```
char FirmwareVersion[]
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetFpgaVersion()

Get the sensor FPGA version.

```
EXPORT int PwrSnsr_GetFpgaVersion
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
char Val[]
)
```

Parameters

Vi: The SessionID handle returned by 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 staoring the version

Returns

Success (0) or error code.

PwrSnsr_GetFrequency()

Get the RF frequency for the current sensor.

```
EXPORT int PwrSnsr_GetFrequency
(
SessionID Vi,
const char *Channel,
float *Frequency
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetGateMode()

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.

```
EXPORT int
PwrSnsr_GetGateMode (
SessionID Vi,
PwrSnsrMeasBuffGateEnum *GateMode
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetGating()

Get whether statistical capture is enabled.

```
EXPORT int PwrSnsr_GetGating
(
SessionID Vi,
const char *Channel,
PwrSnsrStatGatingEnum *Gating
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1" whether the statical capture is gated by markers or free-running.

Gating:

Returns

Success (0) or error code.

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).

```
EXPORT int PwrSnsr_GetHorizontalOffset
(
SessionID Vi,
const char *Channel,
double *HorizontalOffset
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

HorizontalOffset:

Returns

Success (0) or error code.

Get the statistical mode horizontal scale in dB/Div.

```
EXPORT int PwrSnsr_GetHorizontalScale
(
SessionID Vi,
const char *Channel,
double *HorizontalScale
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

HorizontalScale:

Returns

Success (0) or error code.

PwrSnsr_GetImpedance()

Input impedance of the sensor.

```
EXPORT int PwrSnsr_GetImpedance
(
SessionID Vi,
const char *Channel,
float *Impedance
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Impedance:

Returns

Success (0) or error code.

PwrSnsr_GetInitiateContinuous()

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.

```
EXPORT int PwrSnsr_GetInitiateContinuous
(
SessionID Vi,
int *InitiateContinuous
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetInternalSkew()

Gets the skew in seconds for the internal trigger.

```
EXPORT int PwrSnsr_GetInternalSkew
(
SessionID Vi,
const char *Channel,
float *InternalSkew
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetIsAvailable()

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

```
EXPORT int PwrSnsr_GetIsAvailable
(
SessionID Vi,
const char *Channel,
int *IsAvailable
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

IsAvailable: True if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled.

Returns

Success (0) or error code.

PwrSnsr_GetIsAvgSensor()

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

```
EXPORT int PwrSnsr_GetIsAvgSensor
(
SessionID Vi,
const char *Channel,
int *IsAvgSensor
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetIsRunning()

Returns true if modulated/CW measuremnts are actively running.

```
EXPORT int PwrSnsr_GetIsRunning
(
SessionID Vi,
const char *Channel,
int IsRunning
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetManufactureDate()

Date the sensor was manufactured in the following format YYYYmmDD.

```
EXPORT int PwrSnsr_GetManufactureDate
(
SessionID Vi,
const char *Channel,
int ManufactureDateBufferSize,
char ManufactureDate[]
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

ManufactureDateBufferSize: Size of ManufactureDate in bytes.

ManufactureDate: Return value.

Returns

Success (0) or error code.

PwrSnsr_GetMarkerPixelPosition()

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

```
EXPORT int PwrSnsr_GetMarkerPixelPosition
```

```
(
SessionID Vi,
int MarkerNumber,
int *PixelPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber

PixelPosition:

Returns

Success (0) or error code.

PwrSnsr_GetMarkerTimePosition()

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

Note that 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.

```
EXPORT int PwrSnsr_GetMarkerTimePosition
(
SessionID Vi,
int MarkerNumber,
float * TimePosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber

TimePosition:

Returns

Success (0) or error code.

PwrSnsr_GetMaxFreqHighBandwidth()

Maximum frequency carrier the sensor can measure in high bandwidth.

```
EXPORT int PwrSnsr_GetMaxFreqHighBandwidth
(
SessionID Vi,
const char *Channel,
float *MaxFreqHighBandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MaxFreqHighBandwidth:

Returns

Success (0) or error code.

PwrSnsr_GetMaxFreqLowBandwidth()

Maximum frequency carrier the sensor can measure in low bandwidth.

```
EXPORT int PwrSnsr_GetMaxFreqLowBandwidth
(
SessionID Vi,
const char *Channel,
float *MaxFreqLowBandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MaxFreqLowBandwidth:

Returns

Success (0) or error code.

PwrSnsr_GetMaxMeasurements()

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

```
EXPORT int PwrSnsr_GetMaxMeasurements
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetMaxTimebase()

Gets the maximum timebase setting available.

```
EXPORT int PwrSnsr_GetMaxTimebase
(
SessionID Vi,
float *MaxTimebase
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MaxTimebase:

Returns

Success (0) or error code.

PwrSnsr_GetMeasBuffEnabled()

Get whether the measurement buffer has been enabled.

```
EXPORT int PwrSnsr_GetMeasBuffEnabled
(
SessionID Vi,
int *MeasBuffEnabled
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MeasBuffEnabled: True if measurement buffer is enabled.

Returns

Success (0) or error code.

PwrSnsr_GetMeasurementsAvailable()

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

```
EXPORT int PwrSnsr_GetMeasurementsAvailable
(
```

```
SessionID Vi,
const char *Channel,
int *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetMemChanArchive()

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

```
EXPORT int PwrSnsr_GetMemChanArchive
(
SessionID Vi,
const char *memChan,
int ValBufferSize,
char Val[]
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MemChan: The name of the memory channel to get the archive from.

ValBufferSize:

Val: XML document containing settings and readings.

Returns

Success (0) or error code.

PwrSnsr_GetMesial()

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

```
EXPORT int PwrSnsr_GetMesial
(
SessionID Vi,
const char * Channel,
float* Mesial
```

```
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Mesial:

Returns

Success (0) or error code.

PwrSnsr_GetMinFreqHighBandwidth()

Minimum frequency of RF the sensor can measure in high bandwidth.

```
EXPORT int PwrSnsr_GetMinFreqHighBandwidth
(
SessionID Vi,
const char *Channel,
float *MinFreqHighBandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MinFreqHighBandwidth:

Returns

Success (0) or error code.

PwrSnsr_GetMinFreqLowBandwidth()

Minimum frequency carrier the sensor can measure in low bandwidth.

```
EXPORT int PwrSnsr_GetMinFreqLowBandwidth
(
SessionID Vi,
const char *Channel,
float *MinFreqLowBandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MinFreqLowBandwidth:

Returns

Success (0) or error code.

PwrSnsr_GetMinimumSupportedFirmware()

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

```
EXPORT int PwrSnsr_Get MinimumSupportedFirmware
(
int *Version
)
```

Parameters

Version

The version of the software as an integer.

Returns

Success (0) or error code.

PwrSnsr_GetMinimumTrig()

Minimum internal trigger level in dBm.

```
EXPORT int PwrSnsr_GetMinimumTrig
(
SessionID Vi,
const char * Channel,
float* MinimumTrig
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MinimumTrig:

Returns

Success (0) or error code.

PwrSnsr_GetMinMeasurements()

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

```
EXPORT int PwrSnsr_GetMinMeasurements
(
```

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

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetModel()

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

```
EXPORT int PwrSnsr_GetModel
(
SessionID Vi,
const char * Channel,
int ModelBufferSize,
char Model[]
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetNumberOfCals()

Get the number of calibrations left on the license.

```
EXPORT int PwrSnsr_Get NumberOfCals
(
long *val)
```

Parameters

Val: Number of cals left.

Returns

Success (0) or error code.

PwrSnsr_GetOffsetdB()

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.

```
EXPORT int PwrSnsr_GetOffsetdB
(
SessionID Vi,
const char * Channel,
float* OffsetdB
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

OffsetdB:

Returns

Success (0) or error code.

PwrSnsr_GetOverRan()

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

```
EXPORT int PwrSnsr_GetOverRan
(
SessionID Vi,
int *OverRan
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

OverRan: True if the power meter's internal buffer filled up before being emptied.

Returns

Success (0) or error code.

PwrSnsr_GetPeakHoldDecay()

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

```
EXPORT int PwrSnsr_GetPeakHoldDecay
(
SessionID Vi,
const char *Channel,
int *EnvelopeAverage
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

EnvelopeAverage: Out parameter value.

Returns

Success (0) or error code.

PwrSnsr_GetPeakHoldTracking()

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.

```
EXPORT int PwrSnsr_GetPeakHoldTracking
(
SessionID Vi,
const char *Channel,
int *EnvelopeTracking
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetPeakPowerMax()

Maximum power level the sensor can measure.

```
EXPORT int PwrSnsr_GetPeakPowerMax
(
SessionID Vi,
```

```
const char *Channel,
float *PeakPowerMax
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PeakPowerMax:

Returns

Success (0) or error code.

PwrSnsr_GetPeakPowerMin()

Minimum power level the sensor can measure.

```
EXPORT int PwrSnsr_GetPeakPowerMin
(
SessionID Vi,
const char *Channel,
float *PeakPowerMin
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PeakPowerMin:

Returns

Success (0) or error code.

PwrSnsr_GetPercentPosition()

Get the cursor percent on the CCDF plot.

```
EXPORT int PwrSnsr_GetPercentPosition
(
SessionID Vi,
const char *Channel,
double *PercentPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PercentPosition:

Returns

Success (0) or error code.

PwrSnsr_GetPeriod()

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

```
EXPORT int PwrSnsr_GetPeriod
(
SessionID Vi,
float *Period
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetPowerPosition()

Get the cursor power in dB on the CCDF plot.

```
EXPORT int PwrSnsr_GetPowerPosition
(
SessionID Vi,
const char *Channel,
double *PowerPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PowerPosition:

Returns

Success (0) or error code.

PwrSnsr_GetProximal()

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

```
EXPORT int PwrSnsr_GetProximal
```

```
(
SessionID Vi,
const char *Channel,
float *Proximal
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Proximal:

Returns

Success (0) or error code.

PwrSnsr_GetPulseUnits()

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

```
EXPORT int PwrSnsr_GetPulseUnits
(
SessionID Vi,
const char *Channel,
PwrSnsrPulseUnitsEnum *Units
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PwrSnsrPulseUnitsEnum:

Returns

Success (0) or error code.

PwrSnsr_GetRdgsEnableFlag()

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

```
EXPORT int PwrSnsr_GetRdgsEnableFlag
(
SessionID Vi,
int *Flag
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Flag: Bit masked value indicating which measurement arrays will be queried (see PwrSnsrRdgsEnableFlag).

Returns

Success (0) or error code.

PwrSnsr_GetReadingPeriod()

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

```
EXPORT int PwrSnsr_GetReadingPeriod
(
SessionID Vi,
const char *Channel,
float *ReadingPeriod
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetReturnCount()

Get the return count for each measurement query.

```
EXPORT int PwrSnsr_GetReturnCount
(
SessionID Vi,
int *ReturnCount
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetSequenceNumbers()

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

```
EXPORT int PwrSnsr_GetSequenceNumbers
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
long long Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetSerialNumber()

Gets the serial number of the sensor.

```
EXPORT int PwrSnsr_GetSerialNumber
(
SessionID Vi,
const char *Channel,
int SerialNumberBufferSize,
char SerialNumber[]
)
```

Parameters

Vi: The SessionID handle returned by 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 number.

Returns

Success (0) or error code.

PwrSnsr_GetSessionCount()

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

```
EXPORT int PwrSnsr_GetSessionCount
(
SessionID Vi,
int *SessionCount
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetSlaveSkew()

Gets the skew in seconds for the slave trigger.

```
EXPORT int PwrSnsr_GetSlaveSkew
(
SessionID Vi,
const char *Channel,
float *SlaveSkew
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetStartDelay()

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

```
EXPORT int PwrSnsr_GetStartDelay
(
SessionID Vi,
float *StartDelay
```

```
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetStartGate()

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

```
EXPORT int PwrSnsr_GetStartGate
(
SessionID Vi,
const char *Channel,
float *StartGate
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

StartGate:

Returns

Success (0) or error code.

PwrSnsr_GetStartMode()

Get the mode used to start acquisition of buffer entries.

```
EXPORT int PwrSnsr_GetStartMode
(
SessionID Vi,
PwrSnsrMeasBuffStartModeEnum *StartMode
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetStartQual()

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

```
EXPORT int PwrSnsr_GetStartQual
(
SessionID Vi,
float *StartQual
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetStartTimes()

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

```
EXPORT int PwrSnsr_GetStartTimes
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
double Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetSweepTime()

Get sweep time for the trace in seconds.

```
EXPORT int PwrSnsr_GetSweepTime
```

```
(
SessionID Vi,
const char *Channel,
float *SweepTime
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetTempComp()

Get the state of the peak sensor temperature compensation system.

```
EXPORT int PwrSnsr_GetTempComp
(
SessionID Vi,
const char * Channel,
int* TempComp
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetTermAction()

Get the termination action for statistical capturing.

```
EXPORT int PwrSnsr_GetTermAction
(
SessionID Vi,
const char *Channel,
PwrSnsrTermActionEnum *TermAction
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermAction:

Returns

Success (0) or error code.

PwrSnsr_GetTermCount()

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

```
EXPORT int PwrSnsr_GetTermCount
(
SessionID Vi,
const char *Channel,
double *TermCount
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermCount:

Returns

Success (0) or error code.

PwrSnsr_GetTermTime()

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

```
EXPORT int PwrSnsr_GetTermTime
(
SessionID Vi,
const char * Channel,
int * TermTime
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermTime:

Returns

Success (0) or error code.

PwrSnsr_GetTimebase()

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.

```
EXPORT int PwrSnsr_GetTimebase
(
SessionID Vi,
float *Timebase
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Timebase:

Returns

Success (0) or error code.

PwrSnsr_GetTimedOut()

Check if the last measurement buffer session timed out.

```
EXPORT int PwrSnsr_GetTimedOut
(
SessionID Vi,
int *TimedOut
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

TimedOut: True if the last measurement buffer session timed out.

Returns

Success (0) or error code.

PwrSnsr_GetTimeOut()

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

```
EXPORT int PwrSnsr_GetTimeOut
(
SessionID Vi,
long *Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Val: Time out in milliseconds. -1 denote infinite time out.

Returns

Success (0) or error code.

PwrSnsr_GetTimePerPoint()

Get time spacing for each waveform point in seconds.

```
EXPORT int PwrSnsr_GetTimePerPoint
(
SessionID Vi,
const char *Channel,
float *TimePerPoint
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TimePerPoint: Time spacing for each waveform point in seconds.

Returns

Success (0) or error code.

PwrSnsr_GetTimespan()

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.

```
EXPORT int PwrSnsr_GetTimespan
(
SessionID Vi,
float *Timespan
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Timespan:

Returns

Success (0) or error code.

PwrSnsr_GetTraceStartTime()

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

```
EXPORT int PwrSnsr_GetTraceStartTime
(
SessionID Vi,
const char *Channel,
float *TraceStartTime
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TraceStartTime: Time offset (start time) of the trace in seconds. May be negative, indicating pretrigger information.

Returns

Success (0) or error code.

PwrSnsr_GetTrigDelay()

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.

```
EXPORT int PwrSnsr_GetTrigDelay
(
SessionID Vi,
float *Delay
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Delay:

Returns

Success (0) or error code.

PwrSnsr_GetTrigHoldoff()

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.

```
EXPORT int PwrSnsr_GetTrigHoldoff
(
SessionID Vi,
float *Holdoff
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Holdoff:

Returns

Success (0) or error code.

PwrSnsr_GetTrigHoldoffMode()

Returns the holdoff mode to normal or gap holdoff.

```
EXPORT int PwrSnsr_GetTrigHoldoffMode
(
SessionID Vi,
PwrSnsrHoldoffMode Enum *HoldoffMode
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

HoldoffMode: The hold-off mode.

Returns

Success (0) or error code.

PwrSnsr_GetTrigLevel()

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.

```
EXPORT int PwrSnsr_GetTrigLevel
(
SessionID Vi,
```

```
float *Level
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Level: Trigger level in dBm.

Returns

Success (0) or error code.

PwrSnsr_GetTrigMode()

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

```
EXPORT int PwrSnsr_GetTrigMode
(
SessionID Vi,
PwrSnsrTriggerMod_eEnum *Mode
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Mode: Trigger mode.

Returns

Success (0) or error code.

PwrSnsr_GetTrigPosition()

Return the position of the trigger event on displayed sweep.

```
EXPORT int PwrSnsr_GetTrigPosition
(
SessionID Vi,
PwrSnsrTriggerPositionEnum *Position
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Position: Trigger position.

Returns

Success (0) or error code.

PwrSnsr_GetTrigSlope()

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.

```
EXPORT int PwrSnsr_GetTrigSlope
(
SessionID Vi,_
PwrSnsrTriggerSlop_eEnum *Slope
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Slope:

Returns

Success (0) or error code.

PwrSnsr_GetTrigSource()

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

```
EXPORT int PwrSnsr_GetTrigSource
(
SessionID Vi,
PwrSnsrTriggerSourceEnum *Source
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Source:

Returns

Success (0) or error code.

PwrSnsr_GetTrigStatus()

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

```
EXPORT int PwrSnsr_GetTrigStatus
(
SessionID Vi,
PwrSnsrTriggerStatusEnum *Status
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Status: Status of the trigger.

Returns

Success (0) or error code.

PwrSnsr_GetTrigVernier()

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

```
EXPORT int PwrSnsr_GetTrigVernier
(
SessionID Vi,
float *Vernier
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetUnits()

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

```
EXPORT int PwrSnsr_GetUnits
(
SessionID Vi,
const char *Channel,
PwrSnsrUnitsEnum *Units
)
```

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Units:

Returns

Success (0) or error code.

PwrSnsr_GetVerticalCenter()

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

```
EXPORT int PwrSnsr_GetVerticalCenter
(
SessionID Vi,
const char * Channel,
float * VerticalCenter
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetVerticalScale()

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

```
EXPORT int PwrSnsr_GetVerticalScale
(
SessionID Vi,
const char * Channel,
float * VerticalScale
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_GetWriteProtection()

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

```
EXPORT int PwrSnsr_GetWriteProtection
(
SessionID Vi,
int *WriteProtection
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_init()

Initialize a communication session with a supported USB power sensor.

```
EXPORT int PwrSnsr_init
(
char *ResourceName,
SessionID *Vi
)
```

Parameters

ResourceName: Name of the resource. The resource descriptor is in the following format: USB::[VID]::[PID]::[Serial Number]::BTN

Where serial number is the USB power meter's serial number in decimal format, and the VID and PID are in hexidecimal format.

For example, for serial number 1234, VID of 0x1bfe and PID of 0x5500: USB::0x1BFE::0x5500::1234::BTN. Multiple-channel synthetic meters can be defined by combining multiple descriptors separated by semicolons.

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.

For example, to define a synthetic peak-power meter using serial number 1234 for CH1 and serial number 4242 for CH2: USB::0x1BFE::0x5500::1234::BTN;USB::0x1B FE::0x5500::4242::BTN

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_InitiateAquisition()

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

```
EXPORT int PwrSnsr_InitiateAquisition
(
SessionID Vi
```

```
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_IsLicenseDongleConnected()

Get whether the hardware license dongle is connected.

```
EXPORT int PwrSnsr_IsLic enseDongleConnected
(
int *val
)
```

Parameters

Val: Boolean. 1 for connected or 0 for not connected.

Returns

Success (0) or error code.

PwrSnsr_LoadMemChanFromArchive()

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

```
EXPORT int PwrSnsr_LoadMemChanFromArchive
(
SessionID Vi,
const char *memChan,
const char *ArchiveContent
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_MeasurePower()

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

```
EXPORT int PwrSnsr_MeasurePower
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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: Average power in dBm

Returns

Success (0) or error code.

PwrSnsr_MeasureVoltage()

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

```
EXPORT int PwrSnsr_MeasureVoltage
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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: Average voltage in linear volts.

Returns

Success (0) or error code.

PwrSnsr_QueryAverageMeasurements()

Query the power meter for all buffered average power measurements.

```
EXPORT int PwrSnsr_QueryAverageMeasurements
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_QueryDurations()

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

```
EXPORT int PwrSnsr_QueryDurations
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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 buffered measurement durations.

ValActualSize: Actual size of the returned array in elements.

Returns

Success (0) or error code.

PwrSnsr_QueryMaxMeasurements()

Query the power meter for all buffered maximum power measurements.

```
EXPORT int PwrSnsr_QueryMaxMeasurements
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_QueryMinMeasurements()

Query the power meter for all buffered minimum power measurements.

```
EXPORT int PwrSnsr_QueryMinMeasurements
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
float Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_QuerySequenceNumbers()

Query the power meter for all buffered sequence numbers.

```
EXPORT int PwrSnsr_QuerySequenceNumbers
(
SessionID Vi,
const char *Channel,
int ValBufferSize,
long long Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_QueryStartTimes()

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

```
EXPORT int PwrSnsr_QueryStartTimes
(
SessionID Vi,
const char * Channel,
int ValBufferSize,
float Val[],
int* ValActualSize
);
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_ReadArrayMarkerPower()

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

```
EXPORT int PwrSnsr_ReadArrayMarkerPower
(
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
)
```

Parameters

Vi: The SessionID handle returned by 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 the markers.

MaxPowerCondCode: Condition code.

MinPower: Minimum power between the markers.

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.

PwrSnsr_ReadByteArray()

Reads byte array from the meter.

```
EXPORT int PwrSnsr_ReadByteArray
(
SessionID Vi,
const char *Channel,
int Count,
int ValBufferSize,
unsigned char Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_ReadControl()

Reads a control transfer on the USB.

```
EXPORT int PwrSnsr_ReadControl
(
SessionID Vi,
const char *Channel,
```

```
int Count,
int ValBufferSize,
unsigned char Val[],
int *ValActualSize
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_ReadCWArray()

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'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.

```
EXPORT int PwrSnsr_ReadCWArray
(
SessionID Vi,
const char *Channel,
float *PeakAverage,
PwrSnsrCondCodeEnum *PeakAverageValid,
float *PeakMax,
PwrSnsrCondCodeEnum *PeakMaxValid,
float *PeakMin,
PwrSnsrCondCodeEnum *PeakMinValid,
float *PeakToAvgRatio,
PwrSnsrCondCodeEnum *PeakToAvgRatioValid
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

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.

PwrSnsr_ReadCWPower()

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

```
EXPORT int PwrSnsr_ReadCWPower
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *IsValid,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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 value.

Returns

Success (0) or error code.

PwrSnsr_ReadDutyCycle()

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

```
EXPORT int PwrSnsr_ReadDutyCycle
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadEdgeDelay()

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

```
EXPORT int PwrSnsr_ReadEdgeDelay
(
SessionID Vi,
const char * Channel,
PwrSnsrCondCodeEnum * CondCode,
float* Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadFallTime()

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

```
EXPORT int PwrSnsr_ReadFallTime
(
SessionID Vi,
const char * Channel,
PwrSnsrCondCodeEnum * CondCode,
float* Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIEEEBottom()

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.

```
EXPORT int PwrSnsr_ReadIEEEBottom
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIEEETop()

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

```
EXPORT int PwrSnsr_ReadIEEETop
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalAvg()

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.

```
EXPORT int PwrSnsr_ReadIntervalAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalFilteredMax()

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.

```
EXPORT int PwrSnsr_ReadIntervalFilteredMax
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalFilteredMin()

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.

```
EXPORT int PwrSnsr_ReadIntervalFilteredMin
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalMax()

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.

```
EXPORT int PwrSnsr_ReadIntervalMax
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalMaxAvg()

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

```
EXPORT int PwrSnsr_ReadIntervalMaxAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalMin()

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.

```
EXPORT int PwrSnsr_ReadIntervalMin
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalMinAvg()

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

```
EXPORT int PwrSnsr_ReadIntervalMinAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadIntervalPkToAvg()

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.

```
EXPORT int PwrSnsr_ReadIntervalPkToAvg
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerAverage()

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

```
EXPORT int PwrSnsr_ReadMarkerAverage
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerDelta()

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

```
EXPORT int PwrSnsr_ReadMarkerDelta
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerMax()

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

```
EXPORT int PwrSnsr_ReadMarkerMax
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerMin()

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

```
EXPORT int PwrSnsr_ReadMarkerMin
(
SessionID Vi,
const char *Channel,
int Marker,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerRatio()

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

```
EXPORT int PwrSnsr_ReadMarkerRatio
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerRDelta()

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

```
EXPORT int PwrSnsr_ReadMarkerRDelta
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadMarkerRRatio()

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

```
EXPORT int PwrSnsr_ReadMarkerRRatio
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadOfftime()

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

```
EXPORT int PwrSnsr_ReadOfftime
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadOvershoot()

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

```
EXPORT int PwrSnsr_ReadOvershoot
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadPeriod()

Returns the interval between two successive pulses.

```
EXPORT int PwrSnsr_ReadPeriod
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadPowerArray()

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.

```
EXPORT int PwrSnsr_ReadPowerArray
(
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
)
```

Parameters

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.

PwrSnsr_ReadPRF()

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

```
EXPORT int PwrSnsr_ReadPRF
(
SessionID Vi,
const char * Channel,
PwrSnsrCondCodeEnum * CondCode,
float* Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadPulseCycleAvg()

Returns the average power of the entire pulse.

```
EXPORT int PwrSnsr_ReadPulseCycleAvg
(
SessionID Vi,
const char * Channel,
PwrSnsrCondCodeEnum * CondCode,
float* Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadPulseOnAverage()

Average power of the ON portion of the pulse.

```
EXPORT int PwrSnsr_ReadPulseOnAverage
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadPulsePeak()

Returns the peak amplitude during the pulse.

```
EXPORT int PwrSnsr_ReadPulsePeak
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadRiseTime()

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

```
EXPORT int PwrSnsr_ReadRiseTime
(
SessionID Vi,
const char *Channel,
PwrSnsrCondCodeEnum *CondCode,
float *Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_ReadSCPI()

Read a SCPI string response from the instrument.

```
EXPORT int PwrSnsr_ReadSCPIBytes
(
SessionID Vi,
int ValueBufferSize,
long* ValueActualSize,
char Value[],
int Timeout
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

ValueBufferSize: Number of elements in Value.

ValueActualSize: Number of elements actually written to Value.

Value: 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.

PwrSnsr_ReadSCPIBytes()

Read a SCPI byte array response from the instrument.

```
EXPORT int PwrSnsr_ReadSCPIBytes
(
SessionID Vi,
int ValueBufferSize,
char Value[],
long *ValueActualSize,
int Timeout
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

ValueBufferSize: Number of elements in Value.

Value: The byte array returned from the instrument SCPI interface.

ValueActualSize:

Timeout: Time out in milliseconds for the read operation. Use -1 for infinite and -2 to use the existing time out value. 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.

PwrSnsr_ReadSCPIFromNamedParser()

Read a SCPI string response from the instrument.

```
EXPORT int PwrSnsr_ReadSCPIFromNamedParser
(
SessionID Vi,
const char *name,
int ValueBufferSize,
long *ValueActualSize,
char Value[],
int Timeout
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

name: Name of the parser. If parser doesn't exist, returns PWR_SNSR_ERROR_NULL_POINTER.

PwrSnsr_SendSCPIToNamedParser can be used to create a named parser.

ValueBufferSize: Number of elements in Value.

ValueActualSize: Number of elements actually written to Value.

Value: 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.

PwrSnsr_ReadTimeArray()

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 For those commands. Units are the same as the channel's units.

```
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
)
```

Parameters

Vi: The SessionID handle returned by 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.

OfftimeValid: 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.

PwrSnsr_ReadWaveform()

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.

```
EXPORT int PwrSnsr_ReadWaveform
(
SessionID Vi,
const char *Channel,
int WaveformArrayBufferSize,
```

```
float WaveformArray[],
int *WaveformArrayActualSize
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

WaveformArrayBuffer Size: 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.

WaveformArrayActual Size: Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

PwrSnsr_ReadWaveformMinMax()

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.

```
EXPORT int PwrSnsr_ReadWaveformMinMax
(
SessionID Vi,
const char *Channel,
int MinWaveformBufferSize,
float MinWaveform[],
int *MinWaveformActualSize
int MaxWaveformBufferSize,
float MaxWaveform[],
int *MaxWaveformActualSize,
int WaveformArrayBufferSize,
float WaveformArray[],
int *WaveformArrayActualSize
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

MinWaveformBuffer Size: 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: This 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.

PwrSnsr_ReadWidth()

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

```
EXPORT int PwrSnsr_ReadWidth
(
SessionID Vi,
const char * Channel,
PwrSnsrCondCodeEnum * CondCode,
float* Val
)
```

Parameters

Vi: The SessionID handle returned by 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. Condition code.

Val: Measurement value.

Returns

Success (0) or error code.

PwrSnsr_reset()

Places the instrument in a known state.

```
EXPORT int PwrSnsr_reset
(
```

```
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_ResetContinuousCapture()

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.

```
EXPORT int PwrSnsr_Res etContinuousCapture
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_SaveToMemoryChannel()

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

```
EXPORT int PwrSnsr_SaveToMemoryChannel
(
SessionID Vi,
const char * MemChan,
const char * ChannelName
)
```

Parameters

Vi: The SessionID handle returned by 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."

Channel: The channel name to copy from.

Returns

Success (0) or error code.

PwrSnsr_SaveUserCal()

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

```
EXPORT int PwrSnsr_SaveUserCal
(
SessionID Vi,
const char *Channel
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_self_test()

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.

```
EXPORT int PwrSnsr_self_test
(
SessionID Vi,
int *TestResult
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

TestResult: Error or success code.

Returns

Success (0) or error code.

PwrSnsr_SendSCPIBytes()

Send a SCPI command as a byte array.

```
EXPORT int PwrSnsr_SendSCPIBytes
(
SessionID Vi,
int CommandBufferSize,
char Command[]
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SendSCPICommand()

Send a SCPI command to the instrument.

```
EXPORT int PwrSnsr_SendSCPICommand
(
SessionID Vi,
const char *Command
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Command:

Returns

Success (0) or error code.

PwrSnsr_SendSCPIToNamedParser()

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

```
EXPORT int PwrSnsr_SendSCPIToNamedParser
(
SessionID Vi, const char *name,
const char *Command
)
```

Parameters

Vi: The SessionID handle returned by 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:

Returns

Success (0) or error code.

PwrSnsr_SetAverage()

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

```
EXPORT int PwrSnsr_SetAverage
(
SessionID Vi,
const char * Channel,
int Average
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Average:

Returns

Success (0) or error code.

PwrSnsr_SetBandwidth()

Set the sensor video bandwidth for the selected sensor.

```
EXPORT int PwrSnsr_SetBandwidth
(
SessionID Vi,
const char *Channel,
PwrSnsrBandwidthEnum Bandwidth
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Bandwidth :

Returns

Success (0) or error code.

PwrSnsr_SetCalFactor()

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

```
EXPORT int PwrSnsr_SetCalFactor
(
SessionID Vi,
const char *Channel,
float CalFactor
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

CalFactor:

Returns

Success (0) or error code.

PwrSnsr_SetCapture()

Set whether statistical capture is enabled.

```
EXPORT int PwrSnsr_SetCapture
(
SessionID Vi,
const char *Channel,
int Capture
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Capture:

Returns

Success (0) or error code.

PwrSnsr_SetCCDFTraceCount()

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

```
EXPORT int PwrSnsr_SetCCDFTraceCount
(
SessionID Vi,
const char *Channel,
int TraceCount
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TraceCount:

Returns

Success (0) or error code.

PwrSnsr_SetContinuousCapture()

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

```
EXPORT int PwrSnsr_SetContinuousCapture
(
SessionID Vi,
int ContinuousCapture
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetDistal()

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

```
EXPORT int PwrSnsr_SetDistal
(
SessionID Vi,
const char * Channel,
float Distal
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Distal:

Returns

Success (0) or error code.

PwrSnsr_SetDuration()

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

```
EXPORT int PwrSnsr_SetDuration
(
```

```
SessionID Vi,
float Duration
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Duration: The duration samples are captured during each timed mode acquisition in seconds.

Returns

Success (0) or error code.

PwrSnsr_SetEnabled()

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.

```
EXPORT int PwrSnsr_SetEnabled
(
SessionID Vi,
const char *Channel,
int Enabled
)
```

Parameters

Vi: The SessionID handle returned by 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 enable; 0 for disable.

Returns

Success (0) or error code.

PwrSnsr_SetEndDelay()

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.

```
EXPORT int PwrSnsr_SetEndDelay
(
SessionID Vi,
float EndDelay
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

EndDelay: Delay time added to the detected end of a burst for analysis.

Returns

Success (0) or error code.

PwrSnsr_SetEndGate()

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

```
EXPORT int PwrSnsr_SetEndGate
(
SessionID Vi,
const char * Channel,
float EndGate
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

EndGate:

Returns

Success (0) or error code.

PwrSnsr_SetEndQual()

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

```
EXPORT int PwrSnsr_SetEndQual
(
SessionID Vi,
float EndQual
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetExternalSkew()

Sets the skew in seconds for the external trigger.

```
EXPORT int PwrSnsr_SetExternalSkew
(
SessionID Vi,
```

```
const char *Channel,
float External
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetFetchLatency()

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

```
EXPORT int PwrSnsr_SetFetchLatency
(
SessionID Vi,
int Latency
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Latency: Fetch latency in ms.

Returns

Success (0) or error code.

PwrSnsr_SetFilterState()

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

```
EXPORT int PwrSnsr_SetFilterState
(
SessionID Vi,
const char *Channel,
PwrSnsrFilterStateEnum FilterState
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

FilterState:

Returns

Success (0) or error code.

PwrSnsr_SetFilterTime()

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

```
EXPORT int PwrSnsr_SetFilterTime
(
SessionID Vi,
const char *Channel,
float FilterTime
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

FilterTime:

Returns

Success (0) or error code.

PwrSnsr_SetFrequency()

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

```
EXPORT int PwrSnsr_SetFrequency
(
SessionID Vi,
const char *Channel,
float Frequency
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetGateMode()

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

```
EXPORT int PwrSnsr_SetGateMode
```

```
(
SessionID Vi,
PwrSnsrMeasBuffGateEnum GateMode
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetGating()

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

```
EXPORT int PwrSnsr_SetGating
(
SessionID Vi,
const char *Channel,
PwrSnsrStatGatingEnum Gating
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Gating:

Returns

Success (0) or error code.

PwrSnsr_SetHorizontalOffset()

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).

```
EXPORT int PwrSnsr_SetHorizontalOffset
(
SessionID Vi,
const char *Channel,
double HorizontalOffset
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

HorizontalOffset:

Returns

Success (0) or error code.

PwrSnsr_SetHorizontalScale()

Set the statistical mode horizontal scale in dB/Div.

```
EXPORT int PwrSnsr_SetHorizontalScale
(
SessionID Vi,
const char *Channel,
double HorizontalScale
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

HorizontalScale:

Returns

Success (0) or error code.

PwrSnsr_SetInitiateContinuous()

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.

```
EXPORT int PwrSnsr_SetInitiateContinuous
(
SessionID Vi,
int InitiateContinuous
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetInternalSkew()

Sets the skew in seconds for the internal trigger.

```
EXPORT int PwrSnsr_SetInternalSkew
(
SessionID Vi,
const char *Channel,
float InternalSkew
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetMarkerPixelPosition()

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

```
EXPORT int PwrSnsr_SetMarkerPixelPosition
(
SessionID Vi,
int MarkerNumber,
int PixelPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber:

PixelPosition:

Returns

Success (0) or error code.

PwrSnsr_SetMarkerTimePosition()

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

Note that 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.

```
EXPORT int PwrSnsr_SetMarkerTimePosition
```

```
(
SessionID Vi,
int MarkerNumber,
float TimePosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber:

TimePosition:

Returns

Success (0) or error code.

PwrSnsr_SetMeasBuffEnabled()

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

```
EXPORT int PwrSnsr_SetMeasBuffEnabled
(
SessionID Vi,
int MeasBuffEnabled
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetMesial()

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

```
EXPORT int PwrSnsr_SetMesial
(
SessionID Vi,
const char * Channel,
float Mesial
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Mesial:

Returns

Success (0) or error code.

PwrSnsr_SetOffsetdB()

Set 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.

```
EXPORT int PwrSnsr_SetOffsetdB
(
SessionID Vi,
const char * Channel,
float OffsetdB
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

OffsetdB:

Returns

Success (0) or error code.

PwrSnsr_SetPeakHoldDecay()

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

```
EXPORT int PwrSnsr_SetPeakHoldDecay
(
SessionID Vi,
const char *Channel,
int PeakHoldDecay
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

EnvelopeAverage: Peak hold decay value.

Returns

Success (0) or error code.

PwrSnsr_SetPeakHoldTracking()

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.

```
EXPORT int PwrSnsr_SetPeakHoldTracking
(
SessionID Vi,
const char *Channel,
int EnvelopeTracking
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

EnvelopeTracking: Boolean value. Use True to set peak hold tracking on.

Returns

Success (0) or error code.

PwrSnsr_SetPercentPosition()

Set the cursor percent on the CCDF plot.

```
EXPORT int PwrSnsr_SetPercentPosition
(
SessionID Vi,
const char *Channel,
double PercentPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PercentPosition:

Returns

Success (0) or error code.

PwrSnsr_SetPeriod()

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

```
EXPORT int PwrSnsr_SetPeriod
```

```
(
SessionID Vi,
float Period
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetPowerPosition()

Set the cursor power in dB on the CCDF plot.

```
EXPORT int PwrSnsr_SetPowerPosition
(
SessionID Vi,
const char *Channel,
double PowerPosition
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PowerPosition:

Returns

Success (0) or error code.

PwrSnsr_SetProximal()

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

```
EXPORT int PwrSnsr_SetProximal
(
SessionID Vi,
const char * Channel,
float Proximal
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Proximal:

Returns

Success (0) or error code.

PwrSnsr_SetPulseUnits()

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

```
EXPORT int PwrSnsr_SetPulseUnits
(
SessionID Vi,
const char *Channel,
PwrSnsrPulseUnitsEnum PwrSnsrPulseUnitsEnum
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

PwrSnsrPulseUnitsEnum:

Returns

Success (0) or error code.

PwrSnsr_SetRdgsEnableFlag()

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

```
EXPORT int PwrSnsr_SetRdgsEnableFlag
(
SessionID Vi,
int Flag
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Flag: Bit masked value indicating which measurement arrays will be queried (see PwrSnsrRdgsEnableFlag).

Returns

Success (0) or error code.

PwrSnsr_SetReturnCount()

Set the return count for each measurement query.

```
EXPORT int PwrSnsr_SetReturnCount
(
SessionID Vi,
int ReturnCount
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetSessionCount()

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

```
EXPORT int PwrSnsr_SetSessionCount
(
SessionID Vi,
int SessionCount
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetSessionTimeout()

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

```
EXPORT int PwrSnsr_SetSessionTimeout
(
SessionID Vi,
float Seconds
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetSlaveSkew()

Sets the skew in seconds for the slave trigger.

```
EXPORT int PwrSnsr_SetSlaveSkew
(
SessionID Vi,
const char *Channel,
float SlaveSkew
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetStartDelay()

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

```
EXPORT int PwrSnsr_SetStartDelay
(
SessionID Vi,
float StartDelay
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetStartGate()

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

```
EXPORT int PwrSnsr_SetStartGate
(
SessionID Vi,
const char *Channel,
float StartGate
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

StartGate:

Returns

Success (0) or error code.

PwrSnsr_SetStartMode()

Set the mode used to start acquisition of buffer entries.

```
EXPORT int PwrSnsr_SetStartMode
(
SessionID Vi,
PwrSnsrMeasBuffStartModeEnum StartMode
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetStartQual()

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

```
EXPORT int PwrSnsr_SetStartQual
(
SessionID Vi,
float StartQual
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetTempComp()

Set the state of the peak sensor temperature compensation system.

```
EXPORT int PwrSnsr_SetTempComp
(
SessionID Vi,
const char *Channel,
int TempComp
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetTermAction()

Set the termination action for statistical capturing.

```
EXPORT int PwrSnsr_SetTermAction
(
SessionID Vi,
const char *Channel,
PwrSnsrTermActionEnum TermAction
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermAction:

Returns

Success (0) or error code.

PwrSnsr_SetTermCount()

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

```
EXPORT int PwrSnsr_SetTermCount
(
SessionID Vi,
const char *Channel,
double TermCount
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermCount:

Returns

Success (0) or error code.

PwrSnsr_SetTermTime()

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

```
EXPORT int PwrSnsr_SetTermTime
(
SessionID Vi,
const char * Channel,
int TermTime
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

TermTime:

Returns

Success (0) or error code.

PwrSnsr_SetTimebase()

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.

```
EXPORT int PwrSnsr_SetTimebase
(
```

```
SessionID Vi,
float Timebase
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Timebase:

Returns

Success (0) or error code.

PwrSnsr_SetTimeOut()

Sets the time out in milliseconds for I/O.

```
EXPORT int PwrSnsr_SetTimeOut
(
SessionID Vi,
long Milliseconds
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

MilliSeconds: Time out in milliseconds. Use -1 for infinite time out.

Returns

Success (0) or error code.

PwrSnsr_SetTimespan()

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.

```
EXPORT int PwrSnsr_SetTimespan
(
SessionID Vi,
float Timespan
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Timespan:

Returns

Success (0) or error code.

PwSrnsr_SetTrigDelay()

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.

```
EXPORT int PwrSnsr_SetTrigDelay
(
SessionID Vi,
float Delay
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Delay:

Returns

Success (0) or error code.

PwrSnsr_SetTrigHoldoff()

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.

```
EXPORT int PwrSnsr_SetTrigHoldoff
(
SessionID Vi,
float Holdoff
)
```

Parameters

Vi

The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Holdoff:

Returns

Success (0) or error code.

PwrSnsr_SetTrigHoldoffMode()

Sets the holdoff mode to normal or gap holdoff.

```
EXPORT int PwrSnsr_SetTrigHoldoffMode
(
SessionID Vi,
PwrSnsrHoldoffModeEnum HoldoffMode
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

HoldoffMode: Holdoff mode.

Returns

Success (0) or error code.

PwrSnsr_SetTrigLevel()

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.

```
EXPORT int PwrSnsr_SetTrigLevel
(
SessionID Vi,
float Level
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Level: Trigger level in dBm.

Returns

Success (0) or error code.

PwrSnsr_SetTrigMode()

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

```
EXPORT int PwrSnsr_SetTrigMode
(
SessionID Vi,_
PwrSnsrTriggerMod_eEnum Mode
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Mode: Trigger mode.

Returns

Success (0) or error code.

PwrSnsr_SetTrigOutMode()

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

```
EXPORT int PwrSnsr_SetTrigOutMode
(
SessionID Vi,
const char * Channel,
int Mode
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetTrigPosition()

Set the position of the trigger event on displayed sweep.

```
EXPORT int PwrSnsr_SetTrigPosition
(
SessionID Vi,
PwrSnsrTriggerPositionEnum Position
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Position: Trigger position.

Returns

Success (0) or error code.

PwrSnsr_SetTrigSlope()

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.

```
EXPORT int PwrSnsr_SetTrigSlope
(
SessionID Vi,_
PwrSnsrTriggerSlopeEnum Slope
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Slope:

Returns

Success (0) or error code.

PwrSnsr_SetTrigSource()

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

```
EXPORT int PwrSnsr_SetTrigSource
(
SessionID Vi,
PwrSnsrTriggerSourceEnum Source
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Source:

Returns

Success (0) or error code.

PwrSnsr_SetTrigVernier()

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

```
EXPORT int PwrSnsr_SetTrigVernier
(
SessionID Vi,
float Vernier
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetUnits()

Set 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

```
EXPORT int PwrSnsr_SetUnits
(
SessionID Vi,
const char *Channel,
PwrSnsrUnitsEnum Units
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

Units:

Returns

Success (0) or error code.

PwrSnsr_SetVerticalCenter()

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

```
int EXPORT PwrSnsr_SetVerticalCenter
(
SessionID Vi,
const char * Channel,
float VerticalCenter
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_SetVerticalScale()

Sets vertical scale based on current units: <arg> = (range varies by units).

```
int EXPORT PwrSnsr_SetVerticalScale
(
SessionID Vi,
const char * Channel,
float VerticalScale
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel: Channel number. For single instruments, set this to "CH1."

VerticalCenter: Vertical scale in units.

Returns

Success (0) or error code.

PwrSnsr_SetWriteProtection()

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

```
EXPORT int PwrSnsr_SetWriteProtection
(
SessionID Vi,
int WriteProtection
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_StartAcquisition()

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.

```
EXPORT int PwrSnsr_StartAcquisition
(
SessionID Vi
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_StatModeReset()

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

```
EXPORT int PwrSnsr_StatModeReset
(
SessionID Vi,
const char *Channel
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_Status()

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

```
EXPORT int PwrSnsr_Status
(
SessionID Vi,
PwrSnsrAcquisitionStatusEnum * Val
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Val: Status out parameter.

Returns

Success (0) or error code.

PwrSnsr_StopAcquisition()

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

```
EXPORT int PwrSnsr_StopAcquisition
(
SessionID Vi
```

```
)
```

Parameters

Vi: The SessionID handle returned by the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

PwrSnsr_Write()

Write a byte array to the meter.

```
EXPORT int PwrSnsr_Write
(
SessionID Vi,
const char * Channel,
int DataBufferSize,
unsigned char Data[]
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_Zero()

Performs a zero offset null adjustment.

```
EXPORT int PwrSnsr_Zero
(
SessionID Vi,
const char *Channel
)
```

Parameters

Vi: The SessionID handle returned by 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.

PwrSnsr_ZeroQuery()

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

```
EXPORT int PwrSnsr_ZeroQuery
(
SessionID Vi,
const char *Channel,
int *Val
)
```

Parameters

Vi: The SessionID handle returned by 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.

Index

E

enumerations
- error_base
  - ERROR_ALREADY_INITIALIZED 2-6
  - ERROR_INVALID_SESSION_HANDLE 2-6
  - ERROR_NOT_INITIALIZED . 2-6
  - ERROR_NULL_POINTER . . . 2-6
  - ERROR_OPERATION_PENDING 2-6
  - ERROR_OUT_OF_MEMORY . 2-6
  - ERROR_RESET_FAILED 2-6
  - ERROR_RESOURCE_UNKNOWN 2-6
  - ERROR_STATUS_NOT_AVAILABLE 2-6
  - IO_GENERAL 2-6
  - IO_TIMEOUT 2-6
  - MODEL_NOT_SUPPORTED . 2-6
- LIBUSB Errors 2-6
  - ACCESS 2-6
  - BUSY 2-6
  - INTERRUPTED 2-6
  - INVALID_PARAM 2-6
  - IO 2-6
  - NO_DEVICE 2-6
  - NO_MEM 2-6
  - NOT_FOUND 2-6
  - NOT_SUPPORTED 2-6
  - OTHER 2-6
  - OVERFLOW 2-6
  - PIPE 2-6
  - TIMEOUT 2-6
- PwrSnsrAcquisitionStatusEnum . 2-1
- PwrSnsrBandwidthEnum 2-2
- PwrSnsrCondCodeEnum 2-3
- PwrSnsrFilterStateEnum 2-2
- PwrSnsrHoldoffModeEnum 2-3
- PwrSnsrMarkerNumberEnum . . . 2-2
- PwrSnsrMeasBuffGateEnum 2-5
- PwrSnsrMeasBuffStartModeEnum 2-5
- PwrSnsrMeasBuffStopReasonEnum 2-5
- PwrSnsrPulseUnitsEnum 2-2
- PwrSnsrRdgsEnableFlag 2-5
- PwrSnsrStatGatingEnum 2-4
- PwrSnsrTermActionEnum 2-3
- PwrSnsrTriggerModeEnum 2-1
- PwrSnsrTriggerPositionEnum . . . 2-1
- PwrSnsrTriggerSlopeEnum 2-1
- PwrSnsrTriggerSourceEnum 2-4
- PwrSnsrTriggerStatusEnum 2-3
- PwrSnsrTrigOutModeEnum 2-4
- PwrSnsrUnitsEnum 2-2

Error Codes
- ERROR_ALREADY_INITIALIZED 2-6
- ERROR_INVALID_SESSION_HANDLE 2-6
- ERROR_NOT_INITIALIZED 2-6
- ERROR_NULL_POINTER 2-6
- ERROR_OPERATION_PENDING 2-6
- ERROR_OUT_OF_MEMORY 2-6
- ERROR_RESET_FAILED 2-6
- ERROR_RESOURCE_UNKNOWN 2-6
- ERROR_STATUS_NOT_AVAILABLE 2-6
- ERROR_UNEXPECTED_RESPONSE 2-6
- INV_PARAMETER 2-6
- IO_GENERAL 2-6
- IO_TIMEOUT 2-6
- MODEL_NOT_SUPPORTED 2-6

F

FallDistal, PulseInfo 1-1
FallProximal, PulseInfo 1-1
FallTime, PulseInfo 1-1

L

LIBUSB Errors
- ACCESS 2-6
- BUSY 2-6
- INTERRUPTED 2-6
- INVALID_PARAM 2-6
- IO 2-6
- NO_DEVICE 2-6
- NO_MEM 2-6
- NOT_FOUND 2-6
- NOT_SUPPORTED 2-6
- OVERFLOW 2-6
- PIPE 2-6
- TIMEOUT 2-6

M

macros, programming 1-1
Min, PulseInfo 1-1

O

OTHER 2-6

P

Peak 1-1
Position, PulseInfo 1-1
PulseAvg, PulseInfo 1-1
PulseInfo, struct 1-1

R

RiseDistal, PulseInfo 1-1
RiseProximal, PulseInfo 1-1
RiseTime, PulseInfo 1-1

S

struct, pulse information 1-1

W

Width, PulseInfo 1-1

Index

A

acquisition
- continuous, restart 3-119

E

enumerations
- error_base
  - ERROR_ALREADY_INITIALIZED 2-7
  - ERROR_INVALID_SESSION_HANDLE 2-7
  - ERROR_NOT_INITIALIZED . 2-7
  - ERROR_NULL_POINTER . . . 2-7
  - ERROR_OPERATION_PENDING 2-7
  - ERROR_OUT_OF_MEMORY . 2-7
  - ERROR_RESET_FAILED 2-7
  - ERROR_RESOURCE_UNKNOWN 2-7
  - ERROR_STATUS_NOT_AVAILABLE 2-7
  - IO_GENERAL 2-7
  - IO_TIMEOUT 2-7
  - MODEL_NOT_SUPPORTED . 2-7
- LIBUSB Errors 2-7
  - ACCESS 2-7
  - BUSY 2-7
  - INTERRUPTED 2-7
  - INVALID_PARAM 2-7
  - IO 2-7
  - NO_DEVICE 2-7
  - NO_MEM 2-7
  - NOT_FOUND 2-7
  - NOT_SUPPORTED 2-7
  - OTHER 2-7
  - OVERFLOW 2-7
  - PIPE 2-7
  - TIMEOUT 2-7
- PwrSnsrAcquisitionStatusEnum . 2-1
- PwrSnsrBandwidthEnum 2-2
- PwrSnsrCondCodeEnum 2-3
- PwrSnsrFilterStateEnum 2-2
- PwrSnsrHoldoffModeEnum 2-4
- PwrSnsrMarkerNumberEnum . . . 2-2
- PwrSnsrMeasBuffGateEnum 2-6
- PwrSnsrMeasBuffStartModeEnum 2-6
- PwrSnsrMeasBuffStopReasonEnum 2-6
- PwrSnsrPulseUnitsEnum 2-2
- PwrSnsrRdgsEnableFlag 2-6
- PwrSnsrStatGatingEnum 2-4
- PwrSnsrTermActionEnum 2-3
- PwrSnsrTriggerModeEnum 2-1
- PwrSnsrTriggerPositionEnum . . . 2-1
- PwrSnsrTriggerSlopeEnum 2-1
- PwrSnsrTriggerSourceEnum 2-5
- PwrSnsrTriggerStatusEnum 2-3
- PwrSnsrTrigOutModeEnum 2-5
- PwrSnsrUnitsEnum 2-2

Error Codes
- ERROR_ALREADY_INITIALIZED 2-7
- ERROR_INVALID_SESSION_HANDL 2-7
- ERROR_NOT_INITIALIZED 2-7
- ERROR_NULL_POINTER 2-7
- ERROR_OPERATION_PENDING 2-7
- ERROR_OUT_OF_MEMORY 2-7
- ERROR_RESET_FAILED 2-7
- ERROR_RESOURCE_UNKNOWN 2-7
- ERROR_STATUS_NOT_AVAILABLE 2-7
- ERROR_UNEXPECTED_RESPONSE 2-7
- INV_PARAMETER 2-7
- IO_GENERAL 2-7
- IO_TIMEOUT 2-7
- MODEL_NOT_SUPPORTED 2-7

F

FallDistal, PulseInfo 1-1
FallProximal, PulseInfo 1-1
FallTime, PulseInfo 1-1

I

IEEE
- baseline
  - part of power array 3-24
  - returns 3-13
- overshoot of topline
  - returns 3-23
- topline
  - part of power array 3-24
  - returns 3-13

L

LIBUSB Errors
- ACCESS 2-7
- BUSY 2-7

INTERRUPTED 2-7
INVALID_PARAM 2-7
IO 2-7
NO_DEVICE 2-7
NO_MEM 2-7
NOT_FOUND 2-7
NOT_SUPPORTED 2-7
OVERFLOW 2-7
PIPE 2-7
TIMEOUT 2-7

M

macros, programming 1-1
Min, PulseInfo 1-1

O

OTHER 2-7

P

Peak 1-1
Position, PulseInfo 1-1
PRF
fetch 3-26
read 3-109
Pulse Repetition Frequency
fetch 3-26
read 3-109
Pulse RepetitionFrequency
reciprocal 3-23
PulseAvg, PulseInfo 1-1
PulseInfo, struct 1-1
PwrSnsr_Abort 3-1
PwrSnsr_AcquireMeasurements 3-1
PwrSnsr_AdvanceReadIndex 3-2
PwrSnsr_Clear 3-2
PwrSnsr_ClearBuffer 3-2
PwrSnsr_ClearError 3-2
PwrSnsr_ClearMeasurements 3-3
PwrSnsr_ClearUserCal 3-3
PwrSnsr_close 3-3
PwrSnsr_EnableCapturePriority 3-4
PwrSnsr_FetchAllMultiPulse 3-4
PwrSnsr_FetchArrayMarkerPower . . . 3-5
PwrSnsr_FetchCCDFPercent 3-6
PwrSnsr_FetchCCDFPower 3-7
PwrSnsr_FetchCCDFTrace 3-7
PwrSnsr_FetchCursorPercent 3-8
PwrSnsr_FetchCursorPower 3-8
PwrSnsr_FetchCWArray 3-9
PwrSnsr_FetchCWPower 3-10
PwrSnsr_FetchDistal 3-10
PwrSnsr_FetchDutyCycle 3-11
PwrSnsr_FetchEdgeDelay 3-11
PwrSnsr_FetchExtendedWaveform . . 3-12
PwrSnsr_FetchFallTime 3-12
PwrSnsr_FetchIEEEBottom 3-13
PwrSnsr_FetchIEEETop 3-13
PwrSnsr_FetchIntervalAvg 3-14
PwrSnsr_FetchIntervalFilteredMax . 3-14
PwrSnsr_FetchIntervalFilteredMin . 3-15
PwrSnsr_FetchIntervalMax 3-16
PwrSnsr_FetchIntervalMaxAvg 3-16
PwrSnsr_FetchIntervalMin 3-17
PwrSnsr_FetchIntervalMinAvg 3-17
PwrSnsr_FetchIntervalPkToAvg 3-18
PwrSnsr_FetchMarkerAverage 3-18
PwrSnsr_FetchMarkerDelta 3-19
PwrSnsr_FetchMarkerMax 3-19
PwrSnsr_FetchMarkerMin 3-20
PwrSnsr_FetchMarkerRatio 3-20
PwrSnsr_FetchMarkerRDelta 3-21
PwrSnsr_FetchMarkerRRatio 3-21
PwrSnsr_FetchMesial 3-22
PwrSnsr_FetchOfftime 3-22
PwrSnsr_FetchOvershoot 3-23
PwrSnsr_FetchPeriod 3-23
PwrSnsr_FetchPowerArray 3-24
PwrSnsr_FetchPRF 3-26
PwrSnsr_FetchProximal 3-26
PwrSnsr_FetchPulseCycleAvg 3-27
PwrSnsr_FetchPulseOnAverage 3-27
PwrSnsr_FetchPulsePeak 3-28
PwrSnsr_FetchRiseTime 3-28
PwrSnsr_FetchStatMeasurementArray 3-29
PwrSnsr_FetchTimeArray 3-30
PwrSnsr_FetchWaveform 3-33
PwrSnsr_FetchWaveformMinMax . . 3-33
PwrSnsr_FetchWidth 3-34
PwrSnsr_FindResources 3-35
PwrSnsr_GetAcqStatusArray 3-35
PwrSnsr_GetAttenuation 3-36
PwrSnsr_GetAverage 3-36
PwrSnsr_GetBandwidth 3-37
PwrSnsr_GetBufferedAverageMeasurement
s 3-37
PwrSnsr_GetBufferedMeasurementsAvailab
le 3-38
PwrSnsr_GetCalFactor 3-38
PwrSnsr_GetCalFactors 3-39
PwrSnsr_GetCapture 3-40
PwrSnsr_GetCCDFTraceCount 3-40

PwrSnsr_GetChannelByIndex 3-41
PwrSnsr_GetChannelCount 3-41
PwrSnsr_GetChanTraceCount 3-42
PwrSnsr_GetContinuousCapture . . . 3-42
PwrSnsr_GetCurrentTemp 3-42
PwrSnsr_GetDiagStatusArray 3-43
PwrSnsr_GetDistal 3-44
PwrSnsr_GetDongleSerialNumber . . 3-44
PwrSnsr_GetDuration 3-44
PwrSnsr_GetDurations 3-45
PwrSnsr_GetEnabled 3-45
PwrSnsr_GetEndDelay 3-46
PwrSnsr_GetEndGate 3-46
PwrSnsr_GetEndQual 3-47
PwrSnsr_GetError 3-47
PwrSnsr_GetExpirationDate 3-48
PwrSnsr_GetExternalSkew 3-48
PwrSnsr_GetFactoryCalDate 3-48
PwrSnsr_GetFetchLatency 3-49
PwrSnsr_GetFilterState 3-49
PwrSnsr_GetFilterTime 3-50
PwrSnsr_GetFirmwareVersion 3-50
PwrSnsr_GetFpgaVersion 3-51
PwrSnsr_GetFrequency 3-51
PwrSnsr_GetGateMode 3-52
PwrSnsr_GetGating 3-52
PwrSnsr_GetHorizontalOffset 3-53
PwrSnsr_GetHorizontalScale 3-53
PwrSnsr_GetImpedance 3-54
PwrSnsr_GetInitiateContinuous 3-54
PwrSnsr_GetInternalSkew 3-55
PwrSnsr_GetIsAvailable 3-55
PwrSnsr_GetIsAvgSensor 3-56
PwrSnsr_GetIsRunning 3-56
PwrSnsr_GetManufactureDate 3-57
PwrSnsr_GetMarkerPixelPosition . . 3-57
PwrSnsr_GetMarkerTimePosition . . 3-57
PwrSnsr_GetMaxFreqHighBandwidth 3-58
PwrSnsr_GetMaxFreqLowBandwidth 3-59
PwrSnsr_GetMaxMeasurements 3-59
PwrSnsr_GetMaxTimebase 3-60
PwrSnsr_GetMeasBuffEnabled 3-60
PwrSnsr_GetMeasurementsAvailable 3-60
PwrSnsr_GetMemChanArchive 3-61
PwrSnsr_GetMesial 3-61
PwrSnsr_GetMinFreqHighBandwidth 3-62
PwrSnsr_GetMinFreqLowBandwidth 3-62
PwrSnsr_GetMinimumSupportedFirmware 3-62
PwrSnsr_GetMinimumTrig 3-63
PwrSnsr_GetMinMeasurements 3-63
PwrSnsr_GetModel 3-64
PwrSnsr_GetNumberOfCals 3-64
PwrSnsr_GetOffsetdB 3-64
PwrSnsr_GetOverRan 3-65
PwrSnsr_GetPeakHoldDecay 3-65
PwrSnsr_GetPeakHoldTracking 3-66
PwrSnsr_GetPeakPowerMax 3-66
PwrSnsr_GetPeakPowerMin 3-67
PwrSnsr_GetPercentPosition 3-67
PwrSnsr_GetPeriod 3-68
PwrSnsr_GetPowerPosition 3-68
PwrSnsr_GetProximal 3-68
PwrSnsr_GetPulseUnits 3-69
PwrSnsr_GetRdgsEnableFlag 3-69
PwrSnsr_GetReadingPeriod 3-70
PwrSnsr_GetReturnCount 3-70
PwrSnsr_GetSequenceNumbers 3-71
PwrSnsr_GetSerialNumber 3-71
PwrSnsr_GetSessionCount 3-72
PwrSnsr_GetSlaveSkew 3-72
PwrSnsr_GetStartDelay 3-73
PwrSnsr_GetStartGate 3-73
PwrSnsr_GetStartMode 3-73
PwrSnsr_GetStartQual 3-74
PwrSnsr_GetStartTimes 3-74
PwrSnsr_GetSweepTime 3-75
PwrSnsr_GetTempComp 3-75
PwrSnsr_GetTermAction 3-76
PwrSnsr_GetTermCount 3-76
PwrSnsr_GetTermTime 3-77
PwrSnsr_GetTimebase 3-77
PwrSnsr_GetTimedOut 3-77
PwrSnsr_GetTimeOut 3-78
PwrSnsr_GetTimePerPoint 3-78
PwrSnsr_GetTimespan 3-79
PwrSnsr_GetTraceStartTime 3-79
PwrSnsr_GetTrigDelay 3-79
PwrSnsr_GetTrigHoldoff 3-80
PwrSnsr_GetTrigHoldoffMode 3-81
PwrSnsr_GetTrigLevel 3-81
PwrSnsr_GetTrigMode 3-81
PwrSnsr_GetTrigPosition 3-82
PwrSnsr_GetTrigSlope 3-82
PwrSnsr_GetTrigSource 3-82
PwrSnsr_GetTrigStatus 3-83
PwrSnsr_GetTrigVernier 3-83
PwrSnsr_GetUnits 3-84
PwrSnsr_GetVerticalCenter 3-84
PwrSnsr_GetVerticalScale 3-85

PwrSnsr_GetWriteProtection 3-85
PwrSnsr_init 3-85
PwrSnsr_InitiateAquisition 3-86
PwrSnsr_IsLicenseDongleConnected 3-86
PwrSnsr_LoadMemChanFromArchive 3-87
PwrSnsr_MeasurePower 3-87
PwrSnsr_MeasureVoltage 3-88
PwrSnsr_QueryAverageMeasurements 3-88
PwrSnsr_QueryDurations 3-89
PwrSnsr_QueryMaxMeasurements . . 3-89
PwrSnsr_QueryMinMeasurements . . 3-90
PwrSnsr_QuerySequenceNumbers . . 3-90
PwrSnsr_QueryStartTimes 3-91
PwrSnsr_ReadArrayMarkerPower . . 3-92
PwrSnsr_ReadByteArray 3-93
PwrSnsr_ReadControl 3-94
PwrSnsr_ReadCWArray 3-95
PwrSnsr_ReadCWPower 3-96
PwrSnsr_ReadDutyCycle 3-96
PwrSnsr_ReadEdgeDelay 3-97
PwrSnsr_ReadFallTime 3-97
PwrSnsr_ReadIEEEBottom 3-98
PwrSnsr_ReadIEEETop 3-98
PwrSnsr_ReadIntervalAvg 3-99
PwrSnsr_ReadIntervalFilteredMax . 3-99
PwrSnsr_ReadIntervalFilteredMin . 3-100
PwrSnsr_ReadIntervalMax 3-100
PwrSnsr_ReadIntervalMaxAvg 3-101
PwrSnsr_ReadIntervalMin 3-101
PwrSnsr_ReadIntervalMinAvg 3-102
PwrSnsr_ReadIntervalPkToAvg . . . 3-102
PwrSnsr_ReadMarkerAverage 3-103
PwrSnsr_ReadMarkerDelta 3-103
PwrSnsr_ReadMarkerMax 3-104
PwrSnsr_ReadMarkerMin 3-104
PwrSnsr_ReadMarkerRatio 3-105
PwrSnsr_ReadMarkerRDelta 3-105
PwrSnsr_ReadMarkerRRatio 3-106
PwrSnsr_ReadOfftime 3-106
PwrSnsr_ReadOvershoot 3-107
PwrSnsr_ReadPeriod 3-107
PwrSnsr_ReadPowerArray 3-108
PwrSnsr_ReadPRF 3-109
PwrSnsr_ReadPulseCycleAvg 3-110
PwrSnsr_ReadPulseOnAverage 3-110
PwrSnsr_ReadPulsePeak 3-111
PwrSnsr_ReadRiseTime 3-111
PwrSnsr_ReadSCPI 3-112
PwrSnsr_ReadSCPIBytes 3-112
PwrSnsr_ReadSCPIFromNamedParser 3-113

PwrSnsr_ReadTimeArray 3-113
PwrSnsr_ReadWaveform 3-116
PwrSnsr_ReadWaveformMinMax . . 3-117
PwrSnsr_ReadWidth 3-118
PwrSnsr_reset 3-118
PwrSnsr_ResetContinuousCapture . 3-119
PwrSnsr_SaveToMemoryChannel . . 3-119
PwrSnsr_SaveUserCal 3-119
PwrSnsr_self_test 3-120
PwrSnsr_SendSCPIBytes 3-120
PwrSnsr_SendSCPICommand 3-120
PwrSnsr_SendSCPIToNamedParser 3-121
PwrSnsr_SetAverage 3-121
PwrSnsr_SetBandwidth 3-122
PwrSnsr_SetCalFactor 3-122
PwrSnsr_SetCapture 3-123
PwrSnsr_SetCCDFTraceCount 3-123
PwrSnsr_SetContinuousCapture . . 3-124
PwrSnsr_SetDistal 3-124
PwrSnsr_SetDuration 3-124
PwrSnsr_SetEnabled 3-125
PwrSnsr_SetEndDelay 3-125
PwrSnsr_SetEndGate 3-126
PwrSnsr_SetEndQual 3-126
PwrSnsr_SetExternalSkew 3-126
PwrSnsr_SetFetchLatency 3-127
PwrSnsr_SetFilterState 3-127
PwrSnsr_SetFilterTime 3-128
PwrSnsr_SetFrequency 3-128
PwrSnsr_SetGateMode 3-129
PwrSnsr_SetGating 3-129
PwrSnsr_SetHorizontalOffset 3-129
PwrSnsr_SetHorizontalScale 3-130
PwrSnsr_SetInitiateContinuous . . . 3-130
PwrSnsr_SetInternalSkew 3-131
PwrSnsr_SetMarkerPixelPosition . . 3-131
PwrSnsr_SetMarkerTimePosition . . 3-131
PwrSnsr_SetMeasBuffEnabled 3-132
PwrSnsr_SetMesial 3-133
PwrSnsr_SetOffsetdB 3-133
PwrSnsr_SetPeakHoldDecay 3-134
PwrSnsr_SetPeakHoldTracking . . . 3-134
PwrSnsr_SetPercentPosition 3-135
PwrSnsr_SetPeriod 3-135
PwrSnsr_SetPowerPosition 3-135
PwrSnsr_SetProximal 3-136
PwrSnsr_SetPulseUnits 3-136
PwrSnsr_SetRdgsEnableFlag 3-137
PwrSnsr_SetReturnCount 3-137
PwrSnsr_SetSessionCount 3-137

PwrSnsr_SetSessionTimeout 3-138
PwrSnsr_SetSlaveSkew 3-138
PwrSnsr_SetStartDelay 3-139
PwrSnsr_SetStartGate 3-139
PwrSnsr_SetStartMode 3-139
PwrSnsr_SetStartQual 3-140
PwrSnsr_SetTempComp 3-140
PwrSnsr_SetTermAction 3-141
PwrSnsr_SetTermCount 3-141
PwrSnsr_SetTermTime 3-142
PwrSnsr_SetTimebase 3-142
PwrSnsr_SetTimeOut 3-142
PwrSnsr_SetTimespan 3-143
PwrSnsr_SetTrigDelay 3-143
PwrSnsr_SetTrigHoldoff 3-143
PwrSnsr_SetTrigHoldoffMode 3-144
PwrSnsr_SetTrigLevel 3-145
PwrSnsr_SetTrigMode 3-145
PwrSnsr_SetTrigOutMode 3-145
PwrSnsr_SetTrigPosition 3-146
PwrSnsr_SetTrigSlope 3-146
PwrSnsr_SetTrigSource 3-147
PwrSnsr_SetTrigVernier 3-147
PwrSnsr_SetUnits 3-147
PwrSnsr_SetVerticalCenter 3-148
PwrSnsr_SetVerticalScale 3-148
PwrSnsr_SetWriteProtection 3-149
PwrSnsr_StartAcquisition 3-149
PwrSnsr_StatModeReset 3-149
PwrSnsr_Status 3-150
PwrSnsr_StopAcquisition 3-150
PwrSnsr_Write 3-150
PwrSnsr_Zero 3-151
PwrSnsr_ZeroQuery 3-151

R

RiseDistal, PulseInfo 1-1
RiseProximal, PulseInfo 1-1
RiseTime, PulseInfo 1-1

S

struct, pulse information 1-1

W

Width, PulseInfo 1-1

Anritsu

10585-00034

B

Anritsu utilizes recycled paper and environmentally conscious inks and toner.

Anritsu Company
490 Jarvis Drive
Morgan Hill, CA 95037-2809
USA
http://www.anritsu.com