

Power Sensor Library

USER MANUAL

© 2019 WTG
WTG

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

1.	Data Structures	15
1.1	Data Structures	16
1.1.1	PulseInfo	16
1.1.1.1	FallDistal	17
1.1.1.2	FallProximal	17
1.1.1.3	FallTime	17
1.1.1.4	Min	17
1.1.1.5	Peak	18
1.1.1.6	Position	18
1.1.1.7	PulseAvg	18
1.1.1.8	RiseDistal	18
1.1.1.9	RiseProximal	18
1.1.1.10	RiseTime	18
1.1.1.11	Width	18
1.2	Data Structure Index	19
1.3	Data Fields	19
1.3.1	All	19
1.3.2	Variables	20
2.	Files	21
2.1	File List	22
2.1.1	PwrSnsrLib.h	22
2.1.1.1	PulseInfo	261
2.1.1.1.1	FallDistal	262
2.1.1.1.2	FallProximal	262
2.1.1.1.3	FallTime	262
2.1.1.1.4	Min	262
2.1.1.1.5	Peak	262
2.1.1.1.6	Position	263
2.1.1.1.7	PulseAvg	263
2.1.1.1.8	RiseDistal	263
2.1.1.1.9	RiseProximal	263
2.1.1.1.10	RiseTime	263
2.1.1.1.11	Width	263
2.1.1.2	CURRENT_TIMEOUT	264
2.1.1.3	ERROR_BASE	264
2.1.1.4	EXPORT	264
2.1.1.5	SUCCESS	264
2.1.1.6	PulseInfo	320
2.1.1.7	PwrSnsrAcquisitionStatusEnum	320
2.1.1.8	PwrSnsrBandwidthEnum	320
2.1.1.9	PwrSnsrCondCodeEnum	320
2.1.1.10	PwrSnsrErrorCodesEnum	320
2.1.1.11	PwrSnsrFilterStateEnum	320

2.1.1.12	PwrSnsrHoldoffModeEnum	321
2.1.1.13	PwrSnsrMarkerNumberEnum	321
2.1.1.14	PwrSnsrMeasBuffGateEnum	321
2.1.1.15	PwrSnsrMeasBuffStartModeEnum	321
2.1.1.16	PwrSnsrMeasBuffStopReasonEnum	321
2.1.1.17	PwrSnsrPulseUnitsEnum	321
2.1.1.18	PwrSnsrRdgsEnableFlag	321
2.1.1.19	PwrSnsrStatGatingEnum	322
2.1.1.20	PwrSnsrTermActionEnum	322
2.1.1.21	PwrSnsrTriggerModeEnum	322
2.1.1.22	PwrSnsrTriggerPositionEnum	322
2.1.1.23	PwrSnsrTriggerSlopeEnum	322
2.1.1.24	PwrSnsrTriggerSourceEnum	322
2.1.1.25	PwrSnsrTriggerStatusEnum	322
2.1.1.26	PwrSnsrTrigOutModeEnum	323
2.1.1.27	PwrSnsrUnitsEnum	323
2.1.1.28	SessionID	264
2.1.1.29	PwrSnsrAcquisitionStatusEnum	323
2.1.1.29.1	PwrSnsrAcqComplete	323
2.1.1.29.2	PwrSnsrAcqInProgress	323
2.1.1.29.3	PwrSnsrAcqStatusUnknown	323
2.1.1.30	PwrSnsrBandwidthEnum	323
2.1.1.30.1	PwrSnsrBandwidthHigh	323
2.1.1.30.2	PwrSnsrBandwidthLow	323
2.1.1.31	PwrSnsrCondCodeEnum	323
2.1.1.31.1	PwrSnsrCondCodeMeasurementStopped	324
2.1.1.31.2	PwrSnsrCondCodeError	324
2.1.1.31.3	PwrSnsrCondCodeUnderrange	324
2.1.1.31.4	PwrSnsrCondCodeOverrange	324
2.1.1.31.5	PwrSnsrCondCodeNormal	324
2.1.1.32	PwrSnsrErrorCodesEnum	324
2.1.1.32.1	PWR_SNSR_IO_GENERAL	324
2.1.1.32.2	PWR_SNSR_IO_TIMEOUT	324
2.1.1.32.3	PWR_SNSR_MODEL_NOT_SUPPORTED	324
2.1.1.32.4	PWR_SNSR_INV_PARAMETER	324
2.1.1.32.5	PWR_SNSR_ERROR_INVALID_SESSION_HANDLE	324
2.1.1.32.6	PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE	324
2.1.1.32.7	PWR_SNSR_ERROR_RESET_FAILED	324
2.1.1.32.8	PWR_SNSR_ERROR_RESOURCE_UNKNOWN	324
2.1.1.32.9	PWR_SNSR_ERROR_ALREADY_INITIALIZED	324
2.1.1.32.10	PWR_SNSR_ERROR_OUT_OF_MEMORY	324
2.1.1.32.11	PWR_SNSR_ERROR_OPERATION_PENDING	324
2.1.1.32.12	PWR_SNSR_ERROR_NULL_POINTER	325
2.1.1.32.13	PWR_SNSR_ERROR_UNEXPECTED_RESPONSE	325
2.1.1.32.14	PWR_SNSR_ERROR_NOT_INITIALIZED	325
2.1.1.32.15	PWR_SNSR_LIBUSB_ERROR_IO	325
2.1.1.32.16	PWR_SNSR_LIBUSB_ERROR_INVALID_PARAM	325

2.1.1.32.17	PWR_SNSR_LIBUSB_ERROR_ACCESS	325
2.1.1.32.18	PWR_SNSR_LIBUSB_ERROR_NO_DEVICE	325
2.1.1.32.19	PWR_SNSR_LIBUSB_ERROR_NOT_FOUND	325
2.1.1.32.20	PWR_SNSR_LIBUSB_ERROR_BUSY	325
2.1.1.32.21	PWR_SNSR_LIBUSB_ERROR_TIMEOUT	325
2.1.1.32.22	PWR_SNSR_LIBUSB_ERROR_OVERFLOW	325
2.1.1.32.23	PWR_SNSR_LIBUSB_ERROR_PIPE	325
2.1.1.32.24	PWR_SNSR_LIBUSB_ERROR_INTERRUPTED	325
2.1.1.32.25	PWR_SNSR_LIBUSB_ERROR_NO_MEM	325
2.1.1.32.26	PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED	325
2.1.1.32.27	PWR_SNSR_LIBUSB_ERROR_OTHER	325
2.1.1.33	PwrSnsrFilterStateEnum	325
2.1.1.33.1	PwrSnsrFilterStateOff	325
2.1.1.33.2	PwrSnsrFilterStateOn	326
2.1.1.33.3	PwrSnsrFilterStateAuto	326
2.1.1.34	PwrSnsrHoldoffModeEnum	326
2.1.1.34.1	PwrSnsrHoldoffModeNormal	326
2.1.1.34.2	PwrSnsrHoldoffModeGap	326
2.1.1.35	PwrSnsrMarkerNumberEnum	326
2.1.1.35.1	PwrSnsrMarkerNumberMarker1	326
2.1.1.35.2	PwrSnsrMarkerNumberMarker2	326
2.1.1.36	PwrSnsrMeasBuffGateEnum	326
2.1.1.36.1	PwrSnsrMeasBuffGateBurst	264
2.1.1.36.2	PwrSnsrMeasBuffGateMarker	264
2.1.1.36.3	PwrSnsrMeasBuffGateExtGate	264
2.1.1.36.4	PwrSnsrMeasBuffGatePeriodic	264
2.1.1.36.5	PwrSnsrMeasBuffGateExtTrig	264
2.1.1.37	PwrSnsrMeasBuffStartModeEnum	326
2.1.1.37.1	PwrSnsrMeasBuffStartModeImmediate	264
2.1.1.37.2	PwrSnsrMeasBuffStartModeExternalEnable	264
2.1.1.37.3	PwrSnsrMeasBuffStartModeExternalStart	264
2.1.1.38	PwrSnsrMeasBuffStopReasonEnum	327
2.1.1.38.1	PwrSnsrMeasBuffStopReasonCountReached	264
2.1.1.38.2	PwrSnsrMeasBuffStopReasonTimedOut	264
2.1.1.38.3	PwrSnsrMeasBuffStopReasonBufferOverran	264
2.1.1.38.4	PwrSnsrMeasBuffStopReasonNone	264
2.1.1.39	PwrSnsrPulseUnitsEnum	327
2.1.1.39.1	PwrSnsrPulseUnitsWatts	327
2.1.1.39.2	PwrSnsrPulseUnitsVolts	327
2.1.1.40	PwrSnsrRdgsEnableFlag	327
2.1.1.40.1	PwrSnsrSequenceEnable	327
2.1.1.40.2	PwrSnsrStartTimeEnable	327
2.1.1.40.3	PwrSnsrDurationEnable	327
2.1.1.40.4	PwrSnsrMinEnable	327
2.1.1.40.5	PwrSnsrAvgEnable	327
2.1.1.40.6	PwrSnsrMaxEnable	327

2.1.1.41	PwrSnsrStatGatingEnum	327
2.1.1.41.1	PwrSnsrStatGatingFreeRun	328
2.1.1.41.2	PwrSnsrStatGatingMarkers	328
2.1.1.42	PwrSnsrTermActionEnum	328
2.1.1.42.1	PwrSnsrTermActionStop	328
2.1.1.42.2	PwrSnsrTermActionRestart	328
2.1.1.42.3	PwrSnsrTermActionDecimate	328
2.1.1.43	PwrSnsrTriggerModeEnum	328
2.1.1.43.1	PwrSnsrTriggerModeNormal	328
2.1.1.43.2	PwrSnsrTriggerModeAuto	328
2.1.1.43.3	PwrSnsrTriggerModeAutoLevel	329
2.1.1.43.4	PwrSnsrTriggerModeFreerun	264
2.1.1.44	PwrSnsrTriggerPositionEnum	329
2.1.1.44.1	PwrSnsrTriggerPositionLeft	329
2.1.1.44.2	PwrSnsrTriggerPositionMiddle	329
2.1.1.44.3	PwrSnsrTriggerPositionRight	329
2.1.1.45	PwrSnsrTriggerSlopeEnum	329
2.1.1.45.1	PwrSnsrTriggerSlopePositive	329
2.1.1.45.2	PwrSnsrTriggerSlopeNegative	329
2.1.1.46	PwrSnsrTriggerSourceEnum	329
2.1.1.46.1	PwrSnsrTriggerSourceChannel1	329
2.1.1.46.2	PwrSnsrTriggerSourceExternal	330
2.1.1.46.3	PwrSnsrTriggerSourceChannel2	330
2.1.1.46.4	PwrSnsrTriggerSourceChannel3	330
2.1.1.46.5	PwrSnsrTriggerSourceChannel4	330
2.1.1.46.6	PwrSnsrTriggerSourceChannel5	330
2.1.1.46.7	PwrSnsrTriggerSourceChannel6	330
2.1.1.46.8	PwrSnsrTriggerSourceChannel7	330
2.1.1.46.9	PwrSnsrTriggerSourceChannel8	330
2.1.1.46.10	PwrSnsrTriggerSourceChannel9	330
2.1.1.46.11	PwrSnsrTriggerSourceChannel10	330
2.1.1.46.12	PwrSnsrTriggerSourceChannel11	330
2.1.1.46.13	PwrSnsrTriggerSourceChannel12	330
2.1.1.46.14	PwrSnsrTriggerSourceChannel13	330
2.1.1.46.15	PwrSnsrTriggerSourceChannel14	330
2.1.1.46.16	PwrSnsrTriggerSourceChannel15	330
2.1.1.46.17	PwrSnsrTriggerSourceChannel16	330
2.1.1.46.18	PwrSnsrTriggerSourceIndependent	330
2.1.1.47	PwrSnsrTriggerStatusEnum	330
2.1.1.47.1	PwrSnsrTriggerStatusStopped	330
2.1.1.47.2	PwrSnsrTriggerStatusPretrig	331
2.1.1.47.3	PwrSnsrTriggerStatusWaiting	331
2.1.1.47.4	PwrSnsrTriggerStatusAcquiringNew	331
2.1.1.47.5	PwrSnsrTriggerStatusAutoTrig	331
2.1.1.47.6	PwrSnsrTriggerStatusFreerun	331
2.1.1.47.7	PwrSnsrTriggerStatusTriggered	331

2.1.1.47.8	PwrSnsrTriggerStatusRunning	331
2.1.1.48	PwrSnsrTrigOutModeEnum	331
2.1.1.48.1	PwrSnsrTrigOutModeMioOff	264
2.1.1.48.2	PwrSnsrTrigOutModeMioPullUp	264
2.1.1.48.3	PwrSnsrTrigOutModeMioTtl0	264
2.1.1.48.4	PwrSnsrTrigOutModeMioTbRef	264
2.1.1.48.5	PwrSnsrTrigOutModeMioSweepHigh	264
2.1.1.48.6	PwrSnsrTrigOutModeMioSweepLow	264
2.1.1.48.7	PwrSnsrTrigOutModeMioTrigHigh	264
2.1.1.48.8	PwrSnsrTrigOutModeMioTrigLow	264
2.1.1.48.9	PwrSnsrTrigOutModeMioMaster	264
2.1.1.48.10	PwrSnsrTrigOutModeMioSlave	264
2.1.1.49	PwrSnsrUnitsEnum	331
2.1.1.49.1	PwrSnsrUnitsdBm	331
2.1.1.49.2	PwrSnsrUnitswatts	331
2.1.1.49.3	PwrSnsrUnitsvolts	331
2.1.1.49.4	PwrSnsrUnitsDBV	331
2.1.1.49.5	PwrSnsrUnitsDBMV	331
2.1.1.49.6	PwrSnsrUnitsDBUV	331
2.1.1.50	PwrSnsr_Abort	332
2.1.1.51	PwrSnsr_AcquireMeasurements	332
2.1.1.52	PwrSnsr_AdvanceReadIndex	333
2.1.1.53	PwrSnsr_Clear	333
2.1.1.54	PwrSnsr_ClearBuffer	333
2.1.1.55	PwrSnsr_ClearError	334
2.1.1.56	PwrSnsr_ClearMeasurements	334
2.1.1.57	PwrSnsr_ClearUserCal	335
2.1.1.58	PwrSnsr_close	335
2.1.1.59	PwrSnsr_EnableCapturePriority	335
2.1.1.60	PwrSnsr_FetchAllMultiPulse	336
2.1.1.61	PwrSnsr_FetchArrayMarkerPower	337
2.1.1.62	PwrSnsr_FetchCCDFPercent	338
2.1.1.63	PwrSnsr_FetchCCDFPower	339
2.1.1.64	PwrSnsr_FetchCCDFTrace	339
2.1.1.65	PwrSnsr_FetchCursorPercent	340
2.1.1.66	PwrSnsr_FetchCursorPower	340
2.1.1.67	PwrSnsr_FetchCWArray	341
2.1.1.68	PwrSnsr_FetchCWPower	342
2.1.1.69	PwrSnsr_FetchDistal	342
2.1.1.70	PwrSnsr_FetchDutyCycle	343
2.1.1.71	PwrSnsr_FetchEdgeDelay	344
2.1.1.72	PwrSnsr_FetchExtendedWaveform	344
2.1.1.73	PwrSnsr_FetchFallTime	345
2.1.1.74	PwrSnsr_FetchIEEEBottom	345
2.1.1.75	PwrSnsr_FetchIEETop	346
2.1.1.76	PwrSnsr_FetchIntervalAvg	347
2.1.1.77	PwrSnsr_FetchIntervalFilteredMax	347

2.1.1.78	PwrSnsr_FetchIntervalFilteredMin	348
2.1.1.79	PwrSnsr_FetchIntervalMax	348
2.1.1.80	PwrSnsr_FetchIntervalMaxAvg	349
2.1.1.81	PwrSnsr_FetchIntervalMin	350
2.1.1.82	PwrSnsr_FetchIntervalMinAvg	350
2.1.1.83	PwrSnsr_FetchIntervalPkToAvg	351
2.1.1.84	PwrSnsr_FetchMarkerAverage	352
2.1.1.85	PwrSnsr_FetchMarkerDelta	352
2.1.1.86	PwrSnsr_FetchMarkerMax	353
2.1.1.87	PwrSnsr_FetchMarkerMin	353
2.1.1.88	PwrSnsr_FetchMarkerRatio	354
2.1.1.89	PwrSnsr_FetchMarkerRDelta	355
2.1.1.90	PwrSnsr_FetchMarkerRRatio	355
2.1.1.91	PwrSnsr_FetchMesial	356
2.1.1.92	PwrSnsr_FetchOfftime	356
2.1.1.93	PwrSnsr_FetchOvershoot	357
2.1.1.94	PwrSnsr_FetchPeriod	358
2.1.1.95	PwrSnsr_FetchPowerArray	358
2.1.1.96	PwrSnsr_FetchPRF	360
2.1.1.97	PwrSnsr_FetchProximal	361
2.1.1.98	PwrSnsr_FetchPulseCycleAvg	361
2.1.1.99	PwrSnsr_FetchPulseOnAverage	362
2.1.1.100	PwrSnsr_FetchPulsePeak	362
2.1.1.101	PwrSnsr_FetchRiseTime	363
2.1.1.102	PwrSnsr_FetchStatMeasurementArray	363
2.1.1.103	PwrSnsr_FetchTimeArray	365
2.1.1.104	PwrSnsr_FetchWaveform	367
2.1.1.105	PwrSnsr_FetchWaveformMinMax	368
2.1.1.106	PwrSnsr_FetchWidth	369
2.1.1.107	PwrSnsr_FindResources	370
2.1.1.108	PwrSnsr_GetAcqStatusArray	370
2.1.1.109	PwrSnsr_GetAttenuation	371
2.1.1.110	PwrSnsr_GetAverage	372
2.1.1.111	PwrSnsr_GetBandwidth	372
2.1.1.112	PwrSnsr_GetBufferedAverageMeasurements	373
2.1.1.113	PwrSnsr_GetBufferedMeasurementsAvailable	373
2.1.1.114	PwrSnsr_GetCalFactor	374
2.1.1.115	PwrSnsr_GetCalFactors	374
2.1.1.116	PwrSnsr_GetCapture	375
2.1.1.117	PwrSnsr_GetCCDFTraceCount	376
2.1.1.118	PwrSnsr_GetChannelByIndex	376
2.1.1.119	PwrSnsr_GetChannelCount	377
2.1.1.120	PwrSnsr_GetChanTraceCount	377
2.1.1.121	PwrSnsr_GetContinuousCapture	378
2.1.1.122	PwrSnsr_GetCurrentTemp	378
2.1.1.123	PwrSnsr_GetDiagStatusArray	379
2.1.1.124	PwrSnsr_GetDistal	380
2.1.1.125	PwrSnsr_GetDongleSerialNumber	380

2.1.1.126	PwrSnsr_GetDuration	380
2.1.1.127	PwrSnsr_GetDurations	381
2.1.1.128	PwrSnsr_GetEnabled	381
2.1.1.129	PwrSnsr_GetEndDelay	382
2.1.1.130	PwrSnsr_GetEndGate	382
2.1.1.131	PwrSnsr_GetEndQual	383
2.1.1.132	PwrSnsr_GetError	383
2.1.1.133	PwrSnsr_GetExpirationDate	384
2.1.1.134	PwrSnsr_GetExternalSkew	384
2.1.1.135	PwrSnsr_GetFactoryCalDate	385
2.1.1.136	PwrSnsr_GetFetchLatency	385
2.1.1.137	PwrSnsr_GetFilterState	386
2.1.1.138	PwrSnsr_GetFilterTime	386
2.1.1.139	PwrSnsr_GetFirmwareVersion	387
2.1.1.140	PwrSnsr_GetFpgaVersion	387
2.1.1.141	PwrSnsr_GetFrequency	388
2.1.1.142	PwrSnsr_GetGateMode	388
2.1.1.143	PwrSnsr_GetGating	389
2.1.1.144	PwrSnsr_GetHorizontalOffset	389
2.1.1.145	PwrSnsr_GetHorizontalScale	390
2.1.1.146	PwrSnsr_GetImpedance	390
2.1.1.147	PwrSnsr_GetInitiateContinuous	391
2.1.1.148	PwrSnsr_GetInternalSkew	392
2.1.1.149	PwrSnsr_GetIsAvailable	392
2.1.1.150	PwrSnsr_GetIsAvgSensor	393
2.1.1.151	PwrSnsr_GetIsRunning	393
2.1.1.152	PwrSnsr_GetManufactureDate	394
2.1.1.153	PwrSnsr_GetMarkerPixelPosition	394
2.1.1.154	PwrSnsr_GetMarkerTimePosition	395
2.1.1.155	PwrSnsr_GetMaxFreqHighBandwidth	395
2.1.1.156	PwrSnsr_GetMaxFreqLowBandwidth	396
2.1.1.157	PwrSnsr_GetMaxMeasurements	396
2.1.1.158	PwrSnsr_GetMaxTimebase	397
2.1.1.159	PwrSnsr_GetMeasBuffEnabled	397
2.1.1.160	PwrSnsr_GetMeasurementsAvailable	398
2.1.1.161	PwrSnsr_GetMemChanArchive	398
2.1.1.162	PwrSnsr_GetMesial	399
2.1.1.163	PwrSnsr_GetMinFreqHighBandwidth	399
2.1.1.164	PwrSnsr_GetMinFreqLowBandwidth	400
2.1.1.165	PwrSnsr_GetMinimumSupportedFirmware	400
2.1.1.166	PwrSnsr_GetMinimumTrig	401
2.1.1.167	PwrSnsr_GetMinMeasurements	401
2.1.1.168	PwrSnsr_GetModel	402
2.1.1.169	PwrSnsr_GetNumberOfCals	402
2.1.1.170	PwrSnsr_GetOffsetdB	403
2.1.1.171	PwrSnsr_GetOverRan	403
2.1.1.172	PwrSnsr_GetPeakHoldDecay	404
2.1.1.173	PwrSnsr_GetPeakHoldTracking	404

2.1.1.174	PwrSnsr_GetPeakPowerMax	405
2.1.1.175	PwrSnsr_GetPeakPowerMin	405
2.1.1.176	PwrSnsr_GetPercentPosition	406
2.1.1.177	PwrSnsr_GetPeriod	406
2.1.1.178	PwrSnsr_GetPowerPosition	407
2.1.1.179	PwrSnsr_GetProximal	407
2.1.1.180	PwrSnsr_GetPulseUnits	408
2.1.1.181	PwrSnsr_GetRdgsEnableFlag	408
2.1.1.182	PwrSnsr_GetReadingPeriod	409
2.1.1.183	PwrSnsr_GetReturnCount	409
2.1.1.184	PwrSnsr_GetSequenceNumbers	409
2.1.1.185	PwrSnsr_GetSerialNumber	410
2.1.1.186	PwrSnsr_GetSessionCount	411
2.1.1.187	PwrSnsr_GetSlaveSkew	411
2.1.1.188	PwrSnsr_GetStartDelay	412
2.1.1.189	PwrSnsr_GetStartGate	412
2.1.1.190	PwrSnsr_GetStartMode	413
2.1.1.191	PwrSnsr_GetStartQual	413
2.1.1.192	PwrSnsr_GetStartTimes	414
2.1.1.193	PwrSnsr_GetSweepTime	414
2.1.1.194	PwrSnsr_GetTempComp	415
2.1.1.195	PwrSnsr_GetTermAction	415
2.1.1.196	PwrSnsr_GetTermCount	416
2.1.1.197	PwrSnsr_GetTermTime	416
2.1.1.198	PwrSnsr_GetTimebase	417
2.1.1.199	PwrSnsr_GetTimedOut	417
2.1.1.200	PwrSnsr_GetTimeOut	418
2.1.1.201	PwrSnsr_GetTimePerPoint	418
2.1.1.202	PwrSnsr_GetTimespan	419
2.1.1.203	PwrSnsr_GetTraceStartTime	419
2.1.1.204	PwrSnsr_GetTrigDelay	420
2.1.1.205	PwrSnsr_GetTrigHoldoff	420
2.1.1.206	PwrSnsr_GetTrigHoldoffMode	421
2.1.1.207	PwrSnsr_GetTrigLevel	421
2.1.1.208	PwrSnsr_GetTrigMode	422
2.1.1.209	PwrSnsr_GetTrigPosition	422
2.1.1.210	PwrSnsr_GetTrigSlope	423
2.1.1.211	PwrSnsr_GetTrigSource	423
2.1.1.212	PwrSnsr_GetTrigStatus	424
2.1.1.213	PwrSnsr_GetTrigVernier	424
2.1.1.214	PwrSnsr_GetUnits	425
2.1.1.215	PwrSnsr_GetVerticalCenter	425
2.1.1.216	PwrSnsr_GetVerticalScale	426
2.1.1.217	PwrSnsr_GetWriteProtection	426
2.1.1.218	PwrSnsr_init	427
2.1.1.219	PwrSnsr_InitiateAquisition	427
2.1.1.220	PwrSnsr_IsLicenseDongleConnected	428
2.1.1.221	PwrSnsr_LoadMemChanFromArchive	428

2.1.1.222	PwrSnsr_MeasurePower	429
2.1.1.223	PwrSnsr_MeasureVoltage	430
2.1.1.224	PwrSnsr_QueryAverageMeasurements	430
2.1.1.225	PwrSnsr_QueryDurations	431
2.1.1.226	PwrSnsr_QueryMaxMeasurements	431
2.1.1.227	PwrSnsr_QueryMinMeasurements	432
2.1.1.228	PwrSnsr_QuerySequenceNumbers	433
2.1.1.229	PwrSnsr_QueryStartTimes	433
2.1.1.230	PwrSnsr_ReadArrayMarkerPower	434
2.1.1.231	PwrSnsr_ReadByteArray	435
2.1.1.232	PwrSnsr_ReadControl	436
2.1.1.233	PwrSnsr_ReadCWArray	436
2.1.1.234	PwrSnsr_ReadCWPowPower	437
2.1.1.235	PwrSnsr_ReadDutyCycle	438
2.1.1.236	PwrSnsr_ReadEdgeDelay	439
2.1.1.237	PwrSnsr_ReadFallTime	439
2.1.1.238	PwrSnsr_ReadIEEEBottom	440
2.1.1.239	PwrSnsr_ReadIEEETop	440
2.1.1.240	PwrSnsr_ReadIntervalAvg	441
2.1.1.241	PwrSnsr_ReadIntervalFilteredMax	442
2.1.1.242	PwrSnsr_ReadIntervalFilteredMin	442
2.1.1.243	PwrSnsr_ReadIntervalMax	443
2.1.1.244	PwrSnsr_ReadIntervalMaxAvg	443
2.1.1.245	PwrSnsr_ReadIntervalMin	444
2.1.1.246	PwrSnsr_ReadIntervalMinAvg	445
2.1.1.247	PwrSnsr_ReadIntervalPkToAvg	445
2.1.1.248	PwrSnsr_ReadMarkerAverage	446
2.1.1.249	PwrSnsr_ReadMarkerDelta	447
2.1.1.250	PwrSnsr_ReadMarkerMax	447
2.1.1.251	PwrSnsr_ReadMarkerMin	448
2.1.1.252	PwrSnsr_ReadMarkerRatio	448
2.1.1.253	PwrSnsr_ReadMarkerRDelta	449
2.1.1.254	PwrSnsr_ReadMarkerRRatio	450
2.1.1.255	PwrSnsr_ReadOfftime	450
2.1.1.256	PwrSnsr_ReadOvershoot	451
2.1.1.257	PwrSnsr_ReadPeriod	451
2.1.1.258	PwrSnsr_ReadPowerArray	452
2.1.1.259	PwrSnsr_ReadPRF	454
2.1.1.260	PwrSnsr_ReadPulseCycleAvg	454
2.1.1.261	PwrSnsr_ReadPulseOnAverage	455
2.1.1.262	PwrSnsr_ReadPulsePeak	456
2.1.1.263	PwrSnsr_ReadRiseTime	456
2.1.1.264	PwrSnsr_ReadSCPI	457
2.1.1.265	PwrSnsr_ReadSCPIBytes	457
2.1.1.266	PwrSnsr_ReadSCPIFromNamedParser	458
2.1.1.267	PwrSnsr_ReadTimeArray	459
2.1.1.268	PwrSnsr_ReadWaveform	461
2.1.1.269	PwrSnsr_ReadWaveformMinMax	462

2.1.1.270	PwrSnsr_ReadWidth	463
2.1.1.271	PwrSnsr_reset	464
2.1.1.272	PwrSnsr_ResetContinuousCapture	464
2.1.1.273	PwrSnsr_SaveToMemoryChannel	464
2.1.1.274	PwrSnsr_SaveUserCal	465
2.1.1.275	PwrSnsr_self_test	465
2.1.1.276	PwrSnsr_SendSCPIBytes	466
2.1.1.277	PwrSnsr_SendSCPICommand	466
2.1.1.278	PwrSnsr_SendSCPIToNamedParser	467
2.1.1.279	PwrSnsr_SetAverage	467
2.1.1.280	PwrSnsr_SetBandwidth	468
2.1.1.281	PwrSnsr_SetCalFactor	468
2.1.1.282	PwrSnsr_SetCapture	469
2.1.1.283	PwrSnsr_SetCCDFTraceCount	469
2.1.1.284	PwrSnsr_SetContinuousCapture	470
2.1.1.285	PwrSnsr_SetDistal	470
2.1.1.286	PwrSnsr_SetDuration	471
2.1.1.287	PwrSnsr_SetEnabled	471
2.1.1.288	PwrSnsr_SetEndDelay	472
2.1.1.289	PwrSnsr_SetEndGate	472
2.1.1.290	PwrSnsr_SetEndQual	473
2.1.1.291	PwrSnsr_SetExternalSkew	473
2.1.1.292	PwrSnsr_SetFetchLatency	474
2.1.1.293	PwrSnsr_SetFilterState	474
2.1.1.294	PwrSnsr_SetFilterTime	474
2.1.1.295	PwrSnsr_SetFrequency	475
2.1.1.296	PwrSnsr_SetGateMode	475
2.1.1.297	PwrSnsr_SetGating	476
2.1.1.298	PwrSnsr_SetHorizontalOffset	476
2.1.1.299	PwrSnsr_SetHorizontalScale	477
2.1.1.300	PwrSnsr_SetInitiateContinuous	477
2.1.1.301	PwrSnsr_SetInternalSkew	478
2.1.1.302	PwrSnsr_SetMarkerPixelPosition	479
2.1.1.303	PwrSnsr_SetMarkerTimePosition	479
2.1.1.304	PwrSnsr_SetMeasBuffEnabled	480
2.1.1.305	PwrSnsr_SetMesial	480
2.1.1.306	PwrSnsr_SetOffsetdB	481
2.1.1.307	PwrSnsr_SetPeakHoldDecay	481
2.1.1.308	PwrSnsr_SetPeakHoldTracking	482
2.1.1.309	PwrSnsr_SetPercentPosition	482
2.1.1.310	PwrSnsr_SetPeriod	483
2.1.1.311	PwrSnsr_SetPowerPosition	483
2.1.1.312	PwrSnsr_SetProximal	484
2.1.1.313	PwrSnsr_SetPulseUnits	484
2.1.1.314	PwrSnsr_SetRdgsEnableFlag	485
2.1.1.315	PwrSnsr_SetReturnCount	485
2.1.1.316	PwrSnsr_SetSessionCount	486
2.1.1.317	PwrSnsr_SetSessionTimeout	486

2.1.1.318	PwrSnsr_SetSlaveSkew	487
2.1.1.319	PwrSnsr_SetStartDelay	487
2.1.1.320	PwrSnsr_SetStartGate	488
2.1.1.321	PwrSnsr_SetStartMode	488
2.1.1.322	PwrSnsr_SetStartQual	489
2.1.1.323	PwrSnsr_SetTempComp	489
2.1.1.324	PwrSnsr_SetTermAction	490
2.1.1.325	PwrSnsr_SetTermCount	490
2.1.1.326	PwrSnsr_SetTermTime	491
2.1.1.327	PwrSnsr_SetTimebase	491
2.1.1.328	PwrSnsr_SetTimeOut	492
2.1.1.329	PwrSnsr_SetTimespan	492
2.1.1.330	PwrSnsr_SetTrigDelay	492
2.1.1.331	PwrSnsr_SetTrigHoldoff	493
2.1.1.332	PwrSnsr_SetTrigHoldoffMode	494
2.1.1.333	PwrSnsr_SetTrigLevel	494
2.1.1.334	PwrSnsr_SetTrigMode	495
2.1.1.335	PwrSnsr_SetTrigOutMode	495
2.1.1.336	PwrSnsr_SetTrigPosition	496
2.1.1.337	PwrSnsr_SetTrigSlope	496
2.1.1.338	PwrSnsr_SetTrigSource	497
2.1.1.339	PwrSnsr_SetTrigVernier	497
2.1.1.340	PwrSnsr_SetUnits	498
2.1.1.341	PwrSnsr_SetVerticalCenter	498
2.1.1.342	PwrSnsr_SetVerticalScale	499
2.1.1.343	PwrSnsr_SetWriteProtection	499
2.1.1.344	PwrSnsr_StartAcquisition	500
2.1.1.345	PwrSnsr_StatModeReset	500
2.1.1.346	PwrSnsr_Status	500
2.1.1.347	PwrSnsr_StopAcquisition	501
2.1.1.348	PwrSnsr_Write	501
2.1.1.349	PwrSnsr_Zero	502
2.1.1.350	PwrSnsr_ZeroQuery	502

2.2 Globals 503

2.2.1	All	503
2.2.1.1	p	503
2.2.2	Functions	518
2.2.2.1	p	518
2.2.3	Typedefs	528
2.2.4	Enumerations	529
2.2.5	Enumerator	530

Index 535

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Data Structures

1 Data Structures

Power Sensor Library

Data Structures

Here are the data structures with brief descriptions:

PulseInfo	Data structure containing pulse information
---------------------------	---

Generated by  1.8.15

1. .1 PulseInfo

Power Sensor Library

[Data Fields](#)

PulseInfo Struct Reference

Data structure containing pulse information. [More...](#)

```
#include <PwrSnsrLib.h>
```

Data Fields

float	Width
float	Peak
float	Min
float	PulseAvg
float	Position
float	RiseProximal

float	RiseDistal
float	RiseTime
float	FallProximal
float	FallDistal
float	FallTime

Detailed Description

Data structure containing pulse information.

Field Documentation

◆ [FallDistal](#)

float FallDistal

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

◆ [FallProximal](#)

float FallProximal

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

◆ [FallTime](#)

float FallTime

Fall time of the pulse.

◆ [Min](#)

float Min

Minimum instantaneous power measurement.

◆ Peak

float Peak

Peak (max instantaneous) power measurement.

◆ Position

float Position

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

◆ PulseAvg

float PulseAvg

Average power measurement for the pulse.

◆ RiseDistal

float RiseDistal

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

◆ RiseProximal

float RiseProximal

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

◆ RiseTime

float RiseTime

Rise time of the pulse.

◆ Width

float Width

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

The documentation for this struct was generated from the following file:

- `PwrSnsrLib.h`

Generated by  1.8.15

1.2 Data Structure Index

Power Sensor Library

Data Structure Index

[p](#)

p

[PulseInfo](#)

[p](#)

Generated by  1.8.15

1.3 Data Fields

Power Sensor Library

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- FallDistal : [PulseInfo](#)
- FallProximal : [PulseInfo](#)
- FallTime : [PulseInfo](#)
- Min : [PulseInfo](#)
- Peak : [PulseInfo](#)
- Position : [PulseInfo](#)

- PulseAvg : [PulseInfo](#)
 - RiseDistal : [PulseInfo](#)
 - RiseProximal : [PulseInfo](#)
 - RiseTime : [PulseInfo](#)
 - Width : [PulseInfo](#)
-

Generated by  1.8.15

1.3.2 Variables

Power Sensor Library

- FallDistal : [PulseInfo](#)
 - FallProximal : [PulseInfo](#)
 - FallTime : [PulseInfo](#)
 - Min : [PulseInfo](#)
 - Peak : [PulseInfo](#)
 - Position : [PulseInfo](#)
 - PulseAvg : [PulseInfo](#)
 - RiseDistal : [PulseInfo](#)
 - RiseProximal : [PulseInfo](#)
 - RiseTime : [PulseInfo](#)
 - Width : [PulseInfo](#)
-

Generated by  1.8.15

Files

2 Files

Power Sensor Library

File List

Here is a list of all documented files with brief descriptions:

PwrSnsrLib.h	
------------------------------	--

Generated by  1.8.15

2. .1 PwrSnsrLib.h

Power Sensor Library

[Data Structures](#) | [Macros](#) | [Typedefs](#) | [Enumerations](#) | [Functions](#)

PwrSnsrLib.h File Reference

Go to the source code of this file.

Data Structures

struct	PulseInfo
	Data structure containing pulse information. More...

Macros

#define	SUCCESS (0L)
#define	CURRENT_TIMEOUT (-2)
#define	EXPORT
#define	ERROR_BASE (0xBFFFA0000L)

Typedefs	
typedef int	SessionID
typedef enum	<u>PwrSnsrAcquisitionStatusEnum</u>
	<u>PwrSnsrAcquisitionStatusEnum</u>
typedef enum	<u>PwrSnsrTriggerModeEnum</u>
	<u>PwrSnsrTriggerModeEnum</u>
typedef enum	<u>PwrSnsrTriggerSlopeEnum</u>
	<u>PwrSnsrTriggerSlopeEnum</u>
typedef enum	<u>PwrSnsrTriggerPositionEnum</u>
	<u>PwrSnsrTriggerPositionEnum</u>
typedef enum	<u>PwrSnsrTriggerSourceEnum</u>
	<u>PwrSnsrTriggerSourceEnum</u>
typedef enum	<u>PwrSnsrUnitsEnum</u>
	<u>PwrSnsrUnitsEnum</u>
typedef enum	<u>PwrSnsrMarkerNumberEnum</u>
	<u>PwrSnsrMarkerNumberEnum</u>
typedef enum	<u>PwrSnsrBandwidthEnum</u>
	<u>PwrSnsrBandwidthEnum</u>
typedef enum	<u>PwrSnsrFilterStateEnum</u>
	<u>PwrSnsrFilterStateEnum</u>
typedef enum	<u>PwrSnsrPulseUnitsEnum</u>
	<u>PwrSnsrPulseUnitsEnum</u>
typedef enum	<u>PwrSnsrCondCodeEnum</u>
	<u>PwrSnsrCondCodeEnum</u>
typedef enum	<u>PwrSnsrTriggerStatusEnum</u>
	<u>PwrSnsrTriggerStatusEnum</u>
typedef enum	<u>PwrSnsrTermActionEnum</u>
	<u>PwrSnsrTermActionEnum</u>
typedef enum	<u>PwrSnsrHoldoffModeEnum</u>
	<u>PwrSnsrHoldoffModeEnum</u>

typedef enum PwrSnsrStatGatingEnum	PwrSnsrStatGatingEnum
typedef enum PwrSnsrTrigOutModeEnum	PwrSnsrTrigOutModeEnum
typedef enum PwrSnsrMeasBuffGateEnum	PwrSnsrMeasBuffGateEnum
typedef enum PwrSnsrMeasBuffStartModeEnum	PwrSnsrMeasBuffStartModeEnum
typedef enum PwrSnsrMeasBuffStopReasonEnum	PwrSnsrMeasBuffStopReasonEnum
typedef enum PwrSnsrRdgsEnableFlag	PwrSnsrRdgsEnableFlag
typedef enum PwrSnsrErrorCodesEnum	PwrSnsrErrorCodesEnum
typedef struct PulseInfo	PulseInfo
	Data structure containing pulse information. More...
Enumerations	
enum	PwrSnsrAcquisitionStatusEnum { PwrSnsrAcqComplete = 1, PwrSnsrAcqInProgress = 0, PwrSnsrAcqStatusUnknown = -1 }
enum	PwrSnsrTriggerModeEnum { PwrSnsrTriggerModeNormal = 1, PwrSnsrTriggerModeAuto = 2, PwrSnsrTriggerModeAutoLevel = 3, PwrSnsrTriggerModeFreerun = 4 }
enum	PwrSnsrTriggerSlopeEnum { PwrSnsrTriggerSlopePositive = 1, PwrSnsrTriggerSlopeNegative = 0 }

enum	<u>PwrSnsrTriggerPositionEnum</u> { <u>PwrSnsrTriggerPositionLeft</u> = 0, <u>PwrSnsrTriggerPositionMiddle</u> = 1, <u>PwrSnsrTriggerPositionRight</u> = 2 }
enum	<u>PwrSnsrTriggerSourceEnum</u> { <u>PwrSnsrTriggerSourceChannel1</u> = 0, <u>PwrSnsrTriggerSourceExternal</u> = 2, <u>PwrSnsrTriggerSourceChannel2</u> = 1, <u>PwrSnsrTriggerSourceChannel3</u> = 3, <u>PwrSnsrTriggerSourceChannel4</u> = 4, <u>PwrSnsrTriggerSourceChannel5</u> = 5, <u>PwrSnsrTriggerSourceChannel6</u> = 6, <u>PwrSnsrTriggerSourceChannel7</u> = 7, <u>PwrSnsrTriggerSourceChannel8</u> = 8, <u>PwrSnsrTriggerSourceChannel9</u> = 9, <u>PwrSnsrTriggerSourceChannel10</u> = 10, <u>PwrSnsrTriggerSourceChannel11</u> = 11, <u>PwrSnsrTriggerSourceChannel12</u> = 12, <u>PwrSnsrTriggerSourceChannel13</u> = 13, <u>PwrSnsrTriggerSourceChannel14</u> = 14, <u>PwrSnsrTriggerSourceChannel15</u> = 15, <u>PwrSnsrTriggerSourceChannel16</u> = 16, <u>PwrSnsrTriggerSourceIndependent</u> = 17 }
enum	<u>PwrSnsrUnitsEnum</u> { <u>PwrSnsrUnitsdBm</u> = 0, <u>PwrSnsrUnitswatts</u> = 1, <u>PwrSnsrUnitsvolts</u> = 2, <u>PwrSnsrUnitsDBV</u> = 3, <u>PwrSnsrUnitsDBMV</u> = 4, <u>PwrSnsrUnitsDBUV</u> = 5 }
enum	<u>PwrSnsrMarkerNumberEnum</u> { <u>PwrSnsrMarkerNumberMarker1</u> = 1, <u>PwrSnsrMarkerNumberMarker2</u> = 2 }
enum	<u>PwrSnsrBandwidthEnum</u> { <u>PwrSnsrBandwidthHigh</u> = 0, <u>PwrSnsrBandwidthLow</u> = 1 }

enum	<u>PwrSnsrFilterStateEnum</u> { <u>PwrSnsrFilterStateOff</u> = 0, <u>PwrSnsrFilterStateOn</u> = 1, <u>PwrSnsrFilterStateAuto</u> = 2 }
enum	<u>PwrSnsrPulseUnitsEnum</u> { <u>PwrSnsrPulseUnitsWatts</u> = 0, <u>PwrSnsrPulseUnitsVolts</u> = 1 }
enum	<u>PwrSnsrCondCodeEnum</u> { <u>PwrSnsrCondCodeMeasurementStopped</u> = -1, <u>PwrSnsrCondCodeError</u> = 0, <u>PwrSnsrCondCodeUnderrange</u> = 2, <u>PwrSnsrCondCodeOverrange</u> = 3, <u>PwrSnsrCondCodeNormal</u> = 1 }
enum	<u>PwrSnsrTriggerStatusEnum</u> { <u>PwrSnsrTriggerStatusStopped</u> = 0, <u>PwrSnsrTriggerStatusPretrig</u> = 1, <u>PwrSnsrTriggerStatusWaiting</u> = 2, <u>PwrSnsrTriggerStatusAcquiringNew</u> = 3, <u>PwrSnsrTriggerStatusAutoTrig</u> = 4, <u>PwrSnsrTriggerStatusFreerun</u> = 5, <u>PwrSnsrTriggerStatusTriggered</u> = 6, <u>PwrSnsrTriggerStatusRunning</u> = 7 }
enum	<u>PwrSnsrTermActionEnum</u> { <u>PwrSnsrTermActionStop</u> = 0, <u>PwrSnsrTermActionRestart</u> = 1, <u>PwrSnsrTermActionDecimate</u> = 2 }
enum	<u>PwrSnsrHoldoffModeEnum</u> { <u>PwrSnsrHoldoffModeNormal</u> = 1, <u>PwrSnsrHoldoffModeGap</u> = 2 }
enum	<u>PwrSnsrStatGatingEnum</u> { <u>PwrSnsrStatGatingFreeRun</u> = 0, <u>PwrSnsrStatGatingMarkers</u> = 1 }
enum	<u>PwrSnsrTrigOutModeEnum</u> {

	<pre> PwrSnsrTrigOutModeMioOff = 0, PwrSnsrTrigOutModeMioPullUp = 1, PwrSnsrTrigOutModeMioTtl0 = 2, PwrSnsrTrigOutModeMioTbRef = 3, PwrSnsrTrigOutModeMioSweepHigh = 4, PwrSnsrTrigOutModeMioSweepLow = 5, PwrSnsrTrigOutModeMioTrigHigh = 6, PwrSnsrTrigOutModeMioTrigLow = 7, PwrSnsrTrigOutModeMioMaster = 8, PwrSnsrTrigOutModeMioSlave = 9 } </pre>
enum	<pre> PwrSnsrMeasBuffGateEnum { PwrSnsrMeasBuffGateBurst = 0, PwrSnsrMeasBuffGateMarker = 1, PwrSnsrMeasBuffGateExtGate = 2, PwrSnsrMeasBuffGatePeriodic = 3, PwrSnsrMeasBuffGateExtTrig = 4 } </pre>
enum	<pre> PwrSnsrMeasBuffStartModeEnum { PwrSnsrMeasBuffStartModeImmediate = 1, PwrSnsrMeasBuffStartModeExternalEna ble = 2, PwrSnsrMeasBuffStartModeExternalStart = 3 } </pre>
enum	<pre> PwrSnsrMeasBuffStopReasonEnum { PwrSnsrMeasBuffStopReasonCountRea ched = 1, PwrSnsrMeasBuffStopReasonTimedOut = 2, PwrSnsrMeasBuffStopReasonBufferOve rran = 3, PwrSnsrMeasBuffStopReasonNone = 0 } </pre>
enum	<pre> PwrSnsrRdgsEnableFlag { PwrSnsrSequenceEnable = 1, PwrSnsrStartTimeEnable = 2, PwrSnsrDurationEnable = 4, PwrSnsrMinEnable = 8, PwrSnsrAvgEnable = 16, PwrSnsrMaxEnable = 32 } </pre>

enum	<pre> PwrSnsrErrorCodesEnum{ <u>PWR SNSR IO GENERAL</u> = - 2147204588, <u>PWR SNSR IO TIMEOUT</u> = -2147204587, <u>PWR SNSR MODEL NOT SUPPORTED</u> = -2147204586, <u>PWR SNSR INV PARAMETER</u> = - 1073807240, <u>PWR SNSR ERROR INVALID SESSION</u> <u>HANDLE</u> = -1074130544, <u>PWR SNSR ERROR STATUS NOT AVAI</u> <u>lable</u> = -1074134947, <u>PWR SNSR ERROR RESET FAILED</u> = - 1074134945, <u>PWR SNSR ERROR RESOURCE UNKN</u> <u>OWN</u> = -1074134944, <u>PWR SNSR ERROR ALREADY INITIALI</u> <u>ZED</u> = -1074134943, <u>PWR SNSR ERROR OUT OF MEMORY</u> = -1074134954, <u>PWR SNSR ERROR OPERATION PEND</u> <u>ING</u> = -1074134953, <u>PWR SNSR ERROR NULL POINTER</u> = - 1074134952, <u>PWR SNSR ERROR UNEXPECTED R</u> <u>ESPONSE</u> = -1074134951, <u>PWR SNSR ERROR NOT INITIALIZED</u> = -1074135011, <u>PWR SNSR LIBUSB ERROR IO</u> = -1, <u>PWR SNSR LIBUSB ERROR INVALID P</u> <u>ARAM</u> = -2, <u>PWR SNSR LIBUSB ERROR ACCESS</u> = -3, <u>PWR SNSR LIBUSB ERROR NO DEVIC</u> <u>E</u> = -4, <u>PWR SNSR LIBUSB ERROR NOT FOU</u> <u>ND</u> = -5, <u>PWR SNSR LIBUSB ERROR BUSY</u> = -6, <u>PWR SNSR LIBUSB ERROR TIMEOU</u> <u>T</u> = -7, <u>PWR SNSR LIBUSB ERROR OVERFLO</u> <u>W</u> = -8, <u>PWR SNSR LIBUSB ERROR PIPE</u> = -9, <u>PWR SNSR LIBUSB ERROR INTERRU</u> <u>PTED</u> = -10, </pre>
------	---

	<u>PWR SNSR LIBUSB ERROR NO MEM</u> = -11, <u>PWR SNSR LIBUSB ERROR NOT SUP</u> <u>PORTED</u> = -12, <u>PWR SNSR LIBUSB ERROR OTHER</u> = - 99 }
Functions	
EXPORT int	<u>PwrSnsr_SendSCPICommand</u> (SessionID Vi, const char *Command)
	Send a SCPI command to the instrument. More...
EXPORT int	<u>PwrSnsr_ReadSCPI</u> (SessionID Vi, int ValueBufferSize, long *ValueActualSize, char Value[], int Timeout)
	Read a SCPI string response from the instrument. More...
EXPORT int	<u>PwrSnsr_SendSCPIToNamedParser</u> (SessionID Vi, const char *name, const char *Command)
	Send a SCPI command to the instrument using a named SCPI parser. More...
EXPORT int	<u>PwrSnsr_ReadSCPIFromNamedParser</u> (SessionID Vi, const char *name, int ValueBufferSize, long *ValueActualSize, char Value[], int Timeout)
	Read a SCPI string response from the instrument. More...
EXPORT int	<u>PwrSnsr_FindResources</u> (const char *Delimiter, int ValBufferSize, char Val[])
	Returns a delimited string of available resources. These strings can be used in the initialize function to open a session to an instrument. More...

EXPORT int	PwrSnsr_GetMinimumSupportedFirmware (int *Version)
	Gets the minimum supported firmware as an integer. Format is YYYYMMDD. More...
EXPORT int	PwrSnsr_SendSCPIBytes (SessionID Vi, int CommandBufferSize, char Command[])
	Send a SCPI command as a byte array. More...
EXPORT int	PwrSnsr_ReadSCPIBytes (SessionID Vi, int ValueBufferSize, char Value[], long *ValueActualSize, int Timeout)
	Read a SCPI byte array response from the instrument. More...
EXPORT int	PwrSnsr_SetTimeout (SessionID Vi, long Milliseconds)
	Sets the time out in milliseconds for I/O. More...
EXPORT int	PwrSnsr_GetTimeout (SessionID Vi, long *Val)
	Returns the time out value for I/O in milliseconds. More...
EXPORT int	PwrSnsr_init (char *ResourceName, SessionID *Vi)
	Initialize a communication session with a supported USB power sensor. More...
EXPORT int	PwrSnsr_close (SessionID Vi)
	Closes the I/O session to the instrument. Driver methods and properties that access the instrument are not accessible after Close is called. More...
EXPORT int	PwrSnsr_GetError (SessionID Vi, int *ErrorCode, int ErrorDescriptionBufferSize, char ErrorDescription[])
	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. More...
EXPORT int	PwrSnsr_ClearError (SessionID Vi)
	This function clears the error code and error description for the given session. More...
EXPORT int	PwrSnsr_reset (SessionID Vi)
EXPORT int	PwrSnsr_self_test (SessionID Vi, int *TestResult)
	Performs an instrument self test, waits for the instrument to complete the test, and queries the instrument for the results. If the instrument passes the test, TestResult is 0. More...
EXPORT int	PwrSnsr_MeasurePower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return average power using a default instrument configuration in Modulated Mode and dBm units. Instrument remains stopped in Modulated Mode after a measurement. More...
EXPORT int	PwrSnsr_FetchCWPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the most recently acquired CW power. More...
EXPORT int	PwrSnsr_MeasureVoltage (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return average voltage using a default instrument configuration in Modulated Mode and volts units. Instrument remains stopped

	in Modulated Mode after a measurement. More...
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)
	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. More...
EXPORT int	PwrSnsr_ReadWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the waveform for this channel. Call FetchWaveform to obtain the waveforms for other channels. More...
EXPORT int	PwrSnsr_FetchWaveformMinMax (SessionID Vi, const char *Channel, int MinWaveformBufferSize, float MinWaveform[], int *MinWaveformActualSize, int MaxWaveformBufferSize, float MaxWaveform[], int *MaxWaveformActualSize, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Returns the previously acquired minimum and maximum waveforms for this specified channel. The acquisition must be made prior

	to calling this method. Call this method separately for each channel. More...
EXPORT int	PwrSnsr_FetchWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Returns a previously acquired waveform for this channel. The acquisition must be made prior to calling this method. Call this method separately for each channel. More...
EXPORT int	PwrSnsr_FetchPowerArray (SessionID Vi, const char *Channel, float *PulsePeak, PwrSnsrCondCodeEnum *PulsePeakValid, float *PulseCycleAvg, PwrSnsrCondCodeEnum *PulseCycleAvgValid, float *PulseOnAvg, PwrSnsrCondCodeEnum *PulseOnValid, float *IEEETop, PwrSnsrCondCodeEnum *IEEETopValid, float *IEEEBottom, PwrSnsrCondCodeEnum *IEEEBottomValid, float *Overshoot, PwrSnsrCondCodeEnum *OvershootValid, float *Droop, PwrSnsrCondCodeEnum *DroopValid)
	Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform. More...
EXPORT int	PwrSnsr_FetchTimeArray (SessionID Vi, const char *Channel, float *Frequency, PwrSnsrCondCodeEnum *FrequencyValid, float *Period, PwrSnsrCondCodeEnum *PeriodValid, float *Width, PwrSnsrCondCodeEnum *WidthValid, float *Offtime, PwrSnsrCondCodeEnum *OfftimeValid, float *DutyCycle, PwrSnsrCondCodeEnum *DutyCycleValid, float *Risetime, PwrSnsrCondCodeEnum *RisetimeValid, float *Falltime, PwrSnsrCondCodeEnum *FalltimeValid, float *EdgeDelay, PwrSnsrCondCodeEnum *EdgeDelayValid, float *Skew, PwrSnsrCondCodeEnum *SkewValid)

	Returns an array of the current automatic timing measurements performed on a periodic pulse waveform. More...
EXPORT int	PwrSnsr_FetchCWArray (SessionID Vi, const char *Channel, float *PeakAverage, PwrSnsrCondCodeEnum *PeakAverageValid, float *PeakMax, PwrSnsrCondCodeEnum *PeakMaxValid, float *PeakMin, PwrSnsrCondCodeEnum *PeakMinValid, float *PeakToAvgRatio, PwrSnsrCondCodeEnum *PeakToAvgRatioValid)
	Returns the current average, maximum, minimum powers or voltages and the peak-to-average ratio of the specified channel. Units are the same as the channel units. Note the peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units. More...
EXPORT int	PwrSnsr_FetchRiseTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line. More...
EXPORT int	PwrSnsr_FetchWidth (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line. More...
EXPORT int	PwrSnsr_FetchPulsePeak (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the peak amplitude during the pulse. More...

EXPORT int	PwrSnsr_FetchPulseOnAverage (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Average power of the ON portion of the pulse. More...
EXPORT int	PwrSnsr_FetchPulseCycleAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the average power of the entire pulse. More...
EXPORT int	PwrSnsr_FetchPRF (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency). More...
EXPORT int	PwrSnsr_FetchPeriod (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between two successive pulses. (Reciprocal of the Pulse RepetitionFrequency) More...
EXPORT int	PwrSnsr_FetchOvershoot (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units. More...
EXPORT int	PwrSnsr_FetchOfftime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulsewidth). More...

EXPORT int	PwrSnsr_FetchIEEETop (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse. More...
EXPORT int	PwrSnsr_FetchIEEEBottom (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns. More...
EXPORT int	PwrSnsr_FetchFallTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between the last signal crossing of the distal line to the last signalcrossing of the proximal line. More...
EXPORT int	PwrSnsr_FetchEdgeDelay (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform. More...
EXPORT int	PwrSnsr_FetchDutyCycle (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the ratio of the pulse on-time to off-time. More...
EXPORT int	PwrSnsr_GetTrigDelay (SessionID Vi, float *Delay)

	Return the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position. More...
EXPORT int	PwrSnsr_SetTrigDelay (SessionID Vi, float Delay)
	Sets the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position. More...
EXPORT int	PwrSnsr_GetTrigHoldoff (SessionID Vi, float *Holdoff)
	Return the trigger holdoff time in seconds. More...
EXPORT int	PwrSnsr_SetTrigHoldoff (SessionID Vi, float Holdoff)
	Sets the trigger holdoff time in seconds. More...
EXPORT int	PwrSnsr_GetTrigHoldoffMode (SessionID Vi, PwrSnsrHoldoffModeEnum *HoldoffMode)
	Returns the holdoff mode to normal or gap holdoff. More...
EXPORT int	PwrSnsr_SetTrigHoldoffMode (SessionID Vi, PwrSnsrHoldoffModeEnum HoldoffMode)
	Sets the holdoff mode to normal or gap holdoff. More...
EXPORT int	PwrSnsr_GetTrigLevel (SessionID Vi, float *Level)
	Return the trigger level for synchronizing data acquisition with a pulsed input signal. More...
EXPORT int	PwrSnsr_SetTrigLevel (SessionID Vi, float Level)
	Set the trigger level for synchronizing data acquisition with a pulsed input signal. More...

EXPORT int	<u>PwrSnsr_GetTrigMode</u> (SessionID Vi, <u>PwrSnsrTriggerModeEnum</u> *Mode)
	Return the trigger mode for synchronizing data acquisition with pulsed signals. <u>More...</u>
EXPORT int	<u>PwrSnsr_SetTrigMode</u> (SessionID Vi, <u>PwrSnsrTriggerModeEnum</u> Mode)
	Set the trigger mode for synchronizing data acquisition with pulsed signals. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetTrigPosition</u> (SessionID Vi, <u>PwrSnsrTriggerPositionEnum</u> *Position)
	Return the position of the trigger event on displayed sweep. <u>More...</u>
EXPORT int	<u>PwrSnsr_SetTrigPosition</u> (SessionID Vi, <u>PwrSnsrTriggerPositionEnum</u> Position)
	Set the position of the trigger event on displayed sweep. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetTrigSource</u> (SessionID Vi, <u>PwrSnsrTriggerSourceEnum</u> *Source)
	Set the signal the power meter monitors for a trigger. It can be channel external input, or independent. <u>More...</u>
EXPORT int	<u>PwrSnsr_SetTrigSource</u> (SessionID Vi, <u>PwrSnsrTriggerSourceEnum</u> Source)
	Get the signal the power meter monitors for a trigger. It can be channel external input, or independent. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetTrigStatus</u> (SessionID Vi, <u>PwrSnsrTriggerStatusEnum</u> *Status)
	The status of the triggering system. Update rate is controlled by FetchLatency setting. <u>More...</u>
EXPORT int	<u>PwrSnsr_SetFetchLatency</u> (SessionID Vi, int Latency)

	Set the period the library waits to update fetch measurements in ms. More...
EXPORT int	PwrSnsr_GetFetchLatency (SessionID Vi, int *Latency)
	Get the period the library waits to update fetch measurements in ms. More...
EXPORT int	PwrSnsr_GetTrigVernier (SessionID Vi, float *Vernier)
	Return the fine position of the trigger event on the power sweep. More...
EXPORT int	PwrSnsr_SetTrigVernier (SessionID Vi, float Vernier)
	Set the fine position of the trigger event on the power sweep. More...
EXPORT int	PwrSnsr_GetTrigSlope (SessionID Vi, PwrSnsrTriggerSlopeEnum *Slope)
	Return the trigger slope or polarity. More...
EXPORT int	PwrSnsr_SetTrigSlope (SessionID Vi, PwrSnsrTriggerSlopeEnum Slope)
	Sets the trigger slope or polarity. More...
EXPORT int	PwrSnsr_Clear (SessionID Vi)
	Clear all data buffers. Clears averaging filters to empty. More...
EXPORT int	PwrSnsr_InitiateAquisition (SessionID Vi)
	Starts a single measurement cycle when INITiate:CONTinuous is set to OFF. More...
EXPORT int	PwrSnsr_Status (SessionID Vi, PwrSnsrAcquisitionStatusEnum *Val)
	Returns whether an acquisition is in progress, complete, or if the status is unknown. More...

EXPORT int	PwrSnsr_SetInitiateContinuous (SessionID Vi, int InitiateContinuous)
	Set the data acquisition mode for single or free-run measurements. More...
EXPORT int	PwrSnsr_GetInitiateContinuous (SessionID Vi, int *InitiateContinuous)
	Get the data acquisition mode for single or free-run measurements. More...
EXPORT int	PwrSnsr_EnableCapturePriority (SessionID Vi, const char *Channel, int Enabled)
	Sets the 55 series power meter to a buffered capture mode and disables real time processing. More...
EXPORT int	PwrSnsr_GetEnabled (SessionID Vi, const char *Channel, int *Enabled)
	Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed. More...
EXPORT int	PwrSnsr_SetEnabled (SessionID Vi, const char *Channel, int Enabled)
	Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed. More...
EXPORT int	PwrSnsr_GetSerialNumber (SessionID Vi, const char *Channel, int SerialNumberBufferSize, char SerialNumber[])
	Gets the serial number of the sensor. More...
EXPORT int	PwrSnsr_GetChannelCount (SessionID Vi, int *Count)
	Get number of channels. More...

EXPORT int	PwrSnsr_GetUnits (SessionID Vi, const char *Channel, PwrSnsrUnitsEnum *Units)
	Get units for the selected channel. More...
EXPORT int	PwrSnsr_SetUnits (SessionID Vi, const char *Channel, PwrSnsrUnitsEnum Units)
	Set units for the selected channel. More...
EXPORT int	PwrSnsr_GetCurrentTemp (SessionID Vi, const char *Channel, double *CurrentTemp)
	Get current sensor internal temperature in degrees C. More...
EXPORT int	PwrSnsr_GetAverage (SessionID Vi, const char *Channel, int *Average)
	Get the number of traces averaged together to form the measurement result on the selected channel. More...
EXPORT int	PwrSnsr_SetAverage (SessionID Vi, const char *Channel, int Average)
	Set the number of traces averaged together to form the measurement result on the selected channel. More...
EXPORT int	PwrSnsr_GetBandwidth (SessionID Vi, const char *Channel, PwrSnsrBandwidthEnum *Bandwidth)
	Get the sensor video bandwidth for the selected sensor. More...
EXPORT int	PwrSnsr_SetBandwidth (SessionID Vi, const char *Channel, PwrSnsrBandwidthEnum Bandwidth)
	Set the sensor video bandwidth for the selected sensor. More...
EXPORT int	PwrSnsr_GetFilterState (SessionID Vi, const char *Channel, PwrSnsrFilterStateEnum *FilterState)

	Get the current setting of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_SetFilterState (SessionID Vi, const char *Channel, PwrSnsrFilterStateEnum FilterState)
	Set the current setting of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_GetFilterTime (SessionID Vi, const char *Channel, float *FilterTime)
	Get the current length of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_SetFilterTime (SessionID Vi, const char *Channel, float FilterTime)
	Set the current length of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_GetDistal (SessionID Vi, const char *Channel, float *Distal)
	Get the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition. More...
EXPORT int	PwrSnsr_SetDistal (SessionID Vi, const char *Channel, float Distal)
	Set the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition. More...
EXPORT int	PwrSnsr_GetEndGate (SessionID Vi, const char *Channel, float *EndGate)
	Get the point on a pulse, which is used to define the end of the pulse's active interval. More...
EXPORT int	PwrSnsr_SetEndGate (SessionID Vi, const char *Channel, float EndGate)
	Set the point on a pulse, which is used to define the end of the pulse's active interval. More...

EXPORT int	PwrSnsr_GetMesial (SessionID Vi, const char *Channel, float *Mesial)
	Get the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_SetMesial (SessionID Vi, const char *Channel, float Mesial)
	Set the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_GetProximal (SessionID Vi, const char *Channel, float *Proximal)
	Get the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_SetProximal (SessionID Vi, const char *Channel, float Proximal)
	Set the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_GetPulseUnits (SessionID Vi, const char *Channel, PwrSnsrPulseUnitsEnum *Units)
	Get the units for entering the pulse distal, mesial and proximal levels. More...
EXPORT int	PwrSnsr_SetPulseUnits (SessionID Vi, const char *Channel, PwrSnsrPulseUnitsEnum PwrSnsrPulseUnitsEnum)
	Set the units for entering the pulse distal, mesial and proximal levels. More...
EXPORT int	PwrSnsr_GetStartGate (SessionID Vi, const char *Channel, float *StartGate)
	Get the point on a pulse, which is used to define the beginning of the pulse's active

	interval. More...
EXPORT int	PwrSnsr_SetStartGate (SessionID Vi, const char *Channel, float StartGate)
	Set the point on a pulse, which is used to define the beginning of the pulse's active interval. More...
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)
	Query information associated with calibration factors. More...
EXPORT int	PwrSnsr_GetCalFactor (SessionID Vi, const char *Channel, float *CalFactor)
	Get the frequency calibration factor currently in use on the selected channel. More...
EXPORT int	PwrSnsr_SetCalFactor (SessionID Vi, const char *Channel, float CalFactor)
	Set the frequency calibration factor currently in use on the selected channel. More...
EXPORT int	PwrSnsr_GetFrequency (SessionID Vi, const char *Channel, float *Frequency)
	Get the RF frequency for the current sensor. More...
EXPORT int	PwrSnsr_SetFrequency (SessionID Vi, const char *Channel, float Frequency)
	Set the RF frequency for the current sensor, and apply the appropriate frequency calibration factor from the sensor internal table. More...

EXPORT int	PwrSnsr_GetOffsetdB (SessionID Vi, const char *Channel, float *OffsetdB)
	Get a measurement offset in dB for the selected sensor. More...
EXPORT int	PwrSnsr_SetOffsetdB (SessionID Vi, const char *Channel, float OffsetdB)
	Set a measurement offset in dB for the selected sensor. More...
EXPORT int	PwrSnsr_GetTempComp (SessionID Vi, const char *Channel, int *TempComp)
	Get the state of the peak sensor temperature compensation system. More...
EXPORT int	PwrSnsr_SetTempComp (SessionID Vi, const char *Channel, int TempComp)
	Set the state of the peak sensor temperature compensation system. More...
EXPORT int	PwrSnsr_GetTimebase (SessionID Vi, float *Timebase)
	Get the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 (or max timebase) sec in a 1-2-5 sequence,. More...
EXPORT int	PwrSnsr_SetTimebase (SessionID Vi, float Timebase)
	Set the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 sec (or max timebase) in a 1-2-5 sequence,. More...
EXPORT int	PwrSnsr_SetTimespan (SessionID Vi, float Timespan)
	Set the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence. More...

EXPORT int	PwrSnsr_GetTimespan (SessionID Vi, float *Timespan)
	Get the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence. More...
EXPORT int	PwrSnsr_GetMaxTimebase (SessionID Vi, float *MaxTimebase)
	Gets the maximum timebase setting available. More...
EXPORT int	PwrSnsr_FetchArrayMarkerPower (SessionID Vi, const char *Channel, float *AvgPower, PwrSnsrCondCodeEnum *AvgPowerCondCode, float *MaxPower, PwrSnsrCondCodeEnum *MaxPowerCondCode, float *MinPower, PwrSnsrCondCodeEnum *MinPowerCondCode, float *PkToAvgRatio, PwrSnsrCondCodeEnum *PkToAvgRatioCondCode, float *Marker1Power, PwrSnsrCondCodeEnum *Marker1PowerCondCode, float *Marker2Power, PwrSnsrCondCodeEnum *Marker2PowerCondCode, float *MarkerRatio, PwrSnsrCondCodeEnum *MarkerRatioCondCode)
	Returns an array of the current marker measurements for the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerAverage (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)
	For the specified marker, return the average power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerMax (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)

	Forthe specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerMin (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Forthe specified marker, return the minimum power or voltage at the marker. The units are the same as the specified channel. More...
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)
	Returns an array of the current marker measurements for the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerAverage (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Forthe specified marker, return the average power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerMax (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)

	For the specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerMin (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)
	For the specified marker, return the minimum power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalFilteredMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	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. More...
EXPORT int	PwrSnsr_FetchIntervalFilteredMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)

	Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalPkToAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units. More...
EXPORT int	PwrSnsr_ReadIntervalAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalFilteredMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	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. More...
EXPORT int	PwrSnsr_ReadIntervalFilteredMax (SessionID Vi, const char *Channel,

	<u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_ReadIntervalMax</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_ReadIntervalMin</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_ReadIntervalPkToAvg</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units. <u>More...</u>
EXPORT int	<u>PwrSnsr_FetchIntervalMaxAvg</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_FetchIntervalMinAvg</u> (SessionID Vi, const char *Channel,

	<u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_ReadIntervalMaxAvg</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_ReadIntervalMinAvg</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. <u>More...</u>
EXPORT int	<u>PwrSnsr_FetchMarkerDelta</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the difference between MK1 and MK2. The units will be the same as marker units. <u>More...</u>
EXPORT int	<u>PwrSnsr_FetchMarkerRatio</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units. <u>More...</u>
EXPORT int	<u>PwrSnsr_FetchMarkerRDelta</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *condCode, float *Val)
	Return the difference between MK2 and MK1. The units will be the same as marker units.

	More...
EXPORT int	PwrSnsr_FetchMarkerRRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units. More...
EXPORT int	PwrSnsr_ReadMarkerDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the difference between MK1 and MK2. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_ReadMarkerRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units. More...
EXPORT int	PwrSnsr_ReadMarkerRDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the difference between MK2 and MK1. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_ReadMarkerRRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units. More...

EXPORT int	PwrSnsr_ReadCWPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
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)
	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. More...
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)
	Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform. More...
EXPORT int	PwrSnsr_ReadTimeArray (SessionID Vi, const char *Channel, float *Frequency, PwrSnsrCondCodeEnum *FrequencyValid, float *Period, PwrSnsrCondCodeEnum

	<p>*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)</p>
	Returns an array of the current automatic timing measurements performed on a periodic pulse waveform. More...
EXPORT int	PwrSnsr_ReadDutyCycle (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the ratio of the pulse on-time to off-time. More...
EXPORT int	PwrSnsr_ReadEdgeDelay (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform. More...
EXPORT int	PwrSnsr_ReadFallTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the interval between the last signal crossing of the distal line to the last signal crossing of the proximal line. More...
EXPORT int	PwrSnsr_ReadIEEEBottom (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a

	pulse departs and to which it ultimately returns. More...
EXPORT int	PwrSnsr_ReadIEEETop (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse. More...
EXPORT int	PwrSnsr_ReadOfftime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulse width). More...
EXPORT int	PwrSnsr_ReadOvershoot (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units. More...
EXPORT int	PwrSnsr_ReadPeriod (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the interval between two successive pulses. More...
EXPORT int	PwrSnsr_ReadPRF (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency). More...
EXPORT int	PwrSnsr_ReadPulseCycleAvg (SessionID Vi, const char *Channel,

	<u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Returns the average power of the entire pulse. More...
EXPORT int	<u>PwrSnsr_ReadPulseOnAverage</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Average power of the ON portion of the pulse. More...
EXPORT int	<u>PwrSnsr_ReadPulsePeak</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Returns the peak amplitude during the pulse. More...
EXPORT int	<u>PwrSnsr_ReadRiseTime</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line. More...
EXPORT int	<u>PwrSnsr_ReadWidth</u> (SessionID Vi, const char *Channel, <u>PwrSnsrCondCodeEnum</u> *CondCode, float *Val)
	Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line. More...
EXPORT int	<u>PwrSnsr_GetHorizontalOffset</u> (SessionID Vi, const char *Channel, double *HorizontalOffset)
	Get the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative). More...

EXPORT int	PwrSnsr_SetHorizontalOffset (SessionID Vi, const char *Channel, double HorizontalOffset)
	Set the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative). More...
EXPORT int	PwrSnsr_GetHorizontalScale (SessionID Vi, const char *Channel, double *HorizontalScale)
	Get the statistical mode horizontal scale in dB/Div. More...
EXPORT int	PwrSnsr_SetHorizontalScale (SessionID Vi, const char *Channel, double HorizontalScale)
	Set the statistical mode horizontal scale in dB/Div. More...
int EXPORT	PwrSnsr_GetVerticalCenter (SessionID Vi, const char *Channel, float *VerticalCenter)
	Gets vertical center based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_SetVerticalCenter (SessionID Vi, const char *Channel, float VerticalCenter)
	Sets vertical center based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_GetVerticalScale (SessionID Vi, const char *Channel, float *VerticalScale)
	Gets vertical scale based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_SetVerticalScale (SessionID Vi, const char *Channel, float VerticalScale)
	Sets vertical scale based on current units: <arg> = (range varies by units) More...
EXPORT int	PwrSnsr_GetChannelByIndex (SessionID Vi, int BuffSize, char Channel[], int Index)

	Gets the channel name by zero index. Note: SCPI commands use a one-based index. More...
EXPORT int	PwrSnsr_FetchCCDFTrace (SessionID Vi, const char *Channel, int TraceBufferSize, float Trace[], int *TraceActualSize)
	Returns the points in the CCDF trace. More...
EXPORT int	PwrSnsr_StatModeReset (SessionID Vi, const char *Channel)
	Resets statistical capturing mode by clearing the buffers and restarting the acquisition timer. More...
EXPORT int	PwrSnsr_FetchStatMeasurementArray (SessionID Vi, const char *Channel, double *Pavg, PwrSnsrCondCodeEnum *PavgCond, double *Ppeak, PwrSnsrCondCodeEnum *PpeakCond, double *Pmin, PwrSnsrCondCodeEnum *PminCond, double *PkToAvgRatio, PwrSnsrCondCodeEnum *PkToAvgRatioCond, double *CursorPwr, PwrSnsrCondCodeEnum *CursorPwrCond, double *CursorPct, PwrSnsrCondCodeEnum *CursorPctCond, double *SampleCount, PwrSnsrCondCodeEnum *SampleCountCond, double *SecondsRun, PwrSnsrCondCodeEnum *SecondsRunCond)
	Returns an array of the current automatic statistical measurements performed on a sample population. More...
EXPORT int	PwrSnsr_FetchCCDFPower (SessionID Vi, const char *Channel, double Percent, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Return relative power (in dB) for a given percent on the CCDF plot. More...

EXPORT int	PwrSnsr_FetchCCDFPercent (SessionID Vi, const char *Channel, double Power, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Return relative power (in dB) for a given percent on the CCDF plot. More...
EXPORT int	PwrSnsr_GetCapture (SessionID Vi, const char *Channel, int *Capture)
	Get whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_SetCapture (SessionID Vi, const char *Channel, int Capture)
	Set whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_GetGating (SessionID Vi, const char *Channel, PwrSnsrStatGatingEnum *Gating)
	Get whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_SetGating (SessionID Vi, const char *Channel, PwrSnsrStatGatingEnum Gating)
	Set whether the statical capture is gated by markers or free-running. More...
EXPORT int	PwrSnsr_GetTermAction (SessionID Vi, const char *Channel, PwrSnsrTermActionEnum *TermAction)
	Get the termination action for statistical capturing. More...
EXPORT int	PwrSnsr_SetTermAction (SessionID Vi, const char *Channel, PwrSnsrTermActionEnum TermAction)
	Set the termination action for statistical capturing. More...

EXPORT int	PwrSnsr_GetTermCount (SessionID Vi, const char *Channel, double *TermCount)
	Get the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_SetTermCount (SessionID Vi, const char *Channel, double TermCount)
	Set the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_GetTermTime (SessionID Vi, const char *Channel, int *TermTime)
	Get the termination time in seconds for statistical capturing. After the time has elapsed, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_SetTermTime (SessionID Vi, const char *Channel, int TermTime)
	Set the termination time in seconds (1 - 3600) for statistical capturing. After the time has elapsed, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_GetCCDFTraceCount (SessionID Vi, const char *Channel, int *TraceCount)
	Get the number of points in the CCDF trace plot. More...
EXPORT int	PwrSnsr_SetCCDFTraceCount (SessionID Vi, const char *Channel, int TraceCount)
	Set the number of points (1 - 16384) in the CCDF trace plot. More...
EXPORT int	PwrSnsr_FetchCursorPercent (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, double *Val)

	Returns the percent CCDF at the cursor. More...
EXPORT int	PwrSnsr_FetchCursorPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Returns the power CCDF in dB at the cursor. More...
EXPORT int	PwrSnsr_GetPercentPosition (SessionID Vi, const char *Channel, double *PercentPosition)
	Get the cursor percent on the CCDF plot. More...
EXPORT int	PwrSnsr_SetPercentPosition (SessionID Vi, const char *Channel, double PercentPosition)
	Set the cursor percent on the CCDF plot. More...
EXPORT int	PwrSnsr_SetPowerPosition (SessionID Vi, const char *Channel, double PowerPosition)
	Set the cursor power in dB on the CCDF plot. More...
EXPORT int	PwrSnsr_GetPowerPosition (SessionID Vi, const char *Channel, double *PowerPosition)
	Get the cursor power in dB on the CCDF plot. More...
EXPORT int	PwrSnsr_GetAcqStatusArray (SessionID Vi, const char *Channel, int *SweepLength, double *SampleRate, double *SweepRate, double *SweepTime, double *StartTime, int *StatusWord)
	Returns data about the status of the acquisition system. More...
EXPORT int	PwrSnsr_GetDiagStatusArray (SessionID Vi, const char *Channel, float

	*DetectorTemp, float *CpuTemp, float *MioVoltage, float *VccInt10, float *VccAux18, float *Vcc50, float *Vcc25, float *Vcc33)
	Returns diagnostic data. More...
EXPORT int	PwrSnsr_GetMarkerTimePosition (SessionID Vi, int MarkerNumber, float *TimePosition)
	Get the time (x-axis-position) of the selected marker relative to the trigger. More...
EXPORT int	PwrSnsr_SetMarkerTimePosition (SessionID Vi, int MarkerNumber, float TimePosition)
	Set the time (x-axis-position) of the selected marker relative to the trigger. More...
EXPORT int	PwrSnsr_GetMarkerPixelPosition (SessionID Vi, int MarkerNumber, int *PixelPosition)
	Get the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive. More...
EXPORT int	PwrSnsr_SetMarkerPixelPosition (SessionID Vi, int MarkerNumber, int PixelPosition)
	Set the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive. More...
EXPORT int	PwrSnsr_GetManufactureDate (SessionID Vi, const char *Channel, int ManufactureDateBufferSize, char ManufactureDate[])
	Date the sensor was manufactured in the following format YYYYmmDD. More...
EXPORT int	PwrSnsr_GetImpedance (SessionID Vi, const char *Channel, float *Impedance)
	Input impedance of the sensor. More...

EXPORT int	<u>PwrSnsr_GetPeakPowerMax</u> (SessionID Vi, const char *Channel, float *PeakPowerMax)
	Maximum power level the sensor can measure. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetPeakPowerMin</u> (SessionID Vi, const char *Channel, float *PeakPowerMin)
	Minimum power level the sensor can measure. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetAttenuation</u> (SessionID Vi, const char *Channel, float *Attenuation)
	Attenuation in dB for the sensor. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetFactoryCalDate</u> (SessionID Vi, const char *Channel, int FactoryCalDateBufferSize, char FactoryCalDate[])
	The date (YYYYmmDD) the last time the sensor was calibrated at the factory. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetMinimumTrig</u> (SessionID Vi, const char *Channel, float *MinimumTrig)
	Minimum internal trigger level in dBm. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetMinFreqHighBandwidth</u> (SessionID Vi, const char *Channel, float *MinFreqHighBandwidth)
	Minimum frequency of RF the sensor can measure in high bandwidth. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetMaxFreqHighBandwidth</u> (SessionID Vi, const char *Channel, float *MaxFreqHighBandwidth)
	Maximum frequency carrier the sensor can measure in high bandwidth. <u>More...</u>
EXPORT int	<u>PwrSnsr_GetMinFreqLowBandwidth</u> (SessionID Vi, const char *Channel, float

		*MinFreqLowBandwidth)
		Minimum frequency carrier the sensor can measure in low bandwidth. More...
	EXPORT int	PwrSnsr_GetMaxFreqLowBandwidth (SessionID Vi, const char *Channel, float *MaxFreqLowBandwidth)
		Maximum frequency carrier the sensor can measure in low bandwidth. More...
	EXPORT int	PwrSnsr_GetFpgaVersion (SessionID Vi, const char *Channel, int ValBufferSize, char Val[])
		Get the sensor FPGA version. More...
	EXPORT int	PwrSnsr_GetExternalSkew (SessionID Vi, const char *Channel, float *External)
		Gets the skew in seconds for the external trigger. More...
	EXPORT int	PwrSnsr_SetExternalSkew (SessionID Vi, const char *Channel, float External)
		Sets the skew in seconds for the external trigger. More...
	EXPORT int	PwrSnsr_GetSlaveSkew (SessionID Vi, const char *Channel, float *SlaveSkew)
		Gets the skew in seconds for the slave trigger. More...
	EXPORT int	PwrSnsr_SetSlaveSkew (SessionID Vi, const char *Channel, float SlaveSkew)
		Sets the skew in seconds for the slave trigger. More...
	EXPORT int	PwrSnsr_GetInternalSkew (SessionID Vi, const char *Channel, float *InternalSkew)
		Gets the skew in seconds for the internal trigger. More...

EXPORT int	PwrSnsr_SetInternalSkew (SessionID Vi, const char *Channel, float InternalSkew)
	Sets the skew in seconds for the internal trigger. More...
EXPORT int	PwrSnsr_Zero (SessionID Vi, const char *Channel)
	Performs a zero offset null adjustment. More...
EXPORT int	PwrSnsr_ZeroQuery (SessionID Vi, const char *Channel, int *Val)
	Performs a zero offset null adjustment and returns true if successful. More...
EXPORT int	PwrSnsr_Abort (SessionID Vi)
	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. More...
EXPORT int	PwrSnsr_FetchExtendedWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize, int Count)
	When capture priority is enabled, returns up to 100000 points of trace data based on the current timebase starting at the current trigger delay point. More...
EXPORT int	PwrSnsr_GetTimePerPoint (SessionID Vi, const char *Channel, float *TimePerPoint)
	Get time spacing for each waveform point in seconds. More...
EXPORT int	PwrSnsr_GetSweepTime (SessionID Vi, const char *Channel, float *SweepTime)
	Get sweep time for the trace in seconds. More...

EXPORT int	PwrSnsr_GetChanTraceCount (SessionID Vi, const char *Channel, int *TraceCount)
	Get the number of points in the CCDF trace plot. More...
EXPORT int	PwrSnsr_GetTraceStartTime (SessionID Vi, const char *Channel, float *TraceStartTime)
	Get time offset (start time) of the trace in seconds. May be negative, indicating pre-trigger information. More...
EXPORT int	PwrSnsr_FetchDistal (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the distal level in the current channel units. More...
EXPORT int	PwrSnsr_FetchMesial (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the mesial level in the current channel units. More...
EXPORT int	PwrSnsr_FetchProximal (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the proximal level in the current channel units. More...
EXPORT int	PwrSnsr_FetchAllMultiPulse (SessionID Vi, const char *Channel, int PulseInfosSize, PulseInfo PulseInfos[], int *PulseInfosActualSize)
	Return all previously acquired multiple pulse measurements. The elements in the PulseInfos array correspond to pulses on the current trace from left to right (ascending time order). More...

EXPORT int	PwrSnsr_SetTrigOutMode (SessionID Vi, const char *Channel, int Mode)
	Sets the trigger out/mult io mode. Setting trigger mode overrides this command. More...
EXPORT int	PwrSnsr_SaveToMemoryChannel (SessionID Vi, const char *memChan, const char *ChannelName)
	Saves the given channel to a memory channel. If the memory channel does not exist, a new one is created. More...
EXPORT int	PwrSnsr_GetMemChanArchive (SessionID Vi, const char *memChan, int ValBufferSize, char Val[])
	Returns an XML document containing settings and readings obtained using the SaveToMemoryChannel method. More...
EXPORT int	PwrSnsr_LoadMemChanFromArchive (SessionID Vi, const char *memChan, const char *ArchiveContent)
	Loads the named memory channel using the given archive. If the memory channel does not exist, one is created. More...
EXPORT int	PwrSnsr_SaveUserCal (SessionID Vi, const char *Channel)
	Instructs power meter to save the value of fixed cal, zero, and skew values. More...
EXPORT int	PwrSnsr_ClearUserCal (SessionID Vi, const char *Channel)
	Resets the value of fixed cal, zero, and skew to factory defaults. More...
EXPORT int	PwrSnsr_GetIsAvgSensor (SessionID Vi, const char *Channel, int *IsAvgSensor)
	Retruns true if sensor is average responding (not peak detecting). More...

EXPORT int	PwrSnsr_GetIsAvailable (SessionID Vi, const char *Channel, int *IsAvailable)
	Returns true if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled. More...
EXPORT int	PwrSnsr_GetIsRunning (SessionID Vi, const char *Channel, int *IsRunning)
	Returns true if modulated/CW measurements are actively running. More...
EXPORT int	PwrSnsr_GetReadingPeriod (SessionID Vi, const char *Channel, float *ReadingPeriod)
	Returns the period (rate) in seconds per new filtered reading. More...
EXPORT int	PwrSnsr_GetBufferedAverageMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the average power measurements that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_AcquireMeasurements (SessionID Vi, double Timeout, int Count, PwrSnsrMeasBuffStopReasonEnum *StopReason, int *Val)
	Initiates new acquisition from the measurement buffer system (if acquisition is in the stopped state). Blocks until the number of measurements for each enabled channel is equal to count, or a time out has occurred. More...
EXPORT int	PwrSnsr_GetMaxMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the maximum power measurements that were captured during the last call to AcquireMeasurements. More...

EXPORT int	PwrSnsr_GetMinMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the minimum power measurements that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetDuration (SessionID Vi, float *Duration)
	Get the time duration samples are captured during each timed mode acquisition. More...
EXPORT int	PwrSnsr_SetDuration (SessionID Vi, float Duration)
	Set the duration samples are captured during each timed mode acquisition. More...
EXPORT int	PwrSnsr_GetSequenceNumbers (SessionID Vi, const char *Channel, int ValBufferSize, long long Val[], int *ValActualSize)
	Get the sequence number entries that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetStartTimes (SessionID Vi, const char *Channel, int ValBufferSize, double Val[], int *ValActualSize)
	Get the start time entries in seconds that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetDurations (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the duration entries in seconds that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_StartAcquisition (SessionID Vi)
	Starts measurement buffer acquisition. This method allows the user to send a command to the power meter to begin buffering

	measurements without waiting for all measurements to be completed. Alternately, you can call the AcquireReadings method to start buffering measurements and wait for them to be read from the meter. More...
EXPORT int	PwrSnsr_StopAcquisition (SessionID Vi)
	Sends a command to stop the measurement buffer from acquiring readings. More...
EXPORT int	PwrSnsr_ClearBuffer (SessionID Vi)
	Sends a command to the power meter to clear all buffered readings. This method does not clear cached measurements accessible through GetAverageMeasurements, etc. More...
EXPORT int	PwrSnsr_ClearMeasurements (SessionID Vi)
	Clears cached average, min, max, duration, start time, and sequence number measurements. More...
EXPORT int	PwrSnsr_GetMeasurementsAvailable (SessionID Vi, const char *Channel, int *Val)
	Get the number of measurement entries available that were captured during AcquireMeasurements(). More...
EXPORT int	PwrSnsr_SetPeriod (SessionID Vi, float Period)
	Set the period each timed mode acquisition (measurement buffer) is started. More...
EXPORT int	PwrSnsr_GetPeriod (SessionID Vi, float *Period)
	Get the period each timed mode acquisition (measurement buffer) is started. More...
EXPORT int	PwrSnsr_GetRdgsEnableFlag (SessionID Vi, int *Flag)

	Get the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements. More...
EXPORT int	PwrSnsr_SetRdgsEnableFlag (SessionID Vi, int Flag)
	Set the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetGateMode (SessionID Vi, PwrSnsrMeasBuffGateEnum *GateMode)
	Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. The gate signal may be internally or externally generated in several different ways. More...
EXPORT int	PwrSnsr_SetGateMode (SessionID Vi, PwrSnsrMeasBuffGateEnum GateMode)
	Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. More...
EXPORT int	PwrSnsr_GetStartMode (SessionID Vi, PwrSnsrMeasBuffStartModeEnum *StartMode)
	Get the mode used to start acquisition of buffer entries. More...
EXPORT int	PwrSnsr_SetStartMode (SessionID Vi, PwrSnsrMeasBuffStartModeEnum StartMode)
	Set the mode used to start acquisition of buffer entries. More...
EXPORT int	PwrSnsr_AdvanceReadIndex (SessionID Vi)
	Send a command to the meter to notify it the user is done reading and to advance the read index. More...

EXPORT int	PwrSnsr_QueryAverageMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered average power measurements. More...
EXPORT int	PwrSnsr_QueryStartTimes (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered start times in seconds. More...
EXPORT int	PwrSnsr_QuerySequenceNumbers (SessionID Vi, const char *Channel, int ValBufferSize, long long Val[], int *ValActualSize)
	Query the power meter for all buffered sequence numbers. More...
EXPORT int	PwrSnsr_QueryDurations (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered measurement durations in seconds. More...
EXPORT int	PwrSnsr_QueryMaxMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered maximum power measurements. More...
EXPORT int	PwrSnsr_QueryMinMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered minimum power measurements. More...
EXPORT int	PwrSnsr_GetWriteProtection (SessionID Vi, int *WriteProtection)
	Get whether the measurement buffer is set to overwrite members that have not been read by the user. More...

EXPORT int	PwrSnsr_GetTimedOut (SessionID Vi, int *TimedOut)
	Check if the last measurement buffer session timed out. More...
EXPORT int	PwrSnsr_GetSessionCount (SessionID Vi, int *SessionCount)
	Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...
EXPORT int	PwrSnsr_SetSessionCount (SessionID Vi, int SessionCount)
	Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...
EXPORT int	PwrSnsr_SetSessionTimeout (SessionID Vi, float Seconds)
	Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...
EXPORT int	PwrSnsr_GetReturnCount (SessionID Vi, int *ReturnCount)
	Get the return count for each measurement query. More...
EXPORT int	PwrSnsr_SetReturnCount (SessionID Vi, int ReturnCount)
	Set the return count for each measurement query. More...
EXPORT int	PwrSnsr_SetWriteProtection (SessionID Vi, int WriteProtection)
	Set whether to allow the measurement buffer to overwrite entries that have not been read by the user. More...

EXPORT int	PwrSnsr_GetOverRan (SessionID Vi, int *OverRan)
	Get flag indicating whether the power meter's internal buffer filled up before being emptied. More...
EXPORT int	PwrSnsr_GetBufferedMeasurementsAvailable (SessionID Vi, int *MeasurementsAvailable)
	Gets the number of measurements available in the power meter's internal buffer. Note: The number of readings that have been acquired may be more or less. More...
EXPORT int	PwrSnsr_GetMeasBuffEnabled (SessionID Vi, int *MeasBuffEnabled)
	Get whether the measurement buffer has been enabled. More...
EXPORT int	PwrSnsr_SetMeasBuffEnabled (SessionID Vi, int MeasBuffEnabled)
	Enable or disable the measurement buffer. Disabling the measurement buffer enables modulated/CW measurements. Conversely, enabling it disables modulated/CW measurements. More...
EXPORT int	PwrSnsr_ResetContinuousCapture (SessionID Vi)
	Sets a flag indicating to restart continuous capture. This method allows the user to restart continuous acquisition. Has no effect if ContinuousCapture is set to false. More...
EXPORT int	PwrSnsr_GetEndDelay (SessionID Vi, float *EndDelay)
	Get delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst. More...

EXPORT int	PwrSnsr_SetEndDelay (SessionID Vi, float EndDelay)
	Set delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst. More...
EXPORT int	PwrSnsr_GetStartQual (SessionID Vi, float *StartQual)
	Get the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst. More...
EXPORT int	PwrSnsr_SetStartQual (SessionID Vi, float StartQual)
	Set the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst. More...
EXPORT int	PwrSnsr_GetStartDelay (SessionID Vi, float *StartDelay)
	Get delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst. More...
EXPORT int	PwrSnsr_SetStartDelay (SessionID Vi, float StartDelay)
	Set delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst. More...
EXPORT int	PwrSnsr_GetEndQual (SessionID Vi, float *EndQual)
	Get the minimum amount of time power remains below the trigger point to be counted as the end of a burst. More...
EXPORT int	PwrSnsr_SetEndQual (SessionID Vi, float EndQual)
	Set the minimum amount of time power remains below the trigger point to be counted

	as the end of a burst. More...
EXPORT int	PwrSnsr_Write (SessionID Vi, const char *Channel, int DataBufferSize, unsigned char Data[])
	Write a byte array to the meter. More...
EXPORT int	PwrSnsr_ReadByteArray (SessionID Vi, const char *Channel, int Count, int ValBufferSize, unsigned char Val[], int *ValActualSize)
	Reads byte array from the meter. More...
EXPORT int	PwrSnsr_ReadControl (SessionID Vi, const char *Channel, int Count, int ValBufferSize, unsigned char Val[], int *ValActualSize)
	Reads a control transfer on the USB. More...
EXPORT int	PwrSnsr_SetContinuousCapture (SessionID Vi, int ContinuousCapture)
	Set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called. More...
EXPORT int	PwrSnsr_GetContinuousCapture (SessionID Vi, int *ContinuousCapture)
	Get whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called. More...
EXPORT int	PwrSnsr_GetModel (SessionID Vi, const char *Channel, int ModelBufferSize, char Model[])
	Gets the model of the meter connected to the specified channel. More...
EXPORT int	PwrSnsr_GetPeakHoldDecay (SessionID Vi, const char *Channel, int *EnvelopeAverage)

	Get the number of min/max traces averaged together to form the peak hold measurement results on the selected channel. More...
EXPORT int	PwrSnsr_GetPeakHoldTracking (SessionID Vi, const char *Channel, int *EnvelopeTracking)
	Returns whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent. More...
EXPORT int	PwrSnsr_SetPeakHoldTracking (SessionID Vi, const char *Channel, int EnvelopeTracking)
	Sets whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent. More...
EXPORT int	PwrSnsr_GetFirmwareVersion (SessionID Vi, const char *Channel, int FirmwareVersionBufferSize, char FirmwareVersion[])
	Returns the firmware version of the power meter associated with this channel. More...
EXPORT int	PwrSnsr_SetPeakHoldDecay (SessionID Vi, const char *Channel, int PeakHoldDecay)
	Set the number of min/max traces averaged together to form the peak hold measurement results on the selected channel. More...
EXPORT int	PwrSnsr_GetDongleSerialNumber (long *val)
	Get the hardware license serial number. More...
EXPORT int	PwrSnsr_GetExpirationDate (int *Date)
	Get the hardware license expiration date. More...

EXPORT int	PwrSnsr_GetNumberOfCals (long *val)
	Get the number of calibrations left on the license. More...
EXPORT int	PwrSnsr_IsLicenseDongleConnected (int *val)
	Get whether the hardware license dongle is connected. More...

Detailed Description

File containing all user-callable functions.

Typedef Documentation

◆ [PulseInfo](#)

typedef struct [PulseInfo](#) [PulseInfo](#)

Data structure containing pulse information.

◆ [PwrSnsrAcquisitionStatusEnum](#)

typedef enum [PwrSnsrAcquisitionStatusEnum](#) [PwrSnsrAcquisitionStatusEnum](#)

◆ [PwrSnsrBandwidthEnum](#)

typedef enum [PwrSnsrBandwidthEnum](#) [PwrSnsrBandwidthEnum](#)

Video bandwidth enumeration.

◆ [PwrSnsrCondCodeEnum](#)

typedef enum [PwrSnsrCondCodeEnum](#) [PwrSnsrCondCodeEnum](#)

Condition code indicating validity of the measurement.

◆ [PwrSnsrErrorCodesEnum](#)

typedef enum [PwrSnsrErrorCodesEnum](#) [PwrSnsrErrorCodesEnum](#)
Error codes

◆ [PwrSnsrFilterStateEnum](#)

typedef enum [PwrSnsrFilterStateEnum](#) [PwrSnsrFilterStateEnum](#)
Filter state enum.

◆ [PwrSnsrHoldoffModeEnum](#)

typedef enum [PwrSnsrHoldoffModeEnum](#) [PwrSnsrHoldoffModeEnum](#)
Trigger holdoff mode.

◆ [PwrSnsrMarkerNumberEnum](#)

typedef enum [PwrSnsrMarkerNumberEnum](#) [PwrSnsrMarkerNumberEnum](#)
Marker number enumeration.

◆ [PwrSnsrMeasBuffGateEnum](#)

typedef enum [PwrSnsrMeasBuffGateEnum](#) [PwrSnsrMeasBuffGateEnum](#)
Measurement buffer gate modes.

◆ [PwrSnsrMeasBuffStartModeEnum](#)

typedef enum [PwrSnsrMeasBuffStartModeEnum](#) [PwrSnsrMeasBuffStartModeEnum](#)
Measurement buffer start modes.

◆ [PwrSnsrMeasBuffStopReasonEnum](#)

typedef enum [PwrSnsrMeasBuffStopReasonEnum](#) [PwrSnsrMeasBuffStopReasonEnum](#)
Measurement buffer stop reason.

◆ [PwrSnsrPulseUnitsEnum](#)

typedef enum [PwrSnsrPulseUnitsEnum](#) [PwrSnsrPulseUnitsEnum](#)
Enum for pulse calculation units.

◆ [PwrSnsrRdgsEnableFlag](#)

typedef enum [PwrSnsrRdgsEnableFlag](#) [PwrSnsrRdgsEnableFlag](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

◆ [PwrSnsrStatGatingEnum](#)

typedef enum [PwrSnsrStatGatingEnum](#) [PwrSnsrStatGatingEnum](#)
Gating value for statistical capture.

◆ [PwrSnsrTermActionEnum](#)

typedef enum [PwrSnsrTermActionEnum](#) [PwrSnsrTermActionEnum](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

◆ [PwrSnsrTriggerModeEnum](#)

typedef enum [PwrSnsrTriggerModeEnum](#) [PwrSnsrTriggerModeEnum](#)
Trigger mode for synchronizing data acquisition with pulsed signals.

◆ [PwrSnsrTriggerPositionEnum](#)

typedef enum [PwrSnsrTriggerPositionEnum](#) [PwrSnsrTriggerPositionEnum](#)
Set or return the position of the trigger event on displayed sweep.

◆ [PwrSnsrTriggerSlopeEnum](#)

typedef enum [PwrSnsrTriggerSlopeEnum](#) [PwrSnsrTriggerSlopeEnum](#)
Values for edge trigger slope

◆ PwrSnsrTriggerSourceEnum

typedef enum [PwrSnsrTriggerSourceEnum](#) [PwrSnsrTriggerSourceEnum](#)
Trigger source used for synchronizing data acquisition.

◆ PwrSnsrTriggerStatusEnum

typedef enum [PwrSnsrTriggerStatusEnum](#) [PwrSnsrTriggerStatusEnum](#)
Trigger status of the acquisition system.

◆ PwrSnsrTrigOutModeEnum

typedef enum [PwrSnsrTrigOutModeEnum](#) [PwrSnsrTrigOutModeEnum](#)
Multi IO trigger out modes.

◆ PwrSnsrUnitsEnum

typedef enum [PwrSnsrUnitsEnum](#) [PwrSnsrUnitsEnum](#)
Units returned by channel measurements.

Enumeration Type Documentation

◆ PwrSnsrAcquisitionStatusEnum

enum [PwrSnsrAcquisitionStatusEnum](#)

Enumerator	
PwrSnsrAcqComplete	The meter has completed the acquisition..
PwrSnsrAcqInProgress	The meter is still acquiring data.
PwrSnsrAcqStatusUnknown	The meter cannot determine the status of the acquisition.

◆ PwrSnsrBandwidthEnum

enum [PwrSnsrBandwidthEnum](#)

Video bandwidth enumeration.

Enumerator	
PwrSnsrBandwidthHigh	High bandwidth.
PwrSnsrBandwidthLow	Low bandwidth.

◆ PwrSnsrCondCodeEnum

enum [PwrSnsrCondCodeEnum](#)

Condition code indicating validity of the measurement.

Enumerator	
PwrSnsrCondCodeMeasurementStopped	Measurement is STOPPED. Value returned is not updated.
PwrSnsrCondCodeError	Error return. Measurement is not valid.
PwrSnsrCondCodeUnderrange	An Over-range condition exists.
PwrSnsrCondCodeOvrange	An Under-range condition exists.
PwrSnsrCondCodeNormal	Normal return. No error.

◆ PwrSnsrErrorCodesEnum

enum [PwrSnsrErrorCodesEnum](#)

Error codes

Enumerator	
PWR_SNSR_IO_GENERAL	I/O error.
PWR_SNSR_IO_TIMEOUT	I/O timeout error.
PWR_SNSR_MODEL_NOT_SUPPORTED	Instrument model does not support this feature.
PWR_SNSR_INV_PARAMETER	Invalid parameter value
PWR_SNSR_ERROR_INVALID_SESSION_HANDLE	Session ID invalid.

PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE	Status not available.
PWR_SNSR_ERROR_RESET_FAILED	Reset failed.
PWR_SNSR_ERROR_RESOURCE_UNKNOWN	Unknown resource descriptor.
PWR_SNSR_ERROR_ALREADY_INITIALIZED	Session already initialized.
PWR_SNSR_ERROR_OUT_OF_MEMORY	Out of memory.
PWR_SNSR_ERROR_OPERATION_PENDING	Operation pending.
PWR_SNSR_ERROR_NULL_POINTER	Null pointer not allowed.
PWR_SNSR_ERROR_UNEXPECTED_RESPONSE	Unexpected response from the instrument.
PWR_SNSR_ERROR_NOT_INITIALIZED	Session not initialized.
PWR_SNSR_LIBUSB_ERROR_IO	Input/output error
PWR_SNSR_LIBUSB_ERROR_INVALID_PARAMETER	Invalid parameter
PWR_SNSR_LIBUSB_ERROR_ACCESS	Access denied (insufficient permissions)
PWR_SNSR_LIBUSB_ERROR_NO_DEVICE	No such device (it may have been disconnected)
PWR_SNSR_LIBUSB_ERROR_NOT_FOUND	Entity not found
PWR_SNSR_LIBUSB_ERROR_BUSY	Resource busy
PWR_SNSR_LIBUSB_ERROR_TIMEOUT	Operation timed out
PWR_SNSR_LIBUSB_ERROR_OVERFLOW	Overflow
PWR_SNSR_LIBUSB_ERROR_PIPE	Pipe error
PWR_SNSR_LIBUSB_ERROR_INTERRUPTED	System call interrupted (perhaps due to signal)
PWR_SNSR_LIBUSB_ERROR_NO_MEMORY	Insufficient memory
PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED	Operation not supported or unimplemented on this platform

PWR_SNSR_LIBUSB_ERROR_OTHER	Other error
-----------------------------	-------------

◆ PwrSnsrFilterStateEnum

enum [PwrSnsrFilterStateEnum](#)

Filter state enum.

Enumerator	
PwrSnsrFilterStateOff	Filter off.
PwrSnsrFilterStateOn	Filter on.
PwrSnsrFilterStateAuto	Automatically calculated filter.

◆ PwrSnsrHoldoffModeEnum

enum [PwrSnsrHoldoffModeEnum](#)

Trigger holdoff mode.

Enumerator	
PwrSnsrHoldoffModeNormal	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	Trigger will not arm again after the trigger conditions are satisfied and then the amount of time set for trigger holdoff.

◆ PwrSnsrMarkerNumberEnum

enum [PwrSnsrMarkerNumberEnum](#)

Marker number enumeration.

Enumerator	
PwrSnsrMarkerNumberMarker1	Marker 1

PwrSnsrMarkerNumberMarker2

Marker2

◆ PwrSnsrMeasBuffGateEnum

enum [PwrSnsrMeasBuffGateEnum](#)

Measurement buffer gate modes.

◆ PwrSnsrMeasBuffStartModeEnum

enum [PwrSnsrMeasBuffStartModeEnum](#)

Measurement buffer start modes.

◆ PwrSnsrMeasBuffStopReasonEnum

enum [PwrSnsrMeasBuffStopReasonEnum](#)

Measurement buffer stop reason.

◆ PwrSnsrPulseUnitsEnum

enum [PwrSnsrPulseUnitsEnum](#)

Enum for pulse calculation units.

Enumerator	
PwrSnsrPulseUnitsWatts	Calculates distal, mesial, and proximal using watts.
PwrSnsrPulseUnitsVolts	Calculates distal, mesial, and proximal using volts.

◆ PwrSnsrRdgsEnableFlag

enum [PwrSnsrRdgsEnableFlag](#)

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

Enumerator

PwrSnsrSequenceEnable	Enable sequence array capture.
PwrSnsrStartTimeEnable	Enable start time array capture.
PwrSnsrDurationEnable	Enable duration array capture.
PwrSnsrMinEnable	Enable min measurement array capture.
PwrSnsrAvgEnable	Enable average measurement capture.
PwrSnsrMaxEnable	Enable max measurement capture.

◆ PwrSnsrStatGatingEnum

enum [PwrSnsrStatGatingEnum](#)
Gating value for statistical capture.

Enumerator	
PwrSnsrStatGatingFreeRun	No gating.
PwrSnsrStatGatingMarkers	Gating is constrained to the portion of the the trace between the markers.

◆ PwrSnsrTermActionEnum

enum [PwrSnsrTermActionEnum](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

Enumerator	
PwrSnsrTermActionStop	Stop accumulating samples and hold the result.
PwrSnsrTermActionRestart	Clear the CCDF and begin a new one.
PwrSnsrTermActionDecimate	Divide all sample bins by 2 and continue.

◆ PwrSnsrTriggerModeEnum

enum [PwrSnsrTriggerModeEnum](#)

Trigger mode for synchronizing data acquisition with pulsed signals.

Enumerator	
PwrSnsrTriggerModeNormal	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	The power meter automatically triggers if the configured trigger does not occur within the meter's timeout period.
PwrSnsrTriggerModeAutoLevel	The power meter automatically adjusts the trigger level the trigger level to halfway between the highest and lowest power levels detected.

◆ PwrSnsrTriggerPositionEnum

enum [PwrSnsrTriggerPositionEnum](#)

Set or return the position of the trigger event on displayed sweep.

Enumerator	
PwrSnsrTriggerPositionLeft	Left trigger position.
PwrSnsrTriggerPositionMiddle	Middle trigger position.
PwrSnsrTriggerPositionRight	Right trigger position.

◆ PwrSnsrTriggerSlopeEnum

enum [PwrSnsrTriggerSlopeEnum](#)

Values for edge trigger slope

Enumerator	
PwrSnsrTriggerSlopePositive	A negative (falling) edge passing through the trigger level triggers the power meter.

PwrSnsrTriggerSlopeNegative	A positive (rising) edge passing through the trigger level triggers the power meter.
-----------------------------	--

◆ PwrSnsrTriggerSourceEnum

enum [PwrSnsrTriggerSourceEnum](#)

Trigger source used for synchronizing data acquisition.

Enumerator	
PwrSnsrTriggerSourceChannel1	Channel 1
PwrSnsrTriggerSourceExternal	EXT setting uses the signal applied to the rear MULTI I/O connector.
PwrSnsrTriggerSourceChannel2	Channel 2
PwrSnsrTriggerSourceChannel3	Channel 3
PwrSnsrTriggerSourceChannel4	Channel 4
PwrSnsrTriggerSourceChannel5	Channel 5
PwrSnsrTriggerSourceChannel6	Channel 6
PwrSnsrTriggerSourceChannel7	Channel 7
PwrSnsrTriggerSourceChannel8	Channel 8
PwrSnsrTriggerSourceChannel9	Channel 9
PwrSnsrTriggerSourceChannel10	Channel 10
PwrSnsrTriggerSourceChannel11	Channel 11
PwrSnsrTriggerSourceChannel12	Channel 12
PwrSnsrTriggerSourceChannel13	Channel 13
PwrSnsrTriggerSourceChannel14	Channel 14
PwrSnsrTriggerSourceChannel15	Channel 15
PwrSnsrTriggerSourceChannel16	Channel 16

PwrSnsrTriggerSourceIndependent	Sets each sensor in a measurment group to use its own internal trigger.
---------------------------------	---

◆ PwrSnsrTriggerStatusEnum

enum [PwrSnsrTriggerStatusEnum](#)

Trigger status of the acquisition system.

Enumerator	
PwrSnsrTriggerStatusStopped	Acquisition is stopped.
PwrSnsrTriggerStatusPretrig	Aquiring data and waiting for the pre-trigger to be satisfied.
PwrSnsrTriggerStatusWaiting	Meter is armed and waiting for trigger event.
PwrSnsrTriggerStatusAcquiringNew	Acquiring new data.
PwrSnsrTriggerStatusAutoTrig	Meter is autotriggering.
PwrSnsrTriggerStatusFreerun	Trigger is in free-run mode.
PwrSnsrTriggerStatusTriggered	Meter is currently triggered.
PwrSnsrTriggerStatusRunning	Acquisition is running.

◆ PwrSnsrTrigOutModeEnum

enum [PwrSnsrTrigOutModeEnum](#)

Multi IO trigger out modes.

◆ PwrSnsrUnitsEnum

enum [PwrSnsrUnitsEnum](#)

Units returned by channel measurements.

Enumerator	
PwrSnsrUnitsdBm	dBm
PwrSnsrUnitswatts	Watts

PwrSnsrUnitsvolts	Volts
PwrSnsrUnitsDBV	dBV
PwrSnsrUnitsDBMV	dBmV
PwrSnsrUnitsDBUV	dBuV

Function Documentation

◆ PwrSnsr_Abort()

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

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.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_AcquireMeasurements()

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

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

Parameters

Vi

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

Timeout

Maximum time in seconds to continue acquiring samples. Negative values will be treated as infinite.

Count

Number of samples to acquire.

StopReason

Reason acquisition stopped.

Val

Number of samples acquired.

Returns

Success (0) or error code.

◆ PwrSnsr_AdvanceReadIndex()

```
EXPORT int
PwrSnsr_AdvanceReadIndex
(
    SessionID Vi
)
```

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

Parameters

Vi

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

Returns

Success (0) or error code.

◆ PwrSnsr_Clear()

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

Clear all data buffers. Clears averaging filters to empty.

Parameters

Vi

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

Returns

Success (0) or error code.

◆ PwrSnsr_ClearBuffer()

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

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

Parameters**Vi**

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

Returns

Success (0) or error code.

◆ PwrSnsr_ClearError()

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

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

Parameters**Vi**

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

Returns

Success (0) or error code.

◆ PwrSnsr_ClearMeasurements()

```
EXPORT int
PwrSnsr_ClearMeasurements (          SessionID  Vi          )
```

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

Parameters

Vi

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

Returns

Success (0) or error code.

◆ PwrSnsr_ClearUserCal()

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

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

Parameters

Vi

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

Channel

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

Returns

Success (0) or error code.

◆ PwrSnsr_close()

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

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

Parameters

Vi

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

Returns

Success (0) or error code.

◆ PwrSnsr_EnableCapturePriority()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchAllMultiPulse()

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

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

Parameters

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

Channel

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

PulseInfosSize

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()

```
EXPORT int
PwrSnsr_FetchArray
MarkerPower (
```

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

```
)
```

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

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel	identifies a particular instrument session. 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()

```
EXPORT int
PwrSnsr_FetchCCDF
Percent          (
                                SessionID          Vi,
                                const char *        Channel,
                                double              Power,
                                PwrSnsrCondCode    CondCode,
                                Enum *             Val,
                                double *
                                )
```

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

Parameters

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

Power Relative power in dB
CondCode Condition code for the measurement.
Val Percent measurement at power.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchCCDFPower()**

```
EXPORT int
PwrSnsr_FetchCCDF
Power          (          SessionID          Vi,
                  const char *          Channel,
                  double          Percent,
                  PwrSnsrCondCode          CondCode,
                  Enum *          Val
                  double *
                  )
```

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

Parameters

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

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

Percent Statistical percent to retrieve power from.

CondCode Condition code for the measurement.

Val relative power at percent.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchCCDFTrace()**

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

)
Returns the points in the CCDF trace.

Parameters

Vi

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

Channel

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

TraceBufferSize

Trace

TraceActualSize

Returns

Success (0) or error code.

◆ PwrSnsr_FetchCursorPercent()

```
EXPORT int
PwrSnsr_FetchCurso
rPercent      (
```

```
SessionID      Vi,
const char *   Channel,
PwrSnsrCondCode
Enum *         CondCode,
double *       Val
)
```

Returns the percent CCDF at the cursor.

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.

Val

Returns

Success (0) or error code.

◆ PwrSnsr_FetchCursorPower()

```

EXPORT int
PwrSnsr_FetchCurso
rPower          (
    SessionID    Vi,
    const char *  Channel,
    PwrSnsrCondCode
Enum *          CondCode,
    double *      Val
)

```

Returns the power CCDF in dB at the cursor.

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.

Val

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchCWArray\(\)](#)

```

EXPORT int
PwrSnsr_FetchCWAr
ray          (
    SessionID    Vi,
    const char *  Channel,
    float *       PeakAverage,
    PwrSnsrCondCode
Enum *          PeakAverageValid,
    float *       PeakMax,
    PwrSnsrCondCode
Enum *          PeakMaxValid,
    float *       PeakMin,
    PwrSnsrCondCode
Enum *          PeakMinValid,
    float *       PeakToAvgRatio,
    PwrSnsrCondCode
Enum *          PeakToAvgRatioValid
)

```

Returns the current average, maximum, minimum powers or voltages and the

peak-to-average ratio of the specified channel. Units are the same as the channel units. Note the peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

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

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

Returns the most recently acquired CW power.

Parameters

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

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchDistal()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.

Val

Detected power of the distal level in the current channel units.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchDutyCycle()**

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

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

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

IsValid

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

Val

Condition code.

Returns

Measurement return value.

Success (0) or error code.

◆ PwrSnsr_FetchEdgeDelay()

```

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

```

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

Parameters**Vi**

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

Channel

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

IsValid**Val****Returns**

Success (0) or error code.

◆ PwrSnsr_FetchExtendedWaveform()

```

EXPORT int
PwrSnsr_FetchExten
dedWaveform      (
                                SessionID      Vi,
                                const char *   Channel,
                                int             WaveformArrayBuffer
                                                Size,

```

float	WaveformArray[],
int *	WaveformArrayActual
int	Size,
	Count

)

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

Parameters

Vi

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

Channel

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

WaveformArrayBufferSize

Number of elements in the WaveformArray buffer

WaveformArray

Waveform buffer.

WaveformArrayActualSize

Number of elements updated with data.

Count

Number of points to capture.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchFallTime()

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

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

Parameters

Vi

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIEEEBottom()

```

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

```

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

Parameters**Vi**

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIEEETop()

```

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

```

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

Parameters**Vi**

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalAvg()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalFilteredMax()**

```
EXPORT int
PwrSnsr_FetchIntervalFilteredMax (
    SessionID          Vi,
```

```

const char *      Channel,
PwrSnsrCondCode
Enum *            CondCode,
float *           Val

```

```
)
```

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIntervalFilteredMin()

```

EXPORT int
PwrSnsr_FetchInterv
alFilteredMin      (

```

```

SessionID          Vi,
const char *        Channel,
PwrSnsrCondCode
Enum *              CondCode,
float *             Val

```

```
)
```

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.

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalMax\(\)](#)

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalMaxAvg\(\)](#)

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalMin()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalMinAvg()**

```
EXPORT int
PwrSnsr_FetchIntervalMinAvg (
    SessionID          Vi,
```

alMinAvg

const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * CondCode,
float * Val

)

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalPkToAvg\(\)](#)

EXPORT int
PwrSnsr_FetchIntervalPkToAvg (

SessionID Vi,
const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * CondCode,
float * Val

)

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerAverage()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerDelta()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerMax()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerMin()

```

EXPORT int
PwrSnsr_FetchMarke
rMin          (
                SessionID          Vi,
                const char *       Channel,
                int                 Marker,
                PwrSnsrCondCode
Enum *       IsValid,
                float *            Val
            )

```

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

Parameters

Vi

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

Channel

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

Marker

Marker number.

IsValid

Val

measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerRatio()

```

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

```

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

Parameters

Vi

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

Channel

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

CondCodeCondition code for the measurement.
Condition code.**Val**

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerRDelta()

```

EXPORT int
PwrSnsr_FetchMarke
rDelta          (
                                SessionID      Vi,
                                const char *    Channel,
                                PwrSnsrCondCode
                                Enum *         condCode,
                                float *         Val
                                )

```

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

Parameters**Vi**

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

Channel

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

CondCodeCondition code for the measurement.
Condition code.**Val**

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerRRatio()

```

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

```

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMesial()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchOfftime()

```

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

```

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

Parameters

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

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchOvershoot\(\)](#)

```

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

```

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

Parameters

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

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchPeriod()**

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

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

Parameters**Vi**

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchPowerArray()**

```
EXPORT int
PwrSnsr_FetchPower
Array (
    SessionID          Vi,
    const char *       Channel,
    float *            PulsePeak,
    PwrSnsrCondCode
Enum *              PulsePeakValid,
    float *            PulseCycleAvg,
    PwrSnsrCondCode
Enum *              PulseCycleAvgValid,
    float *            PulseOnAvg,
```

<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	PulseOnValid,
float *	IEEETop,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	IEEETopValid,
float *	IEEEBottom,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	IEEEBottomValid,
float *	Overshoot,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	OvershootValid,
float *	Droop,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	DroopValid

)

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

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

Note the pulse overshoot is returned in dB for logarithmic channel units, and percent for all other units. Also, the pulse ?ON interval used for peak

and average calculations is defined by the SENSE:PULSE:STARTGT and :ENDGT time gating settings.

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

Parameters

Vi

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

Channel

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

PulsePeak

The peak amplitude during the pulse.

PulsePeakValid

Condition code.

PulseCycleAvg	Average cycle amplitude.
PulseCycleAvgValid	Condition code.
PulseOnAvg	Average power of the ON portion of the pulse.
PulseOnValid	Condition code.
IEEETop	The IEEE-defined top line, i.e. the portion of a pulse waveform, which represents the second nominal state of a pulse.
IEEETopValid	Condition code.
IEEEBottom	The IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.
IEEEBottomValid	Condition code.
Overshoot	The difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.
OvershootValid	Condition code.
Droop	Pulse droop.
DroopValid	Condition code.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchPRF()

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

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

Parameters

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

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchProximal()

```

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

```

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.

Val

Detected power of the proximal level in the current channel units.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchPulseCycleAvg()

```

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

```

Returns the average power of the entire pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchPulseOnAverage()

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

Average power of the ON portion of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchPulsePeak()

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


float * Val

)

Returns the peak amplitude during the pulse.

Parameters

Vi

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchRiseTime()

```
EXPORT int
PwrSnsr_FetchRiseTime
me (
```

SessionID Vi,
const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * IsValid,
float * Val

)

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

Parameters

Vi

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

Channel

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

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchStatMeasurementArray()

```
EXPORT int
PwrSnsr_FetchStatM
easurementArray (
```

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

```
)
```

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.

Parameters

Vi

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

Channel	Channel number. For single instruments, set this to "CH1".
Pavg	Long term average power in channel units.
PavgCond	Condition code.
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()

```

EXPORT int
PwrSnsr_FetchTimeA
rray          (
    SessionID
    const char *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    Vi,
    Channel,
    Frequency,
    FrequencyValid,
    Period,
    PeriodValid,
    Width,
    WidthValid,
    Offtime,
    OfftimeValid,

```

float *	DutyCycle,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	DutyCycleValid,
float *	Risetime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	RisetimeValid,
float *	Falltime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	FalltimeValid,
float *	EdgeDelay,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	EdgeDelayValid,
float *	Skew,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	SkewValid

)

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

Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window. Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions. It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements. Note that the timing of the edge transitions is defined by the settings of the SENSE:PULSE:DISTal, :MESIal and :PROXimal settings; see the descriptions Forthose commands. Units are the same as the channel's units.

Parameters

Vi

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

Channel	Channel number. For single instruments, set this to "CH1".
Frequency	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid	Condition code.
Period	The interval between two successive pulses.
PeriodValid	Condition code.
Width	The interval between the first and second signal crossings of the mesial line.
WidthValid	Condition code.
Offtime	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid	Condition code.
DutyCycle	The ratio of the pulse on-time to period.
DutyCycleValid	Condition code.
Risetime	The interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.
RisetimeValid	Condition code.
Falltime	The interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.
FalltimeValid	Condition code.
EdgeDelay	Time offset from the trigger reference to the first mesial transition level of either slope on the waveform.
EdgeDelayValid	Condition code.
Skew	The trigger offset between the assigned trigger channel and this channel.
SkewValid	Condition code.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchWaveform()

```
EXPORT int
PwrSnsr_FetchWaveform (
```

```
SessionID
const char *
int
```

```
Vi,
Channel,
WaveformArrayBuffer
Size,
```

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

```

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchWaveformMinMax()

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

```

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FetchWidth()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_FindResources()

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

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

Parameters

Delimiter	The string used to delimit the list of resources ie. " ", " ", ":", etc.
ValBufferSize	Number of elements in Val.
Val	The return string.

Returns

Success (0) or error code.

◆ PwrSnsr_GetAcqStatusArray()

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



```

double *      SampleRate,
double *      SweepRate,
double *      SweepTime,
double *      StartTime,
int *         StatusWord

```

```

)
```

Returns data about the status of the acquisition system.

Parameters

Vi

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

Channel

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

SweepLength

Returns the number of points in the trace.

SampleRate

Returns the sample rate.

SweepRate

Returns the number of triggered sweeps per second.

SweepTime

Returns the sweep time for the trace.

StartTime

Returns the start time relative to the trigger.

StatusWord

Internal use - acquisition system status word.

Returns

Success (0) or error code.

◆ PwrSnsr_GetAttenuation()

```

EXPORT int
PwrSnsr_GetAttenuati
on          (          SessionID      Vi,
                  const char *      Channel,
                  float *            Attenuation
          )

```

Attenuation in dB for the sensor.

Parameters

Vi

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

Channel

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

Attenuation

Returns

Success (0) or error code.

◆ **PwrSnsr_GetAverage()**

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

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

Parameters**Vi**

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

Channel

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

Average**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetBandwidth()**

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

Get the sensor video bandwidth for the selected sensor.

Parameters**Vi**

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

Channel

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

Bandwidth

Returns

Success (0) or error code.

◆ `PwrSnsr_GetBufferedAverageMeasurements()`

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

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

Parameters**Vi**

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

Channel

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

ValBufferSize

Buffer size of Val.

Val

Array of average measurements.

ValActualSize

Actual size of Val.

Returns

Success (0) or error code.

◆ `PwrSnsr_GetBufferedMeasurementsAvailable()`

```
EXPORT int
PwrSnsr_GetBuffered
MeasurementsAvailab
le          (          SessionID          Vi,
               int *          MeasurementsAvailab
               le
          )
```

Gets the number of measurements available in the power meter's internal buffer. Note: The number of readings that have been acquired may be more or less.

Parameters**Vi**

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

MeasurementsAvailable

The number of measurements available in the 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()**

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

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

Parameters**Vi**

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

Channel

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

CalFactor**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetCalFactors()**

```
EXPORT int
PwrSnsr_GetCalFact
ors          (          SessionID          Vi,
                    const char *          Channel,
                    float *          MaxFrequency,
                    float *          MinFrequency,
                    int          FrequencyListBufferSize,
                    )
```

float	FrequencyList[],
int *	FrequencyListActualSize,
int	CalFactorListBufferSize,
float	CalFactorList[],
int *	CalFactorListActualSize,
<u>PwrSnsrBandwidthE</u>	<u>num</u>
	Bandwidth

)

Query information associated with calibration factors.

Parameters

Vi

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

Channel

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

MaxFrequency

Maximum RF frequency measureable by this channel.

MinFrequency

Minimum RF frequency measureable by this channel.

FrequencyListBufferSize

Number of elements in FrequencyList.

FrequencyList

List of frequencies correlated to the cal factors.

FrequencyListActualSize

Actual number of elements returned in FrequencyList.

CalFactorListBufferSize

Number of elements in CalFactorList.

CalFactorList

List of cal factors correlated to the frequencies.

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\(\)](#)

EXPORT int

PwrSnsr_GetCapture (

SessionID

const char *

Vi,

Channel,

int *

Capture

)

Get whether statistical capture is enabled.

Parameters

Vi

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

Channel

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

Capture

Returns

Success (0) or error code.

◆ PwrSnsr_GetCCDFTraceCount()

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

Get the number of points in the CCDF trace plot.

Parameters

Vi

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

Channel

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

TraceCount

Returns

Success (0) or error code.

◆ PwrSnsr_GetChannelByIndex()

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

int

Index

)

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

Parameters

Vi

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

Channel

Buffer size for Channel

Channel

Channel number buffer

Index

the index of the channel

Returns

Success (0) or error code.

◆ PwrSnsr_GetChannelCount()

EXPORT int

PwrSnsr_GetChannel

Count

(

SessionID

Vi,

int *

Count

)

Get number of channels.

Parameters

Vi

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

Count

Number of channels

Returns

Success (0) or error code.

◆ PwrSnsr_GetChanTraceCount()

EXPORT int

PwrSnsr_GetChanTr

aceCount

(

SessionID

Vi,

const char *

Channel,

int *

TraceCount

)

Get the number of points in the CCDF trace plot.

Parameters**Vi**

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

Channel

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

TraceCount

The number of points in the CCDF trace plot.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetContinuousCapture()**

```
EXPORT int
PwrSnsr_GetContinu
ousCapture      (      SessionID      Vi,
                    int *              ContinuousCapture
                )
```

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

Parameters**Vi**

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

ContinuousCapture

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

Returns

Success (0) or error code.

◆ **PwrSnsr_GetCurrentTemp()**

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

Get current sensor internal temperature in degrees C.

Parameters**Vi**

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

Channel

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

CurrentTemp**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetDiagStatusArray()**

```
EXPORT int
PwrSnsr_GetDiagStat
usArray      (
```

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

```
)
```

Returns diagnostic data.

Parameters**Vi**

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

Channel

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

DetectorTemp

Temperature in degrees C at the RF detector.

CpuTemp

Temperature of the CPU in degrees C.

MioVoltage

Voltage at the Multi I/O port.

VccInt10

Vcc 10 voltage.

VccAux18

Vcc Aux 18 voltage.

Vcc50

Vcc 50 voltage.

Vcc25

Vcc 25 voltage.

Vcc33

Vcc 33 voltage.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetDistal()**

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

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

Parameters**Vi**

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

Channel

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

Distal**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetDongleSerialNumber()**

```
EXPORT int
PwrSnsr_GetDongleSerialN
umber (
    long *
    val
)
```

Get the hardware license serial number.

Parameters**val**

Serial number of the license dongle

Returns

Success (0) or error code.

◆ **PwrSnsr_GetDuration()**

```
EXPORT int
PwrSnsr_GetDuration (
    SessionID
    Vi,
```

float *

Duration

)

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

Parameters

Vi

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

Duration

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetDurations()

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

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

Parameters

Vi

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

Channel

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

ValBufferSize

Size of the buffer.

Val

Array of measurement durations in seconds.

ValActualSize

Actual size of the returned buffer.

Returns

Success (0) or error code.

◆ PwrSnsr_GetEnabled()

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

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

Parameters

Vi

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

Channel

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

Enabled

Boolean. 1 for enabled; 0 for disabled.

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndDelay()

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

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

Parameters

Vi

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

EndDelay

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndGate()

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

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

Parameters

Vi

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

Channel

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

EndGate

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndQual()

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

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

Parameters

Vi

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

EndQual

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetError()

```

EXPORT int
PwrSnsr_GetError    (
                                SessionID
                                int *
                                int
                                char
                                Vi,
                                ErrorCode,
                                ErrorDescriptionBuffer
                                rSize,
                                ErrorDescription[]
                                )

```

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.

Parameters

Vi

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

ErrorCode

ErrorDescriptionBufferSize

ErrorDescription

Returns

Success (0) or error code.

◆ PwrSnsr_GetExpirationDate()

```

EXPORT int
PwrSnsr_Get
ExpirationDate (
                                int *
                                Date
                                )

```

Get the hardware license expiration date.

Parameters

Date

expiration date in the format YYYYMMDD

Returns

Success (0) or error code.

◆ PwrSnsr_GetExternalSkew()

```

EXPORT int
PwrSnsr_GetExternal
Skew    (
                                SessionID
                                const char *
                                float *
                                Vi,
                                Channel,
                                External
                                )

```

)
Gets the skew in seconds for the external trigger.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetFactoryCalDate()

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

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

Parameters

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

FactoryCalDate

Returns

Success (0) or error code.

◆ PwrSnsr_GetFetchLatency()

```
EXPORT int
PwrSnsr_GetFetchLa
tency      (
                SessionID      Vi,
```

int * Latency

)

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

Parameters

Vi

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

Latency

Fetch latency in ms.

Returns

Success (0) or error code.

◆ PwrSnsr_GetFilterState()

```
EXPORT int
PwrSnsr_GetFilterSta
te (
    SessionID Vi,
    const char * Channel,
    PwrSnsrFilterStateE
num * FilterState
)
```

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

Parameters

Vi

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

Channel

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

ManufactureDateBufferSize

ManufactureDate

Returns

Success (0) or error code.

◆ PwrSnsr_GetFilterTime()

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


)

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

Parameters

Vi

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

Channel

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

FilterTime

Returns

Success (0) or error code.

◆ PwrSnsr_GetFirmwareVersion()

```
EXPORT int
PwrSnsr_GetFirmwar
eVersion          (          SessionID          Vi,
                    const char * Channel,
                    int FirmwareVersionBuffer
                    char rSize,
                    FirmwareVersion[])
                    )
```

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

Parameters

Vi

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

Channel

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

FirmwareVersionBufferSize

Size of the FirmwareVersion buffer.

FirmwareVersion

Buffer to hold the firmware version.

Returns

Success (0) or error code.

◆ PwrSnsr_GetFpgaVersion()

```
EXPORT int
PwrSnsr_GetFpgaVer
sion              (          SessionID          Vi,
```

```

    )
    Get the sensor FPGA version.
Parameters
    Vi
        The SessionID handle that you obtain from
        the PwrSnsr_init function. The handle
        identifies a particular instrument session.
    Channel
        Channel number. For single instruments, set
        this to "CH1".
    ValBufferSize
        Size of Val in bytes
    Val
        Buffer for storing the version
Returns
        Success (0) or error code.

```

◆ PwrSnsr_GetFrequency()

```

EXPORT int
PwrSnsr_GetFrequency (
    SessionID          Vi,
    const char *       Channel,
    float *             Frequency
)
Get the RF frequency for the current sensor.
Parameters
    Vi
        The SessionID handle that you obtain from
        the PwrSnsr_init function. The handle
        identifies a particular instrument session.
    Channel
        Channel number. For single instruments, set
        this to "CH1".
    Frequency
        RF Frequency in Hz.
Returns
        Success (0) or error code.

```

◆ PwrSnsr_GetGateMode()

```

EXPORT int
PwrSnsr_GetGateMo (
    SessionID          Vi,

```

de

[PwrSnsrMeasBuffGateEnum](#) *

GateMode

)

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

Parameters

Vi

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

GateMode

Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code.

◆ PwrSnsr_GetGating()

EXPORT int

PwrSnsr_GetGating (

SessionID

const char *

Vi,

Channel,

[PwrSnsrStatGatingEnum](#) *

Gating

)

Get whether statistical capture is enabled.

Parameters

Vi

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

Channel

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

Gating

Returns

Success (0) or error code.

◆ PwrSnsr_GetHorizontalOffset()

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

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

Parameters

Vi

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

Channel

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

HorizontalOffset

Returns

Success (0) or error code.

◆ PwrSnsr_GetHorizontalScale()

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

Get the statistical mode horizontal scale in dB/Div.

Parameters

Vi

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

Channel

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

HorizontalScale

Returns

Success (0) or error code.

◆ PwrSnsr_GetImpedance()

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

Input impedance of the sensor.

Parameters

Vi

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

Channel

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

Impedance

Returns

Success (0) or error code.

◆ PwrSnsr_GetInitiateContinuous()

```
EXPORT int
PwrSnsr_GetInitiateC
ontinuous (
    SessionID
    int *
    Vi,
    InitiateContinuous
)
```

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

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

Parameters

Vi

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

InitiateContinuous

Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code.

◆ PwrSnsr_GetInternalSkew()

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

Gets the skew in seconds for the internal trigger.

Parameters**Vi**

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

Channel

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

InternalSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsAvailable()

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

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

Parameters**Vi**

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

Channel

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

IsAvailable

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsAvgSensor()

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

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

Parameters**Vi**

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

Channel

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

IsAvgSensor

True if sensor is average responding.

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsRunning()

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

Returns true if modulated/CW measurements are actively running.

Parameters**Vi**

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

Channel

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

IsRunning

True if modulated/CW measurements are actively running.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetManufactureDate()**

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

Date the sensor was manufactured in the following format YYYYmmDD.

Parameters**Vi**

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

Channel

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

ManufactureDateBufferSize

Size of ManufactureDate in bytes.

ManufactureDate

Return value.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetMarkerPixelPosition()**

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

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

Parameters

Vi

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

MarkerNumber**PixelPosition****Returns**

Success (0) or error code.

◆ PwrSnsr_GetMarkerTimePosition()

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

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

Note 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.

Parameters**Vi**

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

MarkerNumber**TimePosition****Returns**

Success (0) or error code.

◆ PwrSnsr_GetMaxFreqHighBandwidth()

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

Maximum frequency carrier the sensor can measure in high bandwidth.

Parameters

Vi

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

Channel

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

MaxFreqHighBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMaxFreqLowBandwidth()

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

Maximum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi

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

Channel

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

MaxFreqLowBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMaxMeasurements()

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

int * ValActualSize

)

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

Parameters

Vi

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

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()

```
EXPORT int
PwrSnsr_GetMaxTim
ebase          (          SessionID          Vi,
                  float *          MaxTimebase
                  )
```

Gets the maximum timebase setting available.

Parameters

Vi

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

MaxTimebase

Returns

Success (0) or error code.

◆ PwrSnsr_GetMeasBuffEnabled()

```
EXPORT int
PwrSnsr_GetMeasBu
ffEnabled      (          SessionID          Vi,
                  int *          MeasBuffEnabled
                  )
```

Get whether the measurement buffer has been enabled.

Parameters

Vi

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

MeasBuffEnabled

True if measurement buffer is enabled.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMeasurementsAvailable()

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

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

Parameters

Vi

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

Channel

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

Val

Number of measurement entries available.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMemChanArchive()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetMesial()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinFreqHighBandwidth()

```
EXPORT int
PwrSnsr_GetMinFreq
HighBandwidth (          SessionID      Vi,
                             const char *    Channel,
```

```

float *
MinFreqHighBandwidth
h
)

```

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

Parameters

Vi

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

Channel

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

MinFreqHighBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinFreqLowBandwidth()

```

EXPORT int
PwrSnsr_GetMinFreq
LowBandwidth (
                SessionID
                const char *
                float *
                Vi,
                Channel,
                MinFreqLowBandwidth
                h
)

```

Minimum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi

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

Channel

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

MinFreqLowBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinimumSupportedFirmware()

```

EXPORT int
PwrSnsr_Get
MinimumSupp
ortedFirmware (
                int *
                Version
                )

```

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinimumTrig()

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

Minimum internal trigger level in dBm.

Parameters

Vi

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

Channel

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

MinimumTrig

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinMeasurements()

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

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

Parameters

Vi

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

Channel

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

ValBufferSize

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()

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

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

Parameters**Vi**

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

Channel

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

ModelBufferSize

Size of the buffer..

Model

Buffer where the model is read into.

Returns

Success (0) or error code.

◆ PwrSnsr_GetNumberOfCals()

```
EXPORT int
PwrSnsr_Get
NumberOfCal
s (long * val)
```

Get the number of calibrations left on the license.

Parameters**val**

Number of cals left.

Returns

Success (0) or error code.

◆ PwrSnsr_GetOffsetdB()

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

Get a measurement offset in dB for the selected sensor.

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

Parameters

Vi

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

Channel

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

OffsetdB

Returns

Success (0) or error code.

◆ PwrSnsr_GetOverRan()

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

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

Parameters

Vi

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

OverRan

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakHoldDecay()

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

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

Parameters

Vi

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

Channel

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

EnvelopeAverage

Out parameter value.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakHoldTracking()

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

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

Parameters

Vi

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

Channel

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

EnvelopeTracking

Out boolean parameter value.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakPowerMax()

```
EXPORT int
PwrSnsr_GetPeakPowerMax (
                                SessionID      Vi,
                                const char *    Channel,
                                float *         PeakPowerMax
)
```

Maximum power level the sensor can measure.

Parameters**Vi**

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

Channel

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

PeakPowerMax**Returns**

Success (0) or error code.

◆ PwrSnsr_GetPeakPowerMin()

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

Minimum power level the sensor can measure.

Parameters**Vi**

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

Channel

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

PeakPowerMin

Returns

Success (0) or error code.

◆ **PwrSnsr_GetPercentPosition()**

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

Get the cursor percent on the CCDF plot.

Parameters**Vi**

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

Channel

Channel number. For single instruments, set this to "CH1". Channel Channel number. For single instruments, set this to 1.

PercentPosition**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetPeriod()**

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

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

Parameters**Vi**

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

Period

The period in seconds each timed mode acquisition is started.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPowerPosition()

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

Get the cursor power in dB on the CCDF plot.

Parameters

Vi

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

Channel

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

PowerPosition

Returns

Success (0) or error code.

◆ PwrSnsr_GetProximal()

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

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

Parameters

Vi

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

Channel

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

Proximal

Returns

Success (0) or error code.

◆ PwrSnsr_GetPulseUnits()

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

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

Parameters

Vi

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

Channel

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

PwrSnsrPulseUnitsEnum

Returns

Success (0) or error code.

◆ PwrSnsr_GetRdgsEnableFlag()

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

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

Parameters

Vi

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

Flag

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetReadingPeriod()

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

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

Parameters

Vi

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

Channel

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

ReadingPeriod

The period (rate) in seconds per new filtered reading.

Returns

Success (0) or error code.

◆ PwrSnsr_GetReturnCount()

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

Get the return count for each measurement query.

Parameters

Vi

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

ReturnCount

The return count for each measurement query.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSequenceNumbers()

```

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

```

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

Parameters

Vi

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

Channel

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

ValBufferSize

Size of the buffer.

Val

Array of sequence numbers.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSerialNumber()

```

EXPORT int
PwrSnsr_GetSerialNu
mber          (
                SessionID
                const char *
                int
                char
                Vi,
                Channel,
                SerialNumberBufferSi
                ze,
                SerialNumber[]
                )

```

Gets the serial number of the sensor.

Parameters

Vi

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

Channel

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

SerialNumberBufferSize

Size in bytes of Serial number.

SerialNumber

Out parameter. ASCII string serial number.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetSessionCount()**

```
EXPORT int
PwrSnsr_GetSession
Count          (          SessionID          Vi,
                  int *          SessionCount
                  )
```

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

Parameters**Vi**

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

SessionCount

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

Returns

Success (0) or error code.

◆ **PwrSnsr_GetSlaveSkew()**

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

Gets the skew in seconds for the slave trigger.

Parameters**Vi**

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

Channel

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

SlaveSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartDelay()

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

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

Parameters**Vi**

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

StartDelay

Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartGate()

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

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

Parameters**Vi**

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

Channel

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

StartGate**Returns**

Success (0) or error code.

◆ PwrSnsr_GetStartMode()

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

Get the mode used to start acquisition of buffer entries.

Parameters

Vi

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

StartMode

Mode used to start acquisition of buffer entries.

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartQual()

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

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

Parameters

Vi

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

StartQual

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartTimes()

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

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

Parameters

Vi

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

Channel

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

ValBufferSize

Size of the buffer.

Val

Array of start times.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSweepTime()

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

Get sweep time for the trace in seconds.

Parameters

Vi

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

Channel

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

SweepTime

Sweep time for the trace in seconds.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTempComp()

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

Get the state of the peak sensor temperature compensation system.

Parameters**Vi**

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

Channel

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

TempComp

Boolean. 1 for on; 0 for off.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTermAction()

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

Get the termination action for statistical capturing.

Parameters**Vi**

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

Channel

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

TermAction**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTermCount()

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

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

Parameters**Vi**

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

Channel

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

TermCount**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTermTime()

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

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

Parameters**Vi**

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

Channel

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

TermTime**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTimebase()

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

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

Parameters**Vi**

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

Timebase**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTimedOut()

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

Check if the last measurement buffer session timed out.

Parameters**Vi**

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

TimedOut

True if the last measurement buffer session timed out.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTimeOut()

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

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

Parameters

Vi	The SessionID handle that you obtain from 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()

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

Get time spacing for each waveform point in seconds.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetTimespan()

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

Get the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division.

Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi

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

Timespan

Returns

Success (0) or error code.

◆ PwrSnsr_GetTraceStartTime()

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

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

Parameters

Vi

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

Channel

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

TraceStartTime

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

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigDelay()

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

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

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

Parameters

Vi

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

Delay

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigHoldoff()

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

Return the trigger holdoff time in seconds.

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

trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters

Vi

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

Holdoff

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigHoldoffMode()

```
EXPORT int
PwrSnsr_GetTrigHoldoffMode (
                                SessionID      Vi,
                                PwrSnsrHoldoffModeEnum* HoldoffMode
                                )
```

Returns the holdoff mode to normal or gap holdoff.

Parameters

Vi

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

HoldoffMode

holdoff mode.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigLevel()

```
EXPORT int
PwrSnsr_GetTrigLevel (
                                SessionID      Vi,
                                float*         Level
                                )
```

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

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The

trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters

Vi

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

Level

Trigger level in dBm.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigMode()

```
EXPORT int
PwrSnsr_GetTrigMod
e                (                SessionID                Vi,
                                PwrSnsrTriggerMod
                                eEnum *                Mode
                                )
```

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

Parameters

Vi

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

Mode

Trigger mode.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigPosition()

```
EXPORT int
PwrSnsr_GetTrigPosi
tion            (                SessionID                Vi,
                                PwrSnsrTriggerPosi
                                tionEnum *                Position
                                )
```

Return the position of the trigger event on displayed sweep.

Parameters

Vi

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

Position**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigSlope()

```
EXPORT int
PwrSnsr_GetTrigSlop
e                (                SessionID                Vi,
                                PwrSnsrTriggerSlop
                                eEnum *                Slope
                                )
```

Return the trigger slope or polarity.

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

Parameters**Vi**

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

Slope**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigSource()

```
EXPORT int
PwrSnsr_GetTrigSour
ce                (                SessionID                Vi,
                                PwrSnsrTriggerSou
                                rceEnum *                Source
                                )
```

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

Parameters

Vi

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

Source**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigStatus()

```
EXPORT int
PwrSnsr_GetTrigStat
us          (          SessionID          Vi,
              PwrSnsrTriggerStat
              usEnum *          Status
            )
```

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

Parameters**Vi**

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

Status

Status of the trigger.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigVernier()

```
EXPORT int
PwrSnsr_GetTrigVern
ier          (          SessionID          Vi,
                    float *          Vernier
            )
```

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

Parameters**Vi**

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

Vernier

Trigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code.

◆ PwrSnsr_GetUnits()

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

Get units for the selected channel.

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

Parameters

Vi

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

Channel

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

Units

Returns

Success (0) or error code.

◆ PwrSnsr_GetVerticalCenter()

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

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

Parameters

Vi

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

Channel

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

VerticalCenter

Vertical center in units

Returns

Success (0) or error code.

◆ **PwrSnsr_GetVerticalScale()**

```

int EXPORT
PwrSnsr_GetVertical
Scale          (          SessionID          Vi,
                  const char *          Channel,
                  float *          VerticalScale
                  )

```

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

Parameters**Vi**

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

Channel

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

VerticalCenter

Vertical scale in units

Returns

Success (0) or error code.

◆ **PwrSnsr_GetWriteProtection()**

```

EXPORT int
PwrSnsr_GetWritePr
otection        (          SessionID          Vi,
                  int *          WriteProtection
                  )

```

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

Parameters**Vi**

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

WriteProtection

Returns true if the measurement buffer is allowed to overwrite members that have not been read by the user.

Returns

Success (0) or error code.

◆ PwrSnsr_init()

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

Initialize a communication session with a supported USB power sensor.

Parameters**ResourceName**

Name of the resource. The resource descriptor is in the following format:

USB::[VID]::[PID]::[Serial Number]::BTN

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

e.g. 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 more than one descriptor separated by a semicolon.

Channel assignment is determined by the order in the list, in other words CH1 would be the first listed resource, CH2 the second resource, etc.

e.g. Define a synthetic peak power meter using serial number 1234 for CH1 and serial number 4242 for CH2:

USB::0x1BFE::0x5500::1234::BTN;USB::0x1BFE::0x5500::4242::BTN

Returns

Success (0) or error code.

◆ PwrSnsr_InitiateAquisition()

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

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

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

Parameters

Vi

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

Returns

Success (0) or error code.

◆ PwrSnsr_IsLicenseDongleConnected()

```
EXPORT int
PwrSnsr_IsLicenseDongleConnected (int * val)
```

Get whether the hardware license dongle is connected.

Parameters

val

Boolean. 1 for connected or 0 for not connected.

Returns

Success (0) or error code.

◆ PwrSnsr_LoadMemChanFromArchive()

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

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

Parameters**Vi**

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

MemChan

Memory channel name. Must have the form MEM1...n, where n is the number of measurement channels. In single channel configurations, this parameter should always be "MEM1".

ArchiveContent

An xml document containing settings and readings obtained using the SaveToMemoryChannel method. An archive can be obtained using the GetMemChanArchive method.

Returns

Success (0) or error code.

◆ **PwrSnsr_MeasurePower()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.

Val

Average power in dBm

Returns

Success (0) or error code.

◆ PwrSnsr_MeasureVoltage()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_QueryAverageMeasurements()

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

Query the power meter for all buffered average power measurements.

Parameters

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

Channel

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

ValBufferSize

Size of the buffer in elements.

Val

Array of average power measurements.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_QueryDurations()

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

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

Parameters**Vi**

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

Channel

Channel number. For single instruments, 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()

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

```

                                int *          ValActualSize
                                )

```

Query the power meter for all buffered maximum power measurements.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_QueryMinMeasurements()

```

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

```

Query the power meter for all buffered minimum power measurements.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_QuerySequenceNumbers()

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

Query the power meter for all buffered sequence numbers.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_QueryStartTimes()

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

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

Parameters

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

Channel

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

ValBufferSize

Size of the buffer.

Val

Array of start times in seconds.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadArrayMarkerPower()

```
EXPORT int
PwrSnsr_ReadArray
MarkerPower (
```

SessionID	Vi,
const char *	Channel,
float *	AvgPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	AvgPowerCondCode,
float *	MaxPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	MaxPowerCondCode,
float *	MinPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	MinPowerCondCode,
float *	PkToAvgRatio,
<u>PwrSnsrCondCode</u>	PkToAvgRatioCondC
<u>Enum</u> *	ode,
float *	Marker1Power,
<u>PwrSnsrCondCode</u>	Marker1PowerCondC
<u>Enum</u> *	ode,
float *	Marker2Power,
<u>PwrSnsrCondCode</u>	Marker2PowerCondC
<u>Enum</u> *	ode,
float *	MarkerRatio,
<u>PwrSnsrCondCode</u>	MarkerRatioCondCod
<u>Enum</u> *	e

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

Parameters

vi

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

Channel	Channel number. For single instruments, set this to "CH1".
AvgPower	Average power between the markers.
AvgPowerCondCode	Condition code.
MaxPower	Maximum power between 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()

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

Reads byte array from the meter.

Parameters

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

ValBufferSize

Size of the buffer.

Val

Byte array from the USB.

ValActualSize

Actual size of the returned array in bytes.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadControl()

```
EXPORT int
PwrSnsr_ReadControl(
    SessionID
```

SessionID

Vi,

const char *

Channel,

int

Count,

int

ValBufferSize,

unsigned char

Val[],

int *

ValActualSize

)

Reads a control transfer on the USB.

Parameters**Vi**

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

Channel

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

Count

Maximum count to return.

ValBufferSize

Size of the buffer.

Val

Byte array from a USB control transfer.

ValActualSize

Actual size of the returned array in bytes.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadCWArray()

```
EXPORT int
PwrSnsr_ReadCWArray(
    SessionID
```

SessionID

Vi,

const char *

Channel,

float *

PeakAverage,

PwrSnsrCondCode

Enum * PeakAverageValid,
float * PeakMax,

PwrSnsrCondCode

Enum * PeakMaxValid,
float * PeakMin,

PwrSnsrCondCode

Enum * PeakMinValid,
float * PeakToAvgRatio,

PwrSnsrCondCode

Enum * PeakToAvgRatioValid

)

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

Parameters

Vi

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

Channel

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

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()

```
EXPORT int
PwrSnsr_ReadCWPower
(
```

```
SessionID
const char *
```

```
Vi,
Channel,
```

PwrSnsrCondCodeEnum *

float *

IsValid,

Val

)

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

Parameters**Vi**

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

Channel

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

IsValid

Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadDutyCycle()

EXPORT int

PwrSnsr_ReadDutyC

ycle

(

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

CondCode,

Val

)

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadEdgeDelay()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_ReadFallTime()

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

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

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle
-----------	---

Channel	identifies a particular instrument session. 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()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIEEETop()

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

PwrSnsrCondCodeEnum *

float *

CondCode,

Val

)

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalAvg()

EXPORT int

PwrSnsr_ReadInterva

lAvg

(

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

CondCode,

Val

)

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalFilteredMax()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalFilteredMin()

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

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.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.
Returns	
	Success (0) or error code.

◆ PwrSnsr_ReadIntervalMax()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalMaxAvg()

```
EXPORT int
PwrSnsr_ReadIntervalMaxAvg
(
    SessionID          Vi,
```

```

const char *      Channel,
PwrSnsrCondCode
Enum *          CondCode,
float *           Val

```

```

)
```

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalMin\(\)](#)

```

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

```

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalMinAvg\(\)](#)

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalPkToAvg\(\)](#)

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadMarkerAverage()**

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

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

Parameters**Vi**

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

Channel

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

Marker

Marker number.

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerDelta()

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

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerMax()

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

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

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

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

Marker

Marker number.

CondCodeCondition code for the measurement.
Condition code.**Val**

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerMin\(\)](#)

```

EXPORT int
PwrSnsr_ReadMarker
Min          (          SessionID          Vi,
                const char *          Channel,
                int          Marker,
                PwrSnsrCondCode
Enum *          CondCode,
                float *          Val
                )

```

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

Parameters**Vi**

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

Channel

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

Marker

Marker number.

CondCodeCondition code for the measurement.
Condition code.**Val**

measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerRatio\(\)](#)

```

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

```

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerRDelta()

```

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

```

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerRRatio()

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

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

Parameters

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

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

CondCode Condition code for the measurement. Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadOfftime()

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadOvershoot()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadPeriod()**

```
EXPORT int
PwrSnsr_ReadPeriod (
    SessionID          Vi,
```

```

                                const char *      Channel,
                                PwrSnsrCondCode
                                Enum *          CondCode,
                                float *        Val
                                )

```

Returns the interval between two successive pulses.

Parameters

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

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

CondCode Condition code for the measurement. Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadPowerArray\(\)](#)

```

EXPORT int
PwrSnsr_ReadPower
Array          (

```

```

    SessionID      Vi,
    const char *   Channel,
    float *        PulsePeak,
    PwrSnsrCondCode
    Enum *          PulsePeakValid,
    float *        PulseCycleAvg,
    PwrSnsrCondCode
    Enum *          PulseCycleAvgValid,
    float *        PulseOnAvg,
    PwrSnsrCondCode
    Enum *          PulseOnValid,
    float *        IEEETop,
    PwrSnsrCondCode
    Enum *          IEEETopValid,
    float *        IEEEBottom,
    PwrSnsrCondCode
    Enum *          IEEEBottomValid,
    float *        Overshoot,

```

PwrSnsrCondCodeEnum *

float *

OvershootValid,

Droop,

PwrSnsrCondCodeEnum *

DroopValid

)

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

Measurements performed are: peak amplitude during the pulse, average amplitude over a full cycle of the

pulse waveform, average amplitude during the pulse, IEEE top amplitude, IEEE bottom amplitude, and overshoot.

Units are the same as the channel's units. Note the pulse overshoot is returned in dB for logarithmic channel units,

and percent for all other units. Also, the pulse ON interval used for peak and average calculations is

defined by the SENSE:PULSE:STARTGT and :ENDGT time gating settings.

A full pulse (rise and fall) must be visible on the display to make average and peak pulse power measurements,

and a full cycle of the waveform must be visible to calculate average cycle amplitude.

Parameters**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
Overshoot

Condition code.

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()

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

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

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadPulseCycleAvg()

```
EXPORT int
PwrSnsr_ReadPulse
CycleAvg (
    SessionID          Vi,
    const char *       Channel,
```

PwrSnsrCondCodeEnum *

float *

CondCode,

Val

)

Returns the average power of the entire pulse.

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadPulseOnAverage()

EXPORT int

PwrSnsr_ReadPulse

OnAverage

(

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

CondCode,

Val

)

Average power of the ON portion of the pulse.

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadPulsePeak\(\)](#)

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

Returns the peak amplitude during the pulse.

Parameters

Vi

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadRiseTime\(\)](#)

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

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

Parameters

Vi

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

Channel

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

CondCodeCondition code for the measurement.
Condition code.**Val**

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadSCPI()

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

Read a SCPI string response from the instrument.

Parameters**Vi**

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

ValueBufferSize

Number of elements in Value.

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()

```
EXPORT int
PwrSnsr_ReadSCPIB
ytes (
    SessionID      Vi,
    int             ValueBufferSize,
    char            Value[],
    long *          ValueActualSize,

```

```

                                int
                                Timeout
                                )

```

Read a SCPI byte array response from the instrument.

Parameters

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

```

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

```

Read a SCPI string response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
name	Name of the parser. If parser doesn't exist, returns PWR_SNSR_ERROR_NULL_POINTER. PwrSnsr_SendSCPIToNamedParser can be used to create a named parser.

ValueBufferSize

ValueActualSize

Value

Timeout

Returns

Success (0) or error code.

Number of elements in Value.

Number of elements actually written to Value.

The string returned from the instrument SCPI interface.

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

◆ PwrSnsr_ReadTimeArray()

```
EXPORT int
PwrSnsr_ReadTimeA
rray (
```

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

PwrSnsrCondCode
Enum*

SkewValid

)

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

Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window. Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions. It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements. Note that the timing of the edge transitions is defined by the settings of the SENSE:PULSE:DISTal, :MESIal and :PROXimal settings; see the descriptions For those commands. Units are the same as the channel's units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Frequency	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid	Condition code.
Period	The interval between two successive pulses.
PeriodValid	Condition code.
Width	The interval between the first and second signal crossings of the mesial line.
WidthValid	Condition code.
Offtime	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid	Condition code.

DutyCycle

The ratio of the pulse on-time to period.

DutyCycleValid

Condition code.

Risetime

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

RisetimeValid

Condition code.

Falltime

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

FalltimeValid

Condition code.

EdgeDelay

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

EdgeDelayValid

Condition code.

Skew

The trigger offset between the assigned trigger channel and this channel.

SkewValid

Condition code.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadWaveform()

```

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

```

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

Parameters**Vi**

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

Channel

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

WaveformArrayBufferSize

Size in bytes of the Waveform buffer.

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_ReadWaveformMinMax()

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

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

Parameters

Vi

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

Channel

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

MinWaveformBufferSize

Size in bytes of the MinWaveform buffer.

MinWaveform

This array contains the min waveform. Units for the individual array elements are in the channel units setting.

MinWaveformActualSize

Size in bytes of the data written to MinWaveform.

MaxWaveformBufferSize

Size in bytes of the MaxWaveform buffer.

MaxWaveform

This array contains the max waveform. Units for the individual array elements are in the channel units setting.

MaxWaveformActualSize

Size in bytes of the data written to MaxWaveform.

WaveformArrayBufferSize

Size in bytes of the Waveform buffer.

WaveformArray

The array contains the average waveform. Units for the individual array elements are in the channel units setting.

WaveformArrayActualSize

Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadWidth()**

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

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

Parameters**Vi**

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

Channel

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

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_reset()

```
EXPORT int
PwrSnsr_reset
(           SessionID   Vi           )
```

Places the instrument in a known state.

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_ResetContinuousCapture()

```
EXPORT int
PwrSnsr_ResetContinuous
Capture (           SessionID   Vi           )
```

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_SaveToMemoryChannel()

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

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

Parameters

Vi

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

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()

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

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

Parameters

Vi

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

Channel

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

Returns

Success (0) or error code.

◆ PwrSnsr_self_test()

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

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

Parameters

Vi

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

TestResult

Returns

Success (0) or error code.

◆ `PwrSnsr_SendSCPIBytes()`

```
EXPORT int
PwrSnsr_SendSCPIB
ytes          (
```

```
SessionID
int
char
```

```
Vi,
CommandBufferSize,
Command[]
```

```
)
```

Send a SCPI command as a byte array.

Parameters

Vi

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

CommandBufferSize

Command

Returns

Success (0) or error code.

◆ `PwrSnsr_SendSCPICommand()`

```
EXPORT int
PwrSnsr_SendSCPIC
ommand        (
```

```
SessionID
const char *
```

```
Vi,
Command
```

```
)
```

Send a SCPI command to the instrument.

Parameters

Vi

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

Command**Returns**

Success (0) or error code.

◆ PwrSnsr_SendSCPIToNamedParser()

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

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

Parameters**Vi**

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

name

Name of the parser. Creates a new parser if the name is not already used.

Command**Returns**

Success (0) or error code.

◆ PwrSnsr_SetAverage()

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

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

Parameters**Vi**

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

Channel

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

Average**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetBandwidth()**

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

Set the sensor video bandwidth for the selected sensor.

Parameters**Vi**

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

Channel

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

Bandwidth**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetCalFactor()**

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

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

Parameters**Vi**

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

Channel

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

CalFactor**Returns**

Success (0) or error code.

◆ PwrSnsr_SetCapture()

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

Set whether statistical capture is enabled.

Parameters**Vi**

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

Channel

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

Capture**Returns**

Success (0) or error code.

◆ PwrSnsr_SetCCDFTraceCount()

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

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

Parameters**Vi**

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

Channel

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

TraceCount**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetContinuousCapture()**

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

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

Parameters**Vi**

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

ContinuousCapture

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

Returns

Success (0) or error code.

◆ **PwrSnsr_SetDistal()**

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

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

Parameters**Vi**

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

Channel

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

Distal**Returns**

Success (0) or error code.

◆ PwrSnsr_SetDuration()

```
EXPORT int
PwrSnsr_SetDuration (          SessionID      Vi,
                               float           Duration
                               )
```

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

Parameters**Vi**

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

Duration

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetEnabled()

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

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

Parameters**Vi**

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

Channel

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

Enabled

Boolean. 1 for enable; 0 for disable.

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndDelay()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndGate()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndQual()

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

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

Parameters

Vi

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

EndQual

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetExternalSkew()

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

Sets the skew in seconds for the external trigger.

Parameters

Vi

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

Channel

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

External

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_SetFetchLatency()

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

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

Parameters

Vi

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

Latency

Fetch latency in ms.

Returns

Success (0) or error code.

◆ PwrSnsr_SetFilterState()

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

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

Parameters

Vi

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

Channel

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

FilterState

Returns

Success (0) or error code.

◆ PwrSnsr_SetFilterTime()


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

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

Parameters

Vi

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

Channel

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

FilterTime

Returns

Success (0) or error code.

◆ PwrSnsr_SetFrequency()

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

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

Parameters

Vi

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

Channel

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

Frequency

RF Frequency in Hz.

Returns

Success (0) or error code.

◆ PwrSnsr_SetGateMode()

```
EXPORT int
PwrSnsr_SetGateMode (
    SessionID Vi,
    PwrSnsrMeasBufferGateEnum GateMode
)
```

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

Parameters

Vi

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

GateMode

Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code.

◆ PwrSnsr_SetGating()

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

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

Parameters

Vi

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

Channel

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

Gating

Returns

Success (0) or error code.

◆ PwrSnsr_SetHorizontalOffset()

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

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

Parameters

Vi

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

Channel

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

HorizontalOffset

Returns

Success (0) or error code.

◆ PwrSnsr_SetHorizontalScale()

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

Set the statistical mode horizontal scale in dB/Div.

Parameters

Vi

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

Channel

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

HorizontalScale

Returns

Success (0) or error code.

◆ PwrSnsr_SetInitiateContinuous()

```
EXPORT int
PwrSnsr_SetInitiateC
ontinuous          (          SessionID          Vi,
                    int          InitiateContinuous
                    )
```

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

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

Parameters

Vi

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

InitiateContinuous

Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code.

◆ PwrSnsr_SetInternalSkew()

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

Sets the skew in seconds for the internal trigger.

Parameters

Vi

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

Channel

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

InternalSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_SetMarkerPixelPosition()

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

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

Parameters**Vi**

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

MarkerNumber**PixelPosition****Returns**

Success (0) or error code.

◆ PwrSnsr_SetMarkerTimePosition()

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

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

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.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

identifies a particular instrument session.

MarkerNumber

TimePosition

Returns

Success (0) or error code.

◆ PwrSnsr_SetMeasBuffEnabled()

```
EXPORT int
PwrSnsr_SetMeasBuf
fEnabled          (          SessionID          Vi,
                    int          MeasBuffEnabled
                    )
```

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

Parameters

Vi

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

MeasBuffEnabled

True to enable measurement buffer, false to disable.

Returns

Success (0) or error code.

◆ PwrSnsr_SetMesial()

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

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

Parameters

Vi

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

Channel

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

Mesial**Returns**

Success (0) or error code.

◆ PwrSnsr_SetOffsetdB()

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

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.

Parameters**Vi**

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

Channel

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

OffsetdB**Returns**

Success (0) or error code.

◆ PwrSnsr_SetPeakHoldDecay()

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

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

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeAverage	Peak hold decay value.
Returns	Success (0) or error code.

◆ PwrSnsr_SetPeakHoldTracking()

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

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

Parameters

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetPercentPosition()

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


Set the cursor percent on the CCDF plot.

Parameters

Vi

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

Channel

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

PercentPosition

Returns

Success (0) or error code.

◆ PwrSnsr_SetPeriod()

```
EXPORT int
PwrSnsr_SetPeriod (          SessionID      Vi,
                             float           Period
                             )
```

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

Parameters

Vi

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

Period

The period in seconds each timed mode acquisition is started.

Returns

Success (0) or error code.

◆ PwrSnsr_SetPowerPosition()

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

Set the cursor power in dB on the CCDF plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

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

PowerPosition**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetProximal()**

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

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

Parameters**Vi**

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

Channel

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

Proximal**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetPulseUnits()**

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

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

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

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

PwrSnsrPulseUnitsEnum**Returns**

Success (0) or error code.

◆ PwrSnsr_SetRdgsEnableFlag()

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

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

Parameters**Vi**

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

Flag

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

Returns

Success (0) or error code.

◆ PwrSnsr_SetReturnCount()

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

Set the return count for each measurement query.

Parameters**Vi**

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

ReturnCount

The return count for each measurement query.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetSessionCount()**

```
EXPORT int
PwrSnsr_SetSession
Count          (          SessionID          Vi,
                                     int          SessionCount
                                     )
```

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

Parameters**Vi**

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

SessionCount

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

Returns

Success (0) or error code.

◆ **PwrSnsr_SetSessionTimeout()**

```
EXPORT int
PwrSnsr_SetSession
Timeout        (          SessionID          Vi,
                                     float          Seconds
                                     )
```

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

Parameters**Vi**

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

Seconds

Set the time out value. Values less than or equal to 0 will be treated as infinite. Valid range : 0.001 to 1000

Returns

Success (0) or error code.

◆ **PwrSnsr_SetSlaveSkew()**

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

Sets the skew in seconds for the slave trigger.

Parameters**Vi**

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

Channel

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

SlaveSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ **PwrSnsr_SetStartDelay()**

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

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

Parameters**Vi**

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

StartDelay

Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetStartGate()**

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

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

Parameters**Vi**

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

Channel

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

StartGate**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetStartMode()**

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

Set the mode used to start acquisition of buffer entries.

Parameters**Vi**

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

StartMode

Mode used to start acquisition of buffer entries.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetStartQual()**

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

Set the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartQual

The minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTempComp()**

```
EXPORT int
PwrSnsr_SetTempComp (
    SessionID const char * Vi,
    int Channel,
    TempComp
)
```

Set the state of the peak sensor temperature compensation system.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TempComp

Boolean. 1 for on; 0 for off.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermAction()

```
EXPORT int
PwrSnsr_SetTermAct
ion          (          SessionID          Vi,
               const char *          Channel,
               PwrSnsrTermAction          TermAction
               Enum
               )
```

Set the termination action for statistical capturing.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermAction

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermCount()

```
EXPORT int
PwrSnsr_SetTermCo
unt          (          SessionID          Vi,
               const char *          Channel,
               double          TermCount
               )
```

Set the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermCount

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermTime()

```
EXPORT int
PwrSnsr_SetTermTime
(
    SessionID
    const char *
    int
    Vi,
    Channel,
    TermTime
)
```

Set the termination time in seconds (1 - 3600) for statistical capturing. After the time has elapsed, the action determined by TermAction is taken.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermTime**Returns**

Success (0) or error code.

◆ PwrSnsr_SetTimebase()

```
EXPORT int
PwrSnsr_SetTimebase
(
    SessionID
    float
    Vi,
    Timebase
)
```

Set the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 sec (or max timebase) in a 1-2-5 sequence,.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timebase**Returns**

Success (0) or error code.

◆ [PwrSnsr_SetTimeOut\(\)](#)

```
EXPORT int
PwrSnsr_SetTimeOut (          SessionID      Vi,
                               long            Milliseconds
                           )
```

Sets the time out in milliseconds for I/O.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Milliseconds

Time out in milliseconds. Use -1 for infinite time out.

Returns

Success (0) or error code.

◆ [PwrSnsr_SetTimespan\(\)](#)

```
EXPORT int
PwrSnsr_SetTimespan
n          (          SessionID      Vi,
                               float    Timespan
                           )
```

Set the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division.
Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timespan

Returns

Success (0) or error code.

◆ [PwrSnsr_SetTrigDelay\(\)](#)

```
EXPORT int
PwrSnsr_SetTrigDelay (
                                SessionID    Vi,
                                float         Delay
)
```

Sets the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position.

Positive values cause the actual trigger to occur after the trigger condition is met. This places the trigger event to the left of the trigger point on the display, and is useful for viewing events during a pulse, some fixed delay time after the rising edge trigger. Negative trigger delay places the trigger event to the right of the trigger point on the display, and is useful for looking at events before the trigger edge.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Delay

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigHoldoff()

```
EXPORT int
PwrSnsr_SetTrigHoldoff (
                                SessionID    Vi,
                                float         Holdoff
)
```

Sets the trigger holdoff time in seconds.

Trigger holdoff is used to disable the trigger for a specified amount of time after each trigger event. The holdoff time starts immediately after each valid trigger edge, and will not permit any new triggers until the time has expired. When the holdoff time is up, the trigger re-arms, and the next valid trigger event (edge) will cause a new sweep. This feature is used to help synchronize the power meter with burst waveforms such as a TDMA or GSM frame. The trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Holdoff**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetTrigHoldoffMode()**

```
EXPORT int
PwrSnsr_SetTrigHoldoffMode (
                                SessionID      Vi,
                                PwrSnsrHoldoffModeEnum HoldoffMode
                                )
```

Sets the holdoff mode to normal or gap holdoff.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

HoldoffMode

Holdoff mode.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTrigLevel()**

```
EXPORT int
PwrSnsr_SetTrigLevel (
                                SessionID      Vi,
                                float           Level
                                )
```

Set the trigger level for synchronizing data acquisition with a pulsed input signal.

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Level

Trigger level in dBm.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTrigMode()**

```
EXPORT int
PwrSnsr_SetTrigMod
e                (                SessionID                Vi,
                                PwrSnsrTriggerMod
                                eEnum                        Mode
                                )
```

Set the trigger mode for synchronizing data acquisition with pulsed signals.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Mode

Trigger mode.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTrigOutMode()**

```
EXPORT int
PwrSnsr_SetTrigOut
Mode                (                SessionID                Vi,
                                const char *                Channel,
                                int                            Mode
                                )
```

Sets the trigger out/mult io mode. Setting trigger mode overrides this command.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Mode

Trigger out/multi IO mode

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigPosition()

```
EXPORT int
PwrSnsr_SetTrigPosition (
                                SessionID      Vi,
                                PwrSnsrTriggerPositionEnum Position
                                )
```

Set the position of the trigger event on displayed sweep.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Position

Trigger position.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigSlope()

```
EXPORT int
PwrSnsr_SetTrigSlope (
                                SessionID      Vi,
                                PwrSnsrTriggerSlopeEnum Slope
                                )
```

Sets the trigger slope or polarity.

When set to positive, trigger events will be generated when a signal rising edge crosses the trigger level threshold. When negative, trigger events are generated on the falling edge of the pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

identifies a particular instrument session.

Slope

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigSource()

```
EXPORT int
PwrSnsr_SetTrigSource (
    SessionID Vi,
    PwrSnsrTriggerSourceEnum Source
)
```

Get the signal the power meter monitors for a trigger. It can be channel external input, or independent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Source

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigVernier()

```
EXPORT int
PwrSnsr_SetTrigVernier (
    SessionID Vi,
    float Vernier
)
```

Set the fine position of the trigger event on the power sweep.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Vernier

Trigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code.

◆ PwrSnsr_SetUnits()

```
EXPORT int
PwrSnsr_SetUnits    (
                                SessionID      Vi,
                                const char *    Channel,
                                PwrSnsrUnitsEnum Units
                                )
```

Set units for the selected channel.

Voltage is calculated with reference to the sensor input impedance. Note that for ratiometric results, logarithmic units will always return dBr (dB relative) while linear units return percent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Units

Returns

Success (0) or error code.

◆ PwrSnsr_SetVerticalCenter()

```
int EXPORT
PwrSnsr_SetVertical
Center    (
                                SessionID      Vi,
                                const char *    Channel,
                                float           VerticalCenter
                                )
```

Sets vertical center based on current units: <arg> = (range varies by units)

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical center in units

Returns

Success (0) or error code.

◆ **PwrSnsr_SetVerticalScale()**

```
int EXPORT
PwrSnsr_SetVertical
Scale          (          SessionID          Vi,
                  const char *          Channel,
                  float          VerticalScale
                  )
```

Sets vertical scale based on current units: <arg> = (range varies by units)

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical scale in units

Returns

Success (0) or error code.

◆ **PwrSnsr_SetWriteProtection()**

```
EXPORT int
PwrSnsr_SetWritePr
otection        (          SessionID          Vi,
                  int          WriteProtection
                  )
```

Set whether to allow the measurement buffer to overwrite entries that have not been read by the user.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

WriteProtection

Set false to allow the measurement buffer to overwrite entries that have not been read by the user.

Returns

Success (0) or error code.

◆ PwrSnsr_StartAcquisition()

```
EXPORT int
PwrSnsr_Start
Acquisition (          SessionID      Vi          )
```

Starts measurement buffer acquisition. This method allows the user to send a command to the power meter to begin buffering measurements without waiting for all measurements to be completed. Alternately, you can call the AcquireReadings method to start buffering measurements and wait for them to be read from the meter.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_StatModeReset()

```
EXPORT int
PwrSnsr_StatModeR
eset (          SessionID      Vi,
          const char *      Channel
        )
```

Resets statistical capturing mode by clearing the buffers and restarting the acquisition timer.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_Status()

```
EXPORT int
PwrSnsr_Status      (
                                SessionID      Vi,
                                PwrSnsrAcquisition
                                StatusEnum * Val
                                )
```

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Val

Status out parameter.

Returns

Success (0) or error code.

◆ [PwrSnsr_StopAcquisition\(\)](#)

```
EXPORT int
PwrSnsr_Stop
Acquisition      (
                                SessionID      Vi
                                )
```

Sends a command to stop the measurement buffer from acquiring readings.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ [PwrSnsr_Write\(\)](#)

```
EXPORT int
PwrSnsr_Write      (
                                SessionID      Vi,
                                const char *   Channel,
                                int            DataBufferSize,
                                unsigned char  Data[]
                                )
```

Write a byte array to the meter.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

DataBufferSize

Channel number. For single instruments, set this to "CH1".

Data

Size of the buffer in bytes.

Returns

Data to send.

Success (0) or error code.

◆ PwrSnsr_Zero()

```
EXPORT int
PwrSnsr_Zero      (      SessionID      Vi,
                      const char *      Channel
                      )
```

Performs a zero offset null adjustment.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_ZeroQuery()

```
EXPORT int
PwrSnsr_ZeroQuery (      SessionID      Vi,
                      const char *      Channel,
                      int *              Val
                      )
```

Performs a zero offset null adjustment and returns true if successful.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Val

Boolean value for operation success or failure.

Returns

Success (0) or error code.

Generated by  1.8.15**2. .1.1 PulseInfo**

Power Sensor Library

[Data Fields](#)

PulseInfo Struct Reference

Data structure containing pulse information. [More...](#)

```
#include <PwrSnsrLib.h>
```

Data Fields

float	Width
float	Peak
float	Min
float	PulseAvg
float	Position
float	RiseProximal
float	RiseDistal
float	RiseTime

	float	FallProximal
	float	FallDistal
	float	FallTime

Detailed Description

Data structure containing pulse information.

Field Documentation

◆ [FallDistal](#)

float FallDistal

Position in time for the distal crossing on the falling edge of the pulse.

◆ [FallProximal](#)

float FallProximal

Position in time for the proximal crossing on the falling edge of the pulse.

◆ [FallTime](#)

float FallTime

Fall time of the pulse.

◆ [Min](#)

float Min

Minimum instantaneous power measurement.

◆ [Peak](#)

float Peak

Peak (max instantaneous) power measurement.

◆ Position

float Position

Time position corresponding to the mesial crossing of the rising edge for the pulse.

◆ PulseAvg

float PulseAvg

Average power measurement for the pulse.

◆ RiseDistal

float RiseDistal

Position in time for the distal crossing on the rising edge of the pulse.

◆ RiseProximal

float RiseProximal

Position in time for the proximal crossing on the rising edge of the pulse.

◆ RiseTime

float RiseTime

Rise time of the pulse.

◆ Width

float Width

Pulse width is defined as the interval between the first and second signal crossings of the mesial line.

The documentation for this struct was generated from the following file:

- **PwrSnsrLib.h**
-

2. .1.2 CURRENT_TIMEOUT

Power Sensor Library

[Data Structures](#) | [Macros](#) | [Typedefs](#) | [Enumerations](#) | [Functions](#)

PwrSnsrLib.h File Reference

Go to the source code of this file.

Data Structures	
struct	PulseInfo
	Data structure containing pulse information. More...
Macros	
#define	SUCCESS (0L)
#define	CURRENT_TIMEOUT (-2)
#define	EXPORT
#define	ERROR_BASE (0xBFFA0000L)
Typedefs	
typedef int	SessionID
typedef enum	PwrSnsrAcquisitionStatusEnum PwrSnsrAcquisitionStatusEnum
typedef enum	PwrSnsrTriggerModeEnum PwrSnsrTriggerModeEnum

typedef enum <u>PwrSnsrTriggerSlopeEnum</u>	<u>PwrSnsrTriggerSlopeEnum</u>
typedef enum <u>PwrSnsrTriggerPositionEnum</u>	<u>PwrSnsrTriggerPositionEnum</u>
typedef enum <u>PwrSnsrTriggerSourceEnum</u>	<u>PwrSnsrTriggerSourceEnum</u>
typedef enum <u>PwrSnsrUnitsEnum</u>	<u>PwrSnsrUnitsEnum</u>
typedef enum <u>PwrSnsrMarkerNumberEnum</u>	<u>PwrSnsrMarkerNumberEnum</u>
typedef enum <u>PwrSnsrBandwidthEnum</u>	<u>PwrSnsrBandwidthEnum</u>
typedef enum <u>PwrSnsrFilterStateEnum</u>	<u>PwrSnsrFilterStateEnum</u>
typedef enum <u>PwrSnsrPulseUnitsEnum</u>	<u>PwrSnsrPulseUnitsEnum</u>
typedef enum <u>PwrSnsrCondCodeEnum</u>	<u>PwrSnsrCondCodeEnum</u>
typedef enum <u>PwrSnsrTriggerStatusEnum</u>	<u>PwrSnsrTriggerStatusEnum</u>
typedef enum <u>PwrSnsrTermActionEnum</u>	<u>PwrSnsrTermActionEnum</u>
typedef enum <u>PwrSnsrHoldoffModeEnum</u>	<u>PwrSnsrHoldoffModeEnum</u>
typedef enum <u>PwrSnsrStatGatingEnum</u>	<u>PwrSnsrStatGatingEnum</u>
typedef enum <u>PwrSnsrTrigOutModeEnum</u>	<u>PwrSnsrTrigOutModeEnum</u>
typedef enum <u>PwrSnsrMeasBuffGateEnum</u>	<u>PwrSnsrMeasBuffGateEnum</u>
typedef enum <u>PwrSnsrMeasBuffStartModeEnum</u>	<u>PwrSnsrMeasBuffStartModeEnum</u>

typedef enum PwrSnsrMeasBuffStopReasonEnum	PwrSnsrMeasBuffStopReasonEnum
typedef enum PwrSnsrRdgsEnableFlag	PwrSnsrRdgsEnableFlag
typedef enum PwrSnsrErrorCodesEnum	PwrSnsrErrorCodesEnum
typedef struct PulseInfo	PulseInfo
	Data structure containing pulse information. More...
Enumerations	
enum	PwrSnsrAcquisitionStatusEnum { PwrSnsrAcqComplete = 1, PwrSnsrAcqInProgress = 0, PwrSnsrAcqStatusUnknown = -1 }
enum	PwrSnsrTriggerModeEnum { PwrSnsrTriggerModeNormal = 1, PwrSnsrTriggerModeAuto = 2, PwrSnsrTriggerModeAutoLevel = 3, PwrSnsrTriggerModeFreerun = 4 }
enum	PwrSnsrTriggerSlopeEnum { PwrSnsrTriggerSlopePositive = 1, PwrSnsrTriggerSlopeNegative = 0 }
enum	PwrSnsrTriggerPositionEnum { PwrSnsrTriggerPositionLeft = 0, PwrSnsrTriggerPositionMiddle = 1, PwrSnsrTriggerPositionRight = 2 }
enum	PwrSnsrTriggerSourceEnum { PwrSnsrTriggerSourceChannel1 = 0, PwrSnsrTriggerSourceExternal = 2, PwrSnsrTriggerSourceChannel2 = 1, PwrSnsrTriggerSourceChannel3 = 3, PwrSnsrTriggerSourceChannel4 = 4, PwrSnsrTriggerSourceChannel5 = 5,

	<u>PwrSnsrTriggerSourceChannel6</u> = 6, <u>PwrSnsrTriggerSourceChannel7</u> = 7, <u>PwrSnsrTriggerSourceChannel8</u> = 8, <u>PwrSnsrTriggerSourceChannel9</u> = 9, <u>PwrSnsrTriggerSourceChannel10</u> = 10, <u>PwrSnsrTriggerSourceChannel11</u> = 11, <u>PwrSnsrTriggerSourceChannel12</u> = 12, <u>PwrSnsrTriggerSourceChannel13</u> = 13, <u>PwrSnsrTriggerSourceChannel14</u> = 14, <u>PwrSnsrTriggerSourceChannel15</u> = 15, <u>PwrSnsrTriggerSourceChannel16</u> = 16, <u>PwrSnsrTriggerSourceIndependent</u> = 17 }
enum	<u>PwrSnsrUnitsEnum</u> { <u>PwrSnsrUnitsdBm</u> = 0, <u>PwrSnsrUnitswatts</u> = 1, <u>PwrSnsrUnitsvolts</u> = 2, <u>PwrSnsrUnitsDBV</u> = 3, <u>PwrSnsrUnitsDBMV</u> = 4, <u>PwrSnsrUnitsDBUV</u> = 5 }
enum	<u>PwrSnsrMarkerNumberEnum</u> { <u>PwrSnsrMarkerNumberMarker1</u> = 1, <u>PwrSnsrMarkerNumberMarker2</u> = 2 }
enum	<u>PwrSnsrBandwidthEnum</u> { <u>PwrSnsrBandwidthHigh</u> = 0, <u>PwrSnsrBandwidthLow</u> = 1 }
enum	<u>PwrSnsrFilterStateEnum</u> { <u>PwrSnsrFilterStateOff</u> = 0, <u>PwrSnsrFilterStateOn</u> = 1, <u>PwrSnsrFilterStateAuto</u> = 2 }
enum	<u>PwrSnsrPulseUnitsEnum</u> { <u>PwrSnsrPulseUnitsWatts</u> = 0, <u>PwrSnsrPulseUnitsVolts</u> = 1 }
enum	<u>PwrSnsrCondCodeEnum</u> { <u>PwrSnsrCondCodeMeasurementStopped</u> = -1, <u>PwrSnsrCondCodeError</u> = 0,

	<u>PwrSnsrCondCodeUnderrange</u> = 2, <u>PwrSnsrCondCodeOverrange</u> = 3, <u>PwrSnsrCondCodeNormal</u> = 1 }
enum	<u>PwrSnsrTriggerStatusEnum</u> { <u>PwrSnsrTriggerStatusStopped</u> = 0, <u>PwrSnsrTriggerStatusPretrig</u> = 1, <u>PwrSnsrTriggerStatusWaiting</u> = 2, <u>PwrSnsrTriggerStatusAcquiringNew</u> = 3, <u>PwrSnsrTriggerStatusAutoTrig</u> = 4, <u>PwrSnsrTriggerStatusFreerun</u> = 5, <u>PwrSnsrTriggerStatusTriggered</u> = 6, <u>PwrSnsrTriggerStatusRunning</u> = 7 }
enum	<u>PwrSnsrTermActionEnum</u> { <u>PwrSnsrTermActionStop</u> = 0, <u>PwrSnsrTermActionRestart</u> = 1, <u>PwrSnsrTermActionDecimate</u> = 2 }
enum	<u>PwrSnsrHoldoffModeEnum</u> { <u>PwrSnsrHoldoffModeNormal</u> = 1, <u>PwrSnsrHoldoffModeGap</u> = 2 }
enum	<u>PwrSnsrStatGatingEnum</u> { <u>PwrSnsrStatGatingFreeRun</u> = 0, <u>PwrSnsrStatGatingMarkers</u> = 1 }
enum	<u>PwrSnsrTrigOutModeEnum</u> { <u>PwrSnsrTrigOutModeMioOff</u> = 0, <u>PwrSnsrTrigOutModeMioPullUp</u> = 1, <u>PwrSnsrTrigOutModeMioTtl0</u> = 2, <u>PwrSnsrTrigOutModeMioTbRef</u> = 3, <u>PwrSnsrTrigOutModeMioSweepHigh</u> = 4, <u>PwrSnsrTrigOutModeMioSweepLow</u> = 5, <u>PwrSnsrTrigOutModeMioTrigHigh</u> = 6, <u>PwrSnsrTrigOutModeMioTrigLow</u> = 7, <u>PwrSnsrTrigOutModeMioMaster</u> = 8, <u>PwrSnsrTrigOutModeMioSlave</u> = 9 }
enum	<u>PwrSnsrMeasBuffGateEnum</u> {

	<pre> PwrSnsrMeasBuffGateBurst = 0, PwrSnsrMeasBuffGateMarker = 1, PwrSnsrMeasBuffGateExtGate = 2, PwrSnsrMeasBuffGatePeriodic = 3, PwrSnsrMeasBuffGateExtTrig = 4 } </pre>
enum	<pre> PwrSnsrMeasBuffStartModeEnum { PwrSnsrMeasBuffStartModeImmediate = 1, PwrSnsrMeasBuffStartModeExternalEna ble = 2, PwrSnsrMeasBuffStartModeExternalStart = 3 } </pre>
enum	<pre> PwrSnsrMeasBuffStopReasonEnum { PwrSnsrMeasBuffStopReasonCountRea ched = 1, PwrSnsrMeasBuffStopReasonTimedOut = 2, PwrSnsrMeasBuffStopReasonBufferOve rran = 3, PwrSnsrMeasBuffStopReasonNone = 0 } </pre>
enum	<pre> PwrSnsrRdgsEnableFlag { PwrSnsrSequenceEnable = 1, PwrSnsrStartTimeEnable = 2, PwrSnsrDurationEnable = 4, PwrSnsrMinEnable = 8, PwrSnsrAvgEnable = 16, PwrSnsrMaxEnable = 32 } </pre>
enum	<pre> PwrSnsrErrorCodesEnum { PWR_SNSR_IO_GENERAL = - 2147204588, PWR_SNSR_IO_TIMEOUT = -2147204587, PWR_SNSR_MODEL_NOT_SUPPORTED = -2147204586, PWR_SNSR_INV_PARAMETER = - 1073807240, PWR_SNSR_ERROR_INVALID_SESSION HANDLE = -1074130544, PWR_SNSR_ERROR_STATUS_NOT_AVAI LABLE = -1074134947, </pre>

```

PWR_SNSR_ERROR_RESET_FAILED = -
1074134945,
PWR_SNSR_ERROR_RESOURCE_UNKN
OWN = -1074134944,
PWR_SNSR_ERROR_ALREADY_INITIA
LIZED = -1074134943,
PWR_SNSR_ERROR_OUT_OF_MEMORY
= -1074134954,
PWR_SNSR_ERROR_OPERATION_PEND
ING = -1074134953,
PWR_SNSR_ERROR_NULL_POINTER = -
1074134952,
PWR_SNSR_ERROR_UNEXPECTED_R
ESPONSE = -1074134951,
PWR_SNSR_ERROR_NOT_INITIALIZED =
-1074135011,
PWR_SNSR_LIBUSB_ERROR_IO = -1,
PWR_SNSR_LIBUSB_ERROR_INVALID_P
ARAM = -2,
PWR_SNSR_LIBUSB_ERROR_ACCESS
= -3,
PWR_SNSR_LIBUSB_ERROR_NO_DEVIC
E = -4,
PWR_SNSR_LIBUSB_ERROR_NOT_FOU
ND = -5,
PWR_SNSR_LIBUSB_ERROR_BUSY = -6,
PWR_SNSR_LIBUSB_ERROR_TIMEOU
T = -7,
PWR_SNSR_LIBUSB_ERROR_OVERFLO
W = -8,
PWR_SNSR_LIBUSB_ERROR_PIPE = -9,
PWR_SNSR_LIBUSB_ERROR_INTERRUPTED = -10,
PWR_SNSR_LIBUSB_ERROR_NO_MEM
= -11,
PWR_SNSR_LIBUSB_ERROR_NOT_SUP
PORTED = -12,
PWR_SNSR_LIBUSB_ERROR_OTHER = -
99
}

```

Functions

EXPORT int [PwrSnsr_SendSCPICommand](#) (SessionID Vi, const char *Command)

	Send a SCPI command to the instrument. More...
EXPORT int	PwrSnsr_ReadSCPI (SessionID Vi, int ValueBufferSize, long *ValueActualSize, char Value[], int Timeout)
	Read a SCPI string response from the instrument. More...
EXPORT int	PwrSnsr_SendSCPIToNamedParser (SessionID Vi, const char *name, const char *Command)
	Send a SCPI command to the instrument using a named SCPI parser. More...
EXPORT int	PwrSnsr_ReadSCPIFromNamedParser (SessionID Vi, const char *name, int ValueBufferSize, long *ValueActualSize, char Value[], int Timeout)
	Read a SCPI string response from the instrument. More...
EXPORT int	PwrSnsr_FindResources (const char *Delimiter, int ValBufferSize, char Val[])
	Returns a delimited string of available resources. These strings can be used in the initialize function to open a session to an instrument. More...
EXPORT int	PwrSnsr_GetMinimumSupportedFirmware (int *Version)
	Gets the minimum supported firmware as an integer. Format is YYYYMMDD. More...
EXPORT int	PwrSnsr_SendSCPIBytes (SessionID Vi, int CommandBufferSize, char Command[])
	Send a SCPI command as a byte array. More...
EXPORT int	PwrSnsr_ReadSCPIBytes (SessionID Vi, int ValueBufferSize, char Value[], long *ValueActualSize, int Timeout)

	Read a SCPI byte array response from the instrument. More...
EXPORT int	PwrSnsr_SetTimeOut (SessionID Vi, long Milliseconds)
	Sets the time out in milliseconds for I/O. More...
EXPORT int	PwrSnsr_GetTimeOut (SessionID Vi, long *Val)
	Returns the time out value for I/O in milliseconds. More...
EXPORT int	PwrSnsr_init (char *ResourceName, SessionID *Vi)
	Initialize a communication session with a supported USB power sensor. More...
EXPORT int	PwrSnsr_close (SessionID Vi)
	Closes the I/O session to the instrument. Driver methods and properties that access the instrument are not accessible after Close is called. More...
EXPORT int	PwrSnsr_GetError (SessionID Vi, int *ErrorCode, int ErrorDescriptionBufferSize, char ErrorDescription[])
	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. More...
EXPORT int	PwrSnsr_ClearError (SessionID Vi)
	This function clears the error code and error description for the given session. More...
EXPORT int	PwrSnsr_reset (SessionID Vi)
EXPORT int	PwrSnsr_self_test (SessionID Vi, int *TestResult)

	Performs an instrument self test, waits for the instrument to complete the test, and queries the instrument for the results. If the instrument passes the test, TestResult is 0. More...
EXPORT int	PwrSnsr_MeasurePower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return average power using a default instrument configuration in Modulated Mode and dBm units. Instrument remains stopped in Modulated Mode after a measurement. More...
EXPORT int	PwrSnsr_FetchCWPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the most recently acquired CW power. More...
EXPORT int	PwrSnsr_MeasureVoltage (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return average voltage using a default instrument configuration in Modulated Mode and volts units. Instrument remains stopped in Modulated Mode after a measurement. More...
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)
	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 <code>FetchMinMaxWaveform</code> to obtain the min/max waveforms for other channels. More...
EXPORT int	PwrSnsr_ReadWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the waveform for this channel. Call <code>FetchWaveform</code> to obtain the waveforms for other channels. More...
EXPORT int	PwrSnsr_FetchWaveformMinMax (SessionID Vi, const char *Channel, int MinWaveformBufferSize, float MinWaveform[], int *MinWaveformActualSize, int MaxWaveformBufferSize, float MaxWaveform[], int *MaxWaveformActualSize, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Returns the previously acquired minimum and maximum waveforms for this specified channel. The acquisition must be made prior to calling this method. Call this method separately for each channel. More...
EXPORT int	PwrSnsr_FetchWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize)
	Returns a previously acquired waveform for this channel. The acquisition must be made prior to calling this method. Call this method separately for each channel. More...
EXPORT int	PwrSnsr_FetchPowerArray (SessionID Vi, const char *Channel, float *PulsePeak,

	<p><u>PwrSnsrCondCodeEnum</u> *PulsePeakValid, float *PulseCycleAvg, <u>PwrSnsrCondCodeEnum</u> *PulseCycleAvgValid, float *PulseOnAvg, <u>PwrSnsrCondCodeEnum</u> *PulseOnValid, float *IEEETop, <u>PwrSnsrCondCodeEnum</u> *IEEETopValid, float *IEEEBottom, <u>PwrSnsrCondCodeEnum</u> *IEEEBottomValid, float *Overshoot, <u>PwrSnsrCondCodeEnum</u> *OvershootValid, float *Droop, <u>PwrSnsrCondCodeEnum</u> *DroopValid)</p>
	Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform. <u>More...</u>
EXPORT int	<p><u>PwrSnsr_FetchTimeArray</u> (SessionID Vi, const char *Channel, float *Frequency, <u>PwrSnsrCondCodeEnum</u> *FrequencyValid, float *Period, <u>PwrSnsrCondCodeEnum</u> *PeriodValid, float *Width, <u>PwrSnsrCondCodeEnum</u> *WidthValid, float *Offtime, <u>PwrSnsrCondCodeEnum</u> *OfftimeValid, float *DutyCycle, <u>PwrSnsrCondCodeEnum</u> *DutyCycleValid, float *Risetime, <u>PwrSnsrCondCodeEnum</u> *RisetimeValid, float *Falltime, <u>PwrSnsrCondCodeEnum</u> *FalltimeValid, float *EdgeDelay, <u>PwrSnsrCondCodeEnum</u> *EdgeDelayValid, float *Skew, <u>PwrSnsrCondCodeEnum</u> *SkewValid)</p>
	Returns an array of the current automatic timing measurements performed on a periodic pulse waveform. <u>More...</u>
EXPORT int	<p><u>PwrSnsr_FetchCWArray</u> (SessionID Vi, const char *Channel, float *PeakAverage, <u>PwrSnsrCondCodeEnum</u> *PeakAverageValid, float *PeakMax, <u>PwrSnsrCondCodeEnum</u> *PeakMaxValid, float *PeakMin, <u>PwrSnsrCondCodeEnum</u> *PeakMinValid, float *PeakToAvgRatio, <u>PwrSnsrCondCodeEnum</u> *PeakToAvgRatioValid)</p>
	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. More...
EXPORT int	PwrSnsr_FetchRiseTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line. More...
EXPORT int	PwrSnsr_FetchWidth (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line. More...
EXPORT int	PwrSnsr_FetchPulsePeak (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the peak amplitude during the pulse. More...
EXPORT int	PwrSnsr_FetchPulseOnAverage (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Average power of the ON portion of the pulse. More...
EXPORT int	PwrSnsr_FetchPulseCycleAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the average power of the entire pulse. More...

EXPORT int	PwrSnsr_FetchPRF (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency). More...
EXPORT int	PwrSnsr_FetchPeriod (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between two successive pulses. (Reciprocal of the Pulse RepetitionFrequency) More...
EXPORT int	PwrSnsr_FetchOvershoot (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units. More...
EXPORT int	PwrSnsr_FetchOfftime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulsewidth). More...
EXPORT int	PwrSnsr_FetchIEEETop (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse. More...
EXPORT int	PwrSnsr_FetchIEEEBottom (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)

	Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns. More...
EXPORT int	PwrSnsr_FetchFallTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the interval between the last signal crossing of the distal line to the last signalcrossing of the proximal line. More...
EXPORT int	PwrSnsr_FetchEdgeDelay (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform. More...
EXPORT int	PwrSnsr_FetchDutyCycle (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Returns the ratio of the pulse on-time to off-time. More...
EXPORT int	PwrSnsr_GetTrigDelay (SessionID Vi, float *Delay)
	Return the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position. More...
EXPORT int	PwrSnsr_SetTrigDelay (SessionID Vi, float Delay)
	Sets the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position. More...
EXPORT int	PwrSnsr_GetTrigHoldoff (SessionID Vi, float *Holdoff)

	Return the trigger holdoff time in seconds. More...
EXPORT int	PwrSnsr_SetTrigHoldoff (SessionID Vi, float Holdoff)
	Sets the trigger holdoff time in seconds. More...
EXPORT int	PwrSnsr_GetTrigHoldoffMode (SessionID Vi, PwrSnsrHoldoffModeEnum *HoldoffMode)
	Returns the holdoff mode to normal or gap holdoff. More...
EXPORT int	PwrSnsr_SetTrigHoldoffMode (SessionID Vi, PwrSnsrHoldoffModeEnum HoldoffMode)
	Sets the holdoff mode to normal or gap holdoff. More...
EXPORT int	PwrSnsr_GetTrigLevel (SessionID Vi, float *Level)
	Return the trigger level for synchronizing data acquisition with a pulsed input signal. More...
EXPORT int	PwrSnsr_SetTrigLevel (SessionID Vi, float Level)
	Set the trigger level for synchronizing data acquisition with a pulsed input signal. More...
EXPORT int	PwrSnsr_GetTrigMode (SessionID Vi, PwrSnsrTriggerModeEnum *Mode)
	Return the trigger mode for synchronizing data acquisition with pulsed signals. More...
EXPORT int	PwrSnsr_SetTrigMode (SessionID Vi, PwrSnsrTriggerModeEnum Mode)
	Set the trigger mode for synchronizing data acquisition with pulsed signals. More...

EXPORT int	PwrSnsr_GetTrigPosition (SessionID Vi, PwrSnsrTriggerPositionEnum *Position)
	Return the position of the trigger event on displayed sweep. More...
EXPORT int	PwrSnsr_SetTrigPosition (SessionID Vi, PwrSnsrTriggerPositionEnum Position)
	Set the position of the trigger event on displayed sweep. More...
EXPORT int	PwrSnsr_GetTrigSource (SessionID Vi, PwrSnsrTriggerSourceEnum *Source)
	Set the signal the power meter monitors for a trigger. It can be channel external input, or independent. More...
EXPORT int	PwrSnsr_SetTrigSource (SessionID Vi, PwrSnsrTriggerSourceEnum Source)
	Get the signal the power meter monitors for a trigger. It can be channel external input, or independent. More...
EXPORT int	PwrSnsr_GetTrigStatus (SessionID Vi, PwrSnsrTriggerStatusEnum *Status)
	The status of the triggering system. Update rate is controlled by FetchLatency setting. More...
EXPORT int	PwrSnsr_SetFetchLatency (SessionID Vi, int Latency)
	Set the period the library waits to update fetch measurements in ms. More...
EXPORT int	PwrSnsr_GetFetchLatency (SessionID Vi, int *Latency)
	Get the period the library waits to update fetch measurements in ms. More...
EXPORT int	PwrSnsr_GetTrigVernier (SessionID Vi, float *Vernier)

	Return the fine position of the trigger event on the power sweep. More...
EXPORT int	PwrSnsr_SetTrigVernier (SessionID Vi, float Vernier)
	Set the fine position of the trigger event on the power sweep. More...
EXPORT int	PwrSnsr_GetTrigSlope (SessionID Vi, PwrSnsrTriggerSlopeEnum *Slope)
	Return the trigger slope or polarity. More...
EXPORT int	PwrSnsr_SetTrigSlope (SessionID Vi, PwrSnsrTriggerSlopeEnum Slope)
	Sets the trigger slope or polarity. More...
EXPORT int	PwrSnsr_Clear (SessionID Vi)
	Clear all data buffers. Clears averaging filters to empty. More...
EXPORT int	PwrSnsr_InitiateAquisition (SessionID Vi)
	Starts a single measurement cycle when INITiate:CONTinuous is set to OFF. More...
EXPORT int	PwrSnsr_Status (SessionID Vi, PwrSnsrAcquisitionStatusEnum *Val)
	Returns whether an acquisition is in progress, complete, or if the status is unknown. More...
EXPORT int	PwrSnsr_SetInitiateContinuous (SessionID Vi, int InitiateContinuous)
	Set the data acquisition mode for single or free-run measurements. More...
EXPORT int	PwrSnsr_GetInitiateContinuous (SessionID Vi, int *InitiateContinuous)
	Get the data acquisition mode for single or free-run measurements. More...

EXPORT int	PwrSnsr_EnableCapturePriority (SessionID Vi, const char *Channel, int Enabled)
	Sets the 55 series power meter to a buffered capture mode and disables real time processing. More...
EXPORT int	PwrSnsr_GetEnabled (SessionID Vi, const char *Channel, int *Enabled)
	Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed. More...
EXPORT int	PwrSnsr_SetEnabled (SessionID Vi, const char *Channel, int Enabled)
	Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed. More...
EXPORT int	PwrSnsr_GetSerialNumber (SessionID Vi, const char *Channel, int SerialNumberBufferSize, char SerialNumber[])
	Gets the serial number of the sensor. More...
EXPORT int	PwrSnsr_GetChannelCount (SessionID Vi, int *Count)
	Get number of channels. More...
EXPORT int	PwrSnsr_GetUnits (SessionID Vi, const char *Channel, PwrSnsrUnitsEnum *Units)
	Get units for the selected channel. More...
EXPORT int	PwrSnsr_SetUnits (SessionID Vi, const char *Channel, PwrSnsrUnitsEnum Units)
	Set units for the selected channel. More...

EXPORT int	PwrSnsr_GetCurrentTemp (SessionID Vi, const char *Channel, double *CurrentTemp)
	Get current sensor internal temperature in degrees C. More...
EXPORT int	PwrSnsr_GetAverage (SessionID Vi, const char *Channel, int *Average)
	Get the number of traces averaged together to form the measurement result on the selected channel. More...
EXPORT int	PwrSnsr_SetAverage (SessionID Vi, const char *Channel, int Average)
	Set the number of traces averaged together to form the measurement result on the selected channel. More...
EXPORT int	PwrSnsr_GetBandwidth (SessionID Vi, const char *Channel, PwrSnsrBandwidthEnum *Bandwidth)
	Get the sensor video bandwidth for the selected sensor. More...
EXPORT int	PwrSnsr_SetBandwidth (SessionID Vi, const char *Channel, PwrSnsrBandwidthEnum Bandwidth)
	Set the sensor video bandwidth for the selected sensor. More...
EXPORT int	PwrSnsr_GetFilterState (SessionID Vi, const char *Channel, PwrSnsrFilterStateEnum *FilterState)
	Get the current setting of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_SetFilterState (SessionID Vi, const char *Channel, PwrSnsrFilterStateEnum FilterState)
	Set the current setting of the integration filter on the selected channel. More...

EXPORT int	PwrSnsr_GetFilterTime (SessionID Vi, const char *Channel, float *FilterTime)
	Get the current length of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_SetFilterTime (SessionID Vi, const char *Channel, float FilterTime)
	Set the current length of the integration filter on the selected channel. More...
EXPORT int	PwrSnsr_GetDistal (SessionID Vi, const char *Channel, float *Distal)
	Get the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition. More...
EXPORT int	PwrSnsr_SetDistal (SessionID Vi, const char *Channel, float Distal)
	Set the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition. More...
EXPORT int	PwrSnsr_GetEndGate (SessionID Vi, const char *Channel, float *EndGate)
	Get the point on a pulse, which is used to define the end of the pulse's active interval. More...
EXPORT int	PwrSnsr_SetEndGate (SessionID Vi, const char *Channel, float EndGate)
	Set the point on a pulse, which is used to define the end of the pulse's active interval. More...
EXPORT int	PwrSnsr_GetMesial (SessionID Vi, const char *Channel, float *Mesial)
	Get the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_SetMesial (SessionID Vi, const char *Channel, float Mesial)

	Set the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_GetProximal (SessionID Vi, const char *Channel, float *Proximal)
	Get the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_SetProximal (SessionID Vi, const char *Channel, float Proximal)
	Set the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition. More...
EXPORT int	PwrSnsr_GetPulseUnits (SessionID Vi, const char *Channel, PwrSnsrPulseUnitsEnum *Units)
	Get the units for entering the pulse distal, mesial and proximal levels. More...
EXPORT int	PwrSnsr_SetPulseUnits (SessionID Vi, const char *Channel, PwrSnsrPulseUnitsEnum PwrSnsrPulseUnitsEnum)
	Set the units for entering the pulse distal, mesial and proximal levels. More...
EXPORT int	PwrSnsr_GetStartGate (SessionID Vi, const char *Channel, float *StartGate)
	Get the point on a pulse, which is used to define the beginning of the pulse's active interval. More...
EXPORT int	PwrSnsr_SetStartGate (SessionID Vi, const char *Channel, float StartGate)
	Set the point on a pulse, which is used to define the beginning of the pulse's active interval. More...

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)
	Query information associated with calibration factors. More...
EXPORT int	PwrSnsr_GetCalFactor (SessionID Vi, const char *Channel, float *CalFactor)
	Get the frequency calibration factor currently in use on the selected channel. More...
EXPORT int	PwrSnsr_SetCalFactor (SessionID Vi, const char *Channel, float CalFactor)
	Set the frequency calibration factor currently in use on the selected channel. More...
EXPORT int	PwrSnsr_GetFrequency (SessionID Vi, const char *Channel, float *Frequency)
	Get the RF frequency for the current sensor. More...
EXPORT int	PwrSnsr_SetFrequency (SessionID Vi, const char *Channel, float Frequency)
	Set the RF frequency for the current sensor, and apply the appropriate frequency calibration factor from the sensor internal table. More...
EXPORT int	PwrSnsr_GetOffsetdB (SessionID Vi, const char *Channel, float *OffsetdB)
	Get a measurement offset in dB for the selected sensor. More...
EXPORT int	PwrSnsr_SetOffsetdB (SessionID Vi, const char *Channel, float OffsetdB)
	Set a measurement offset in dB for the selected sensor. More...

EXPORT int	PwrSnsr_GetTempComp (SessionID Vi, const char *Channel, int *TempComp)
	Get the state of the peak sensor temperature compensation system. More...
EXPORT int	PwrSnsr_SetTempComp (SessionID Vi, const char *Channel, int TempComp)
	Set the state of the peak sensor temperature compensation system. More...
EXPORT int	PwrSnsr_GetTimebase (SessionID Vi, float *Timebase)
	Get the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 (or max timebase) sec in a 1-2-5 sequence,. More...
EXPORT int	PwrSnsr_SetTimebase (SessionID Vi, float Timebase)
	Set the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 sec (or max timebase) in a 1-2-5 sequence,. More...
EXPORT int	PwrSnsr_SetTimespan (SessionID Vi, float Timespan)
	Set the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence. More...
EXPORT int	PwrSnsr_GetTimespan (SessionID Vi, float *Timespan)
	Get the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division. Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence. More...
EXPORT int	PwrSnsr_GetMaxTimebase (SessionID Vi, float *MaxTimebase)

	Gets the maximum timebase setting available. More...
EXPORT int	PwrSnsr_FetchArrayMarkerPower (SessionID Vi, const char *Channel, float *AvgPower, PwrSnsrCondCodeEnum *AvgPowerCondCode, float *MaxPower, PwrSnsrCondCodeEnum *MaxPowerCondCode, float *MinPower, PwrSnsrCondCodeEnum *MinPowerCondCode, float *PkToAvgRatio, PwrSnsrCondCodeEnum *PkToAvgRatioCondCode, float *Marker1Power, PwrSnsrCondCodeEnum *Marker1PowerCondCode, float *Marker2Power, PwrSnsrCondCodeEnum *Marker2PowerCondCode, float *MarkerRatio, PwrSnsrCondCodeEnum *MarkerRatioCondCode)
	Returns an array of the current marker measurements for the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerAverage (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Forthe specified marker, return the average power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerMax (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Forthe specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerMin (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *IsValid, float *Val)
	Forthe specified marker, return the minimum power or voltage at the marker. The units are

	the same as the specified channel. More...
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)
	Returns an array of the current marker measurements for the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerAverage (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Forthe specified marker, return the average power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerMax (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Forthe specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadMarkerMin (SessionID Vi, const char *Channel, int Marker, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Forthe specified marker, return the minimum power or voltage at the marker. The units are the same as the specified channel. More...

EXPORT int	PwrSnsr_FetchIntervalAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)	
		Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalFilteredMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)	
		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. More...
EXPORT int	PwrSnsr_FetchIntervalFilteredMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)	
		Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)	
		Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)	
		Return the minimum instantaneous power or voltage in the time interval between marker1

	and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalPkToAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units. More...
EXPORT int	PwrSnsr_ReadIntervalAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalFilteredMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	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. More...
EXPORT int	PwrSnsr_ReadIntervalFilteredMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalMax (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)

	Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalMin (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalPkToAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units. More...
EXPORT int	PwrSnsr_FetchIntervalMaxAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchIntervalMinAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalMaxAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)

	Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_ReadIntervalMinAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel. More...
EXPORT int	PwrSnsr_FetchMarkerDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the difference between MK1 and MK2. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_FetchMarkerRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units. More...
EXPORT int	PwrSnsr_FetchMarkerRDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *condCode, float *Val)
	Return the difference between MK2 and MK1. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_FetchMarkerRRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units. More...

EXPORT int	PwrSnsr_ReadMarkerDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the difference between MK1 and MK2. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_ReadMarkerRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units. More...
EXPORT int	PwrSnsr_ReadMarkerRDelta (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the difference between MK2 and MK1. The units will be the same as marker units. More...
EXPORT int	PwrSnsr_ReadMarkerRRatio (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units. More...
EXPORT int	PwrSnsr_ReadCWPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *IsValid, float *Val)
EXPORT int	PwrSnsr_ReadCWArray (SessionID Vi, const char *Channel, float *PeakAverage, PwrSnsrCondCodeEnum *PeakAverageValid, float *PeakMax, PwrSnsrCondCodeEnum *PeakMaxValid, float *PeakMin, PwrSnsrCondCodeEnum *PeakMinValid, float *PeakToAvgRatio,

	<u>PwrSnsrCondCodeEnum</u> *PeakToAvgRatioValid)
	Returns the current average, maximum, minimum powers or voltages and the peak-to-average ratio of the specified channel. Units are the same as the channel'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. More...
EXPORT int	<u>PwrSnsr_ReadPowerArray</u> (SessionID Vi, const char *Channel, float *PulsePeak, <u>PwrSnsrCondCodeEnum</u> *PulsePeakValid, float *PulseCycleAvg, <u>PwrSnsrCondCodeEnum</u> *PulseCycleAvgValid, float *PulseOnAvg, <u>PwrSnsrCondCodeEnum</u> *PulseOnValid, float *IEEETop, <u>PwrSnsrCondCodeEnum</u> *IEEETopValid, float *IEEEBottom, <u>PwrSnsrCondCodeEnum</u> *IEEEBottomValid, float *Overshoot, <u>PwrSnsrCondCodeEnum</u> *OvershootValid, float *Droop, <u>PwrSnsrCondCodeEnum</u> *DroopValid)
	Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform. More...
EXPORT int	<u>PwrSnsr_ReadTimeArray</u> (SessionID Vi, const char *Channel, float *Frequency, <u>PwrSnsrCondCodeEnum</u> *FrequencyValid, float *Period, <u>PwrSnsrCondCodeEnum</u> *PeriodValid, float *Width, <u>PwrSnsrCondCodeEnum</u> *WidthValid, float *Offtime, <u>PwrSnsrCondCodeEnum</u> *OfftimeValid, float *DutyCycle, <u>PwrSnsrCondCodeEnum</u> *DutyCycleValid, float *Risetime, <u>PwrSnsrCondCodeEnum</u> *RisetimeValid, float *Falltime, <u>PwrSnsrCondCodeEnum</u> *FalltimeValid, float *EdgeDelay, <u>PwrSnsrCondCodeEnum</u> *EdgeDelayValid, float *Skew, <u>PwrSnsrCondCodeEnum</u> *SkewValid)

	Returns an array of the current automatic timing measurements performed on a periodic pulse waveform. More...
EXPORT int	PwrSnsr_ReadDutyCycle (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the ratio of the pulse on-time to off-time. More...
EXPORT int	PwrSnsr_ReadEdgeDelay (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform. More...
EXPORT int	PwrSnsr_ReadFallTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the interval between the last signal crossing of the distal line to the last signal crossing of the proximal line. More...
EXPORT int	PwrSnsr_ReadIEEEBottom (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns. More...
EXPORT int	PwrSnsr_ReadIEEETop (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse. More...

EXPORT int	PwrSnsr_ReadOfftime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulse width). More...
EXPORT int	PwrSnsr_ReadOvershoot (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units. More...
EXPORT int	PwrSnsr_ReadPeriod (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the interval between two successive pulses. More...
EXPORT int	PwrSnsr_ReadPRF (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency). More...
EXPORT int	PwrSnsr_ReadPulseCycleAvg (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the average power of the entire pulse. More...
EXPORT int	PwrSnsr_ReadPulseOnAverage (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Average power of the ON portion of the pulse. More...

EXPORT int	PwrSnsr_ReadPulsePeak (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the peak amplitude during the pulse. More...
EXPORT int	PwrSnsr_ReadRiseTime (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line. More...
EXPORT int	PwrSnsr_ReadWidth (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line. More...
EXPORT int	PwrSnsr_GetHorizontalOffset (SessionID Vi, const char *Channel, double *HorizontalOffset)
	Get the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative). More...
EXPORT int	PwrSnsr_SetHorizontalOffset (SessionID Vi, const char *Channel, double HorizontalOffset)
	Set the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative). More...
EXPORT int	PwrSnsr_GetHorizontalScale (SessionID Vi, const char *Channel, double *HorizontalScale)

	Get the statistical mode horizontal scale in dB/Div. More...
EXPORT int	PwrSnsr_SetHorizontalScale (SessionID Vi, const char *Channel, double HorizontalScale)
	Set the statistical mode horizontal scale in dB/Div. More...
int EXPORT	PwrSnsr_GetVerticalCenter (SessionID Vi, const char *Channel, float *VerticalCenter)
	Gets vertical center based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_SetVerticalCenter (SessionID Vi, const char *Channel, float VerticalCenter)
	Sets vertical center based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_GetVerticalScale (SessionID Vi, const char *Channel, float *VerticalScale)
	Gets vertical scale based on current units: <arg> = (range varies by units) More...
int EXPORT	PwrSnsr_SetVerticalScale (SessionID Vi, const char *Channel, float VerticalScale)
	Sets vertical scale based on current units: <arg> = (range varies by units) More...
EXPORT int	PwrSnsr_GetChannelByIndex (SessionID Vi, int BuffSize, char Channel[], int Index)
	Gets the channel name by zero index. Note: SCPI commands use a one-based index. More...
EXPORT int	PwrSnsr_FetchCCDFTrace (SessionID Vi, const char *Channel, int TraceBufferSize, float Trace[], int *TraceActualSize)
	Returns the points in the CCDF trace. More...

EXPORT int	PwrSnsr_StatModeReset (SessionID Vi, const char *Channel)
	Resets statistical capturing mode by clearing the buffers and restarting the aquisition timer. More...
EXPORT int	PwrSnsr_FetchStatMeasurementArray (SessionID Vi, const char *Channel, double *Pavg, PwrSnsrCondCodeEnum *PavgCond, double *Ppeak, PwrSnsrCondCodeEnum *PpeakCond, double *Pmin, PwrSnsrCondCodeEnum *PminCond, double *PkToAvgRatio, PwrSnsrCondCodeEnum *PkToAvgRatioCond, double *CursorPwr, PwrSnsrCondCodeEnum *CursorPwrCond, double *CursorPct, PwrSnsrCondCodeEnum *CursorPctCond, double *SampleCount, PwrSnsrCondCodeEnum *SampleCountCond, double *SecondsRun, PwrSnsrCondCodeEnum *SecondsRunCond)
	Returns an array of the current automatic statistical measurements performed on a sample population. More...
EXPORT int	PwrSnsr_FetchCCDFPower (SessionID Vi, const char *Channel, double Percent, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Return relative power (in dB) for a given percent on the CCDF plot. More...
EXPORT int	PwrSnsr_FetchCCDFPercent (SessionID Vi, const char *Channel, double Power, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Return relative power (in dB) for a given percent on the CCDF plot. More...
EXPORT int	PwrSnsr_GetCapture (SessionID Vi, const char *Channel, int *Capture)

	Get whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_SetCapture (SessionID Vi, const char *Channel, int Capture)
	Set whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_GetGating (SessionID Vi, const char *Channel, PwrSnsrStatGatingEnum *Gating)
	Get whether statistical capture is enabled. More...
EXPORT int	PwrSnsr_SetGating (SessionID Vi, const char *Channel, PwrSnsrStatGatingEnum Gating)
	Set whether the statical capture is gated by markers or free-running. More...
EXPORT int	PwrSnsr_GetTermAction (SessionID Vi, const char *Channel, PwrSnsrTermActionEnum *TermAction)
	Get the termination action for statistical capturing. More...
EXPORT int	PwrSnsr_SetTermAction (SessionID Vi, const char *Channel, PwrSnsrTermActionEnum TermAction)
	Set the termination action for statistical capturing. More...
EXPORT int	PwrSnsr_GetTermCount (SessionID Vi, const char *Channel, double *TermCount)
	Get the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_SetTermCount (SessionID Vi, const char *Channel, double TermCount)

	Set the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_GetTermTime (SessionID Vi, const char *Channel, int *TermTime)
	Get the termination time in seconds for statistical capturing. After the time has elapsed, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_SetTermTime (SessionID Vi, const char *Channel, int TermTime)
	Set the termination time in seconds (1 - 3600) for statistical capturing. After the time has elapsed, the action determined by TermAction is taken. More...
EXPORT int	PwrSnsr_GetCCDFTraceCount (SessionID Vi, const char *Channel, int *TraceCount)
	Get the number of points in the CCDF trace plot. More...
EXPORT int	PwrSnsr_SetCCDFTraceCount (SessionID Vi, const char *Channel, int TraceCount)
	Set the number of points (1 - 16384) in the CCDF trace plot. More...
EXPORT int	PwrSnsr_FetchCursorPercent (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Returns the percent CCDF at the cursor. More...
EXPORT int	PwrSnsr_FetchCursorPower (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, double *Val)
	Returns the power CCDF in dB at the cursor. More...

EXPORT int	PwrSnsr_GetPercentPosition (SessionID Vi, const char *Channel, double *PercentPosition)
	Get the cursor percent on the CCDF plot. More...
EXPORT int	PwrSnsr_SetPercentPosition (SessionID Vi, const char *Channel, double PercentPosition)
	Set the cursor percent on the CCDF plot. More...
EXPORT int	PwrSnsr_SetPowerPosition (SessionID Vi, const char *Channel, double PowerPosition)
	Set the cursor power in dB on the CCDF plot. More...
EXPORT int	PwrSnsr_GetPowerPosition (SessionID Vi, const char *Channel, double *PowerPosition)
	Get the cursor power in dB on the CCDF plot. More...
EXPORT int	PwrSnsr_GetAcqStatusArray (SessionID Vi, const char *Channel, int *SweepLength, double *SampleRate, double *SweepRate, double *SweepTime, double *StartTime, int *StatusWord)
	Returns data about the status of the acquisition system. More...
EXPORT int	PwrSnsr_GetDiagStatusArray (SessionID Vi, const char *Channel, float *DetectorTemp, float *CpuTemp, float *MioVoltage, float *VccInt10, float *VccAux18, float *Vcc50, float *Vcc25, float *Vcc33)
	Returns diagnostic data. More...
EXPORT int	PwrSnsr_GetMarkerTimePosition (SessionID Vi, int MarkerNumber, float *TimePosition)

	Get the time (x-axis-position) of the selected marker relative to the trigger. More...
EXPORT int	PwrSnsr_SetMarkerTimePosition (SessionID Vi, int MarkerNumber, float TimePosition)
	Set the time (x-axis-position) of the selected marker relative to the trigger. More...
EXPORT int	PwrSnsr_GetMarkerPixelPosition (SessionID Vi, int MarkerNumber, int *PixelPosition)
	Get the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive. More...
EXPORT int	PwrSnsr_SetMarkerPixelPosition (SessionID Vi, int MarkerNumber, int PixelPosition)
	Set the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive. More...
EXPORT int	PwrSnsr_GetManufactureDate (SessionID Vi, const char *Channel, int ManufactureDateBufferSize, char ManufactureDate[])
	Date the sensor was manufactured in the following format YYYYmmDD. More...
EXPORT int	PwrSnsr_GetImpedance (SessionID Vi, const char *Channel, float *Impedance)
	Input impedance of the sensor. More...
EXPORT int	PwrSnsr_GetPeakPowerMax (SessionID Vi, const char *Channel, float *PeakPowerMax)
	Maximum power level the sensor can measure. More...

EXPORT int	PwrSnsr_GetPeakPowerMin (SessionID Vi, const char *Channel, float *PeakPowerMin)
	Minimum power level the sensor can measure. More...
EXPORT int	PwrSnsr_GetAttenuation (SessionID Vi, const char *Channel, float *Attenuation)
	Attenuation in dB for the sensor. More...
EXPORT int	PwrSnsr_GetFactoryCalDate (SessionID Vi, const char *Channel, int FactoryCalDateBufferSize, char FactoryCalDate[])
	The date (YYYYmmDD) the last time the sensor was calibrated at the factory. More...
EXPORT int	PwrSnsr_GetMinimumTrig (SessionID Vi, const char *Channel, float *MinimumTrig)
	Minimum internal trigger level in dBm. More...
EXPORT int	PwrSnsr_GetMinFreqHighBandwidth (SessionID Vi, const char *Channel, float *MinFreqHighBandwidth)
	Minimum frequency of RF the sensor can measure in high bandwidth. More...
EXPORT int	PwrSnsr_GetMaxFreqHighBandwidth (SessionID Vi, const char *Channel, float *MaxFreqHighBandwidth)
	Maximum frequency carrier the sensor can measure in high bandwidth. More...
EXPORT int	PwrSnsr_GetMinFreqLowBandwidth (SessionID Vi, const char *Channel, float *MinFreqLowBandwidth)
	Minimum frequency carrier the sensor can measure in low bandwidth. More...
EXPORT int	PwrSnsr_GetMaxFreqLowBandwidth (SessionID Vi, const char *Channel, float *MaxFreqLowBandwidth)

	Maximum frequency carrier the sensor can measure in low bandwidth. More...
EXPORT int	PwrSnsr_GetFpgaVersion (SessionID Vi, const char *Channel, int ValBufferSize, char Val[])
	Get the sensor FPGA version. More...
EXPORT int	PwrSnsr_GetExternalSkew (SessionID Vi, const char *Channel, float *External)
	Gets the skew in seconds for the external trigger. More...
EXPORT int	PwrSnsr_SetExternalSkew (SessionID Vi, const char *Channel, float External)
	Sets the skew in seconds for the external trigger. More...
EXPORT int	PwrSnsr_GetSlaveSkew (SessionID Vi, const char *Channel, float *SlaveSkew)
	Gets the skew in seconds for the slave trigger. More...
EXPORT int	PwrSnsr_SetSlaveSkew (SessionID Vi, const char *Channel, float SlaveSkew)
	Sets the skew in seconds for the slave trigger. More...
EXPORT int	PwrSnsr_GetInternalSkew (SessionID Vi, const char *Channel, float *InternalSkew)
	Gets the skew in seconds for the internal trigger. More...
EXPORT int	PwrSnsr_SetInternalSkew (SessionID Vi, const char *Channel, float InternalSkew)
	Sets the skew in seconds for the internal trigger. More...
EXPORT int	PwrSnsr_Zero (SessionID Vi, const char *Channel)

	Performs a zero offset null adjustment. More...
EXPORT int	PwrSnsr_ZeroQuery (SessionID Vi, const char *Channel, int *Val)
	Performs a zero offset null adjustment and returns true if successful. More...
EXPORT int	PwrSnsr_Abort (SessionID Vi)
	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. More...
EXPORT int	PwrSnsr_FetchExtendedWaveform (SessionID Vi, const char *Channel, int WaveformArrayBufferSize, float WaveformArray[], int *WaveformArrayActualSize, int Count)
	When capture priority is enabled, returns up to 100000 points of trace data based on the current timebase starting at the current trigger delay point. More...
EXPORT int	PwrSnsr_GetTimePerPoint (SessionID Vi, const char *Channel, float *TimePerPoint)
	Get time spacing for each waveform point in seconds. More...
EXPORT int	PwrSnsr_GetSweepTime (SessionID Vi, const char *Channel, float *SweepTime)
	Get sweep time for the trace in seconds. More...
EXPORT int	PwrSnsr_GetChanTraceCount (SessionID Vi, const char *Channel, int *TraceCount)
	Get the number of points in the CCDF trace plot. More...
EXPORT int	PwrSnsr_GetTraceStartTime (SessionID Vi, const char *Channel, float *TraceStartTime)

	Get time offset (start time) of the trace in seconds. May be negative, indicating pre-trigger information. More...
EXPORT int	PwrSnsr_FetchDistal (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the distal level in the current channel units. More...
EXPORT int	PwrSnsr_FetchMesial (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the mesial level in the current channel units. More...
EXPORT int	PwrSnsr_FetchProximal (SessionID Vi, const char *Channel, PwrSnsrCondCodeEnum *CondCode, float *Val)
	Returns the actual detected power of the proximal level in the current channel units. More...
EXPORT int	PwrSnsr_FetchAllMultiPulse (SessionID Vi, const char *Channel, int PulseInfosSize, PulseInfo PulseInfos[], int *PulseInfosActualSize)
	Return all previously acquired multiple pulse measurements. The elements in the PulseInfos array correspond to pulses on the current trace from left to right (ascending time order). More...
EXPORT int	PwrSnsr_SetTrigOutMode (SessionID Vi, const char *Channel, int Mode)
	Sets the trigger out/mult io mode. Setting trigger mode overrides this command. More...
EXPORT int	PwrSnsr_SaveToMemoryChannel (SessionID Vi, const char *memChan, const

	char *ChannelName)
	Saves the given channel to a memory channel. If the memory channel does not exist, a new one is created. More...
EXPORT int	PwrSnsr_GetMemChanArchive (SessionID Vi, const char *memChan, int ValBufferSize, char Val[])
	Returns an XML document containing settings and readings obtained using the SaveToMemoryChannel method. More...
EXPORT int	PwrSnsr_LoadMemChanFromArchive (SessionID Vi, const char *memChan, const char *ArchiveContent)
	Loads the named memory channel using the given archive. If the memory channel does not exist, one is created. More...
EXPORT int	PwrSnsr_SaveUserCal (SessionID Vi, const char *Channel)
	Instructs power meter to save the value of fixed cal, zero, and skew values. More...
EXPORT int	PwrSnsr_ClearUserCal (SessionID Vi, const char *Channel)
	Resets the value of fixed cal, zero, and skew to factory defaults. More...
EXPORT int	PwrSnsr_GetIsAvgSensor (SessionID Vi, const char *Channel, int *IsAvgSensor)
	Returns true if sensor is average responding (not peak detecting). More...
EXPORT int	PwrSnsr_GetIsAvailable (SessionID Vi, const char *Channel, int *IsAvailable)
	Returns true if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled. More...
EXPORT int	PwrSnsr_GetIsRunning (SessionID Vi, const char *Channel, int *IsRunning)

	Returns true if modulated/CW measuremnts are actively running. More...
EXPORT int	PwrSnsr_GetReadingPeriod (SessionID Vi, const char *Channel, float *ReadingPeriod)
	Returns the period (rate) in seconds per new filtered reading. More...
EXPORT int	PwrSnsr_GetBufferedAverageMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the average power measurements that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_AcquireMeasurements (SessionID Vi, double Timeout, int Count, PwrSnsrMeasBuffStopReasonEnum *StopReason, int *Val)
	Initiates new acquisition from the measurement buffer system (if acquisition is in the stopped state). Blocks until the number of measurements for each enabled channel is equal to count, or a time out has occurred. More...
EXPORT int	PwrSnsr_GetMaxMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the maximum power measurements that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetMinMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the minimum power measurements that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetDuration (SessionID Vi, float *Duration)

	Get the time duration samples are captured during each timed mode acquisition. More...
EXPORT int	PwrSnsr_SetDuration (SessionID Vi, float Duration)
	Set the duration samples are captured during each timed mode acquisition. More...
EXPORT int	PwrSnsr_GetSequenceNumbers (SessionID Vi, const char *Channel, int ValBufferSize, long long Val[], int *ValActualSize)
	Get the sequence number entries that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetStartTimes (SessionID Vi, const char *Channel, int ValBufferSize, double Val[], int *ValActualSize)
	Get the start time entries in seconds that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_GetDurations (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Get the duration entries in seconds that were captured during the last call to AcquireMeasurements. More...
EXPORT int	PwrSnsr_StartAcquisition (SessionID Vi)
	Starts measurement buffer acquisition. This method allows the user to send a command to the power meter to begin buffering measurements without waiting for all measurements to be completed. Alternately, you can call the AcquireReadings method to start buffering measurements and wait for them to be read from the meter. More...
EXPORT int	PwrSnsr_StopAcquisition (SessionID Vi)
	Sends a command to stop the measurement buffer from acquiring readings. More...

EXPORT int	PwrSnsr_ClearBuffer (SessionID Vi)
	Sends a command to the power meter to clear all buffered readings. This method does not clear cached measurements accessible through GetAverageMeasurements, etc. More...
EXPORT int	PwrSnsr_ClearMeasurements (SessionID Vi)
	Clears cached average, min, max, duration, start time, and sequence number measurements. More...
EXPORT int	PwrSnsr_GetMeasurementsAvailable (SessionID Vi, const char *Channel, int *Val)
	Get the number of measurement entries available that were captured during AcquireMeasurements(). More...
EXPORT int	PwrSnsr_SetPeriod (SessionID Vi, float Period)
	Set the period each timed mode acquisition (measurement buffer) is started. More...
EXPORT int	PwrSnsr_GetPeriod (SessionID Vi, float *Period)
	Get the period each timed mode acquisition (measurement buffer) is started. More...
EXPORT int	PwrSnsr_GetRdgsEnableFlag (SessionID Vi, int *Flag)
	Get the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements. More...
EXPORT int	PwrSnsr_SetRdgsEnableFlag (SessionID Vi, int Flag)
	Set the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements. More...

EXPORT int	PwrSnsr_GetGateMode (SessionID Vi, PwrSnsrMeasBuffGateEnum *GateMode)
	Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. The gate signal may be internally or externally generated in several different ways. More...
EXPORT int	PwrSnsr_SetGateMode (SessionID Vi, PwrSnsrMeasBuffGateEnum GateMode)
	Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. More...
EXPORT int	PwrSnsr_GetStartMode (SessionID Vi, PwrSnsrMeasBuffStartModeEnum *StartMode)
	Get the mode used to start acquisition of buffer entries. More...
EXPORT int	PwrSnsr_SetStartMode (SessionID Vi, PwrSnsrMeasBuffStartModeEnum StartMode)
	Set the mode used to start acquisition of buffer entries. More...
EXPORT int	PwrSnsr_AdvanceReadIndex (SessionID Vi)
	Send a command to the meter to notify it the user is done reading and to advance the read index. More...
EXPORT int	PwrSnsr_QueryAverageMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered average power measurements. More...
EXPORT int	PwrSnsr_QueryStartTimes (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered start times in seconds. More...

EXPORT int	PwrSnsr_QuerySequenceNumbers (SessionID Vi, const char *Channel, int ValBufferSize, long long Val[], int *ValActualSize)
	Query the power meter for all buffered sequence numbers. More...
EXPORT int	PwrSnsr_QueryDurations (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered measurement durations in seconds. More...
EXPORT int	PwrSnsr_QueryMaxMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered maximum power measurements. More...
EXPORT int	PwrSnsr_QueryMinMeasurements (SessionID Vi, const char *Channel, int ValBufferSize, float Val[], int *ValActualSize)
	Query the power meter for all buffered minimum power measurements. More...
EXPORT int	PwrSnsr_GetWriteProtection (SessionID Vi, int *WriteProtection)
	Get whether the measurement buffer is set to overwrite members that have not been read by the user. More...
EXPORT int	PwrSnsr_GetTimedOut (SessionID Vi, int *TimedOut)
	Check if the last measurement buffer session timed out. More...
EXPORT int	PwrSnsr_GetSessionCount (SessionID Vi, int *SessionCount)
	Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...

EXPORT int	PwrSnsr_SetSessionCount (SessionID Vi, int SessionCount)
	Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...
EXPORT int	PwrSnsr_SetSessionTimeout (SessionID Vi, float Seconds)
	Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements. More...
EXPORT int	PwrSnsr_GetReturnCount (SessionID Vi, int *ReturnCount)
	Get the return count for each measurement query. More...
EXPORT int	PwrSnsr_SetReturnCount (SessionID Vi, int ReturnCount)
	Set the return count for each measurement query. More...
EXPORT int	PwrSnsr_SetWriteProtection (SessionID Vi, int WriteProtection)
	Set whether to allow the measurement buffer to overwrite entries that have not been read by the user. More...
EXPORT int	PwrSnsr_GetOverRan (SessionID Vi, int *OverRan)
	Get flag indicating whether the power meter's internal buffer filled up before being emptied. More...
EXPORT int	PwrSnsr_GetBufferedMeasurementsAvailable (SessionID Vi, int *MeasurementsAvailable)
	Gets the number of measurements available in the power meter's internal buffer. Note: The

	number of readings that have been acquired may be more or less. More...
EXPORT int	PwrSnsr_GetMeasBuffEnabled (SessionID Vi, int *MeasBuffEnabled)
	Get whether the measurement buffer has been enabled. More...
EXPORT int	PwrSnsr_SetMeasBuffEnabled (SessionID Vi, int MeasBuffEnabled)
	Enable or disable the measurement buffer. Disabling the measurement buffer enables modulated/CW measurements. Conversely, enabling it disables modulated/CW measurements. More...
EXPORT int	PwrSnsr_ResetContinuousCapture (SessionID Vi)
	Sets a flag indicating to restart continuous capture. This method allows the user to restart continuous acquisition. Has no effect if ContinuousCapture is set to false. More...
EXPORT int	PwrSnsr_GetEndDelay (SessionID Vi, float *EndDelay)
	Get delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst. More...
EXPORT int	PwrSnsr_SetEndDelay (SessionID Vi, float EndDelay)
	Set delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst. More...
EXPORT int	PwrSnsr_GetStartQual (SessionID Vi, float *StartQual)
	Get the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst. More...

EXPORT int	PwrSnsr_SetStartQual (SessionID Vi, float StartQual)
	Set the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst. More...
EXPORT int	PwrSnsr_GetStartDelay (SessionID Vi, float *StartDelay)
	Get delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst. More...
EXPORT int	PwrSnsr_SetStartDelay (SessionID Vi, float StartDelay)
	Set delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst. More...
EXPORT int	PwrSnsr_GetEndQual (SessionID Vi, float *EndQual)
	Get the minimum amount of time power remains below the trigger point to be counted as the end of a burst. More...
EXPORT int	PwrSnsr_SetEndQual (SessionID Vi, float EndQual)
	Set the minimum amount of time power remains below the trigger point to be counted as the end of a burst. More...
EXPORT int	PwrSnsr_Write (SessionID Vi, const char *Channel, int DataBufferSize, unsigned char Data[])
	Write a byte array to the meter. More...
EXPORT int	PwrSnsr_ReadByteArray (SessionID Vi, const char *Channel, int Count, int ValBufferSize, unsigned char Val[], int *ValActualSize)
	Reads byte array from the meter. More...

EXPORT int	PwrSnsr_ReadControl (SessionID Vi, const char *Channel, int Count, int ValBufferSize, unsigned char Val[], int *ValActualSize)
	Reads a control transfer on the USB. More...
EXPORT int	PwrSnsr_SetContinuousCapture (SessionID Vi, int ContinuousCapture)
	Set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called. More...
EXPORT int	PwrSnsr_GetContinuousCapture (SessionID Vi, int *ContinuousCapture)
	Get whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called. More...
EXPORT int	PwrSnsr_GetModel (SessionID Vi, const char *Channel, int ModelBufferSize, char Model[])
	Gets the model of the meter connected to the specified channel. More...
EXPORT int	PwrSnsr_GetPeakHoldDecay (SessionID Vi, const char *Channel, int *EnvelopeAverage)
	Get the number of min/max traces averaged together to form the peak hold measurement results on the selected channel. More...
EXPORT int	PwrSnsr_GetPeakHoldTracking (SessionID Vi, const char *Channel, int *EnvelopeTracking)
	Returns whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent. More...

EXPORT int	PwrSnsr_SetPeakHoldTracking (SessionID Vi, const char *Channel, int EnvelopeTracking)
	Sets whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent. More...
EXPORT int	PwrSnsr_GetFirmwareVersion (SessionID Vi, const char *Channel, int FirmwareVersionBufferSize, char FirmwareVersion[])
	Returns the firmware version of the power meter associated with this channel. More...
EXPORT int	PwrSnsr_SetPeakHoldDecay (SessionID Vi, const char *Channel, int PeakHoldDecay)
	Set the number of min/max traces averaged together to form the peak hold measurement results on the selected channel. More...
EXPORT int	PwrSnsr_GetDongleSerialNumber (long *val)
	Get the hardware license serial number. More...
EXPORT int	PwrSnsr_GetExpirationDate (int *Date)
	Get the hardware license expiration date. More...
EXPORT int	PwrSnsr_GetNumberOfCals (long *val)
	Get the number of calibrations left on the license. More...
EXPORT int	PwrSnsr_IsLicenseDongleConnected (int *val)
	Get whether the hardware license dongle is connected. More...

Detailed Description

File containing all user-callable functions.

Typedef Documentation

◆ [PulseInfo](#)

typedef struct [PulseInfo](#) [PulseInfo](#)
Data structure containing pulse information.

◆ [PwrSnsrAcquisitionStatusEnum](#)

typedef enum [PwrSnsrAcquisitionStatusEnum](#) [PwrSnsrAcquisitionStatusEnum](#)

◆ [PwrSnsrBandwidthEnum](#)

typedef enum [PwrSnsrBandwidthEnum](#) [PwrSnsrBandwidthEnum](#)
Video bandwidth enumeration.

◆ [PwrSnsrCondCodeEnum](#)

typedef enum [PwrSnsrCondCodeEnum](#) [PwrSnsrCondCodeEnum](#)
Condition code indicating validity of the measurement.

◆ [PwrSnsrErrorCodesEnum](#)

typedef enum [PwrSnsrErrorCodesEnum](#) [PwrSnsrErrorCodesEnum](#)
Error codes

◆ [PwrSnsrFilterStateEnum](#)

typedef enum [PwrSnsrFilterStateEnum](#) [PwrSnsrFilterStateEnum](#)
Filter state enum.

◆ [PwrSnsrHoldoffModeEnum](#)

typedef enum [PwrSnsrHoldoffModeEnum](#) [PwrSnsrHoldoffModeEnum](#)
Trigger holdoff mode.

◆ [PwrSnsrMarkerNumberEnum](#)

typedef enum [PwrSnsrMarkerNumberEnum](#) [PwrSnsrMarkerNumberEnum](#)
Marker number enumeration.

◆ [PwrSnsrMeasBuffGateEnum](#)

typedef enum [PwrSnsrMeasBuffGateEnum](#) [PwrSnsrMeasBuffGateEnum](#)
Measurement buffer gate modes.

◆ [PwrSnsrMeasBuffStartModeEnum](#)

typedef enum [PwrSnsrMeasBuffStartModeEnum](#) [PwrSnsrMeasBuffStartModeEnum](#)
Measurement buffer start modes.

◆ [PwrSnsrMeasBuffStopReasonEnum](#)

typedef enum [PwrSnsrMeasBuffStopReasonEnum](#) [PwrSnsrMeasBuffStopReasonEnum](#)
Measurement buffer stop reason.

◆ [PwrSnsrPulseUnitsEnum](#)

typedef enum [PwrSnsrPulseUnitsEnum](#) [PwrSnsrPulseUnitsEnum](#)
Enum for pulse calculation units.

◆ [PwrSnsrRdgsEnableFlag](#)

typedef enum [PwrSnsrRdgsEnableFlag](#) [PwrSnsrRdgsEnableFlag](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

◆ [PwrSnsrStatGatingEnum](#)

typedef enum [PwrSnsrStatGatingEnum](#) [PwrSnsrStatGatingEnum](#)
Gating value for statistical capture.

◆ [PwrSnsrTermActionEnum](#)

typedef enum [PwrSnsrTermActionEnum](#) [PwrSnsrTermActionEnum](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

◆ [PwrSnsrTriggerModeEnum](#)

typedef enum [PwrSnsrTriggerModeEnum](#) [PwrSnsrTriggerModeEnum](#)
Trigger mode for synchronizing data acquisition with pulsed signals.

◆ [PwrSnsrTriggerPositionEnum](#)

typedef enum [PwrSnsrTriggerPositionEnum](#) [PwrSnsrTriggerPositionEnum](#)
Set or return the position of the trigger event on displayed sweep.

◆ [PwrSnsrTriggerSlopeEnum](#)

typedef enum [PwrSnsrTriggerSlopeEnum](#) [PwrSnsrTriggerSlopeEnum](#)
Values for edge trigger slope

◆ [PwrSnsrTriggerSourceEnum](#)

typedef enum [PwrSnsrTriggerSourceEnum](#) [PwrSnsrTriggerSourceEnum](#)
Trigger source used for synchronizing data acquisition.

◆ [PwrSnsrTriggerStatusEnum](#)

typedef enum [PwrSnsrTriggerStatusEnum](#) [PwrSnsrTriggerStatusEnum](#)
Trigger status of the acquisition system.

◆ [PwrSnsrTrigOutModeEnum](#)

typedef enum [PwrSnsrTrigOutModeEnum](#) [PwrSnsrTrigOutModeEnum](#)

Multi IO trigger out modes.

◆ [PwrSnsrUnitsEnum](#)

typedef enum [PwrSnsrUnitsEnum](#) [PwrSnsrUnitsEnum](#)

Units returned by channel measurements.

Enumeration Type Documentation

◆ [PwrSnsrAcquisitionStatusEnum](#)

enum [PwrSnsrAcquisitionStatusEnum](#)

Enumerator	
PwrSnsrAcqComplete	The meter has completed the acquisition..
PwrSnsrAcqInProgress	The meter is still acquiring data.
PwrSnsrAcqStatusUnknown	The meter cannot determine the status of the acquisition.

◆ [PwrSnsrBandwidthEnum](#)

enum [PwrSnsrBandwidthEnum](#)

Video bandwidth enumeration.

Enumerator	
PwrSnsrBandwidthHigh	High bandwidth.
PwrSnsrBandwidthLow	Low bandwidth.

◆ [PwrSnsrCondCodeEnum](#)

enum [PwrSnsrCondCodeEnum](#)

Condition code indicating validity of the measurement.

Enumerator	
PwrSnsrCondCodeMeasurementStopped	Measurement is STOPPED. Value returned is not updated.
PwrSnsrCondCodeError	Error return. Measurement is not valid.
PwrSnsrCondCodeUnderrange	An Over-range condition exists.
PwrSnsrCondCodeOvrange	An Under-range condition exists.
PwrSnsrCondCodeNormal	Normal return. No error.

◆ PwrSnsrErrorCodesEnum

enum [PwrSnsrErrorCodesEnum](#)

Error codes

Enumerator	
PWR_SNSR_IO_GENERAL	I/O error.
PWR_SNSR_IO_TIMEOUT	I/O timeout error.
PWR_SNSR_MODEL_NOT_SUPPORTED	Instrument model does not support this feature.
PWR_SNSR_INV_PARAMETER	Invalid parameter value
PWR_SNSR_ERROR_INVALID_SESSION_HANDLE	Session ID invalid.
PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE	Status not available.
PWR_SNSR_ERROR_RESET_FAILED	Reset failed.
PWR_SNSR_ERROR_RESOURCE_UNKNOWN	Unknown resource descriptor.
PWR_SNSR_ERROR_ALREADY_INITIALIZED	Session already initialized.
PWR_SNSR_ERROR_OUT_OF_MEMORY	Out of memory.
PWR_SNSR_ERROR_OPERATION_PENDING	Operation pending.

PWR_SNSR_ERROR_NULL_POINTER	Null pointer not allowed.
PWR_SNSR_ERROR_UNEXPECTED_RESPONSE	Unexpected response from the instrument.
PWR_SNSR_ERROR_NOT_INITIALIZED	Session not initialized.
PWR_SNSR_LIBUSB_ERROR_IO	Input/output error
PWR_SNSR_LIBUSB_ERROR_INVALID_PARAMETER	Invalid parameter
PWR_SNSR_LIBUSB_ERROR_ACCESS	Access denied (insufficient permissions)
PWR_SNSR_LIBUSB_ERROR_NO_DEVICE	No such device (it may have been disconnected)
PWR_SNSR_LIBUSB_ERROR_NOT_FOUND	Entity not found
PWR_SNSR_LIBUSB_ERROR_BUSY	Resource busy
PWR_SNSR_LIBUSB_ERROR_TIMEOUT	Operation timed out
PWR_SNSR_LIBUSB_ERROR_OVERFLOW	Overflow
PWR_SNSR_LIBUSB_ERROR_PIPE	Pipe error
PWR_SNSR_LIBUSB_ERROR_INTERRUPTED	System call interrupted (perhaps due to signal)
PWR_SNSR_LIBUSB_ERROR_NO_MEMORY	Insufficient memory
PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED	Operation not supported or unimplemented on this platform
PWR_SNSR_LIBUSB_ERROR_OTHER	Other error

◆ PwrSnsrFilterStateEnum

enum [PwrSnsrFilterStateEnum](#)

Filter state enum.

Enumerator	
PwrSnsrFilterStateOff	Filter off.

PwrSnsrFilterStateOn	Filter on.
PwrSnsrFilterStateAuto	Automatically calculated filter.

◆ PwrSnsrHoldoffModeEnum

enum [PwrSnsrHoldoffModeEnum](#)

Trigger holdoff mode.

Enumerator	
PwrSnsrHoldoffModeNormal	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	Trigger will not arm again after the trigger conditions are satisfied and then the amount of time set for trigger holdoff.

◆ PwrSnsrMarkerNumberEnum

enum [PwrSnsrMarkerNumberEnum](#)

Marker number enumeration.

Enumerator	
PwrSnsrMarkerNumberMarker1	Marker 1
PwrSnsrMarkerNumberMarker2	Marker2

◆ PwrSnsrMeasBuffGateEnum

enum [PwrSnsrMeasBuffGateEnum](#)

Measurement buffer gate modes.

◆ PwrSnsrMeasBuffStartModeEnum

enum [PwrSnsrMeasBuffStartModeEnum](#)

Measurement buffer start modes.

◆ [PwrSnsrMeasBuffStopReasonEnum](#)

enum [PwrSnsrMeasBuffStopReasonEnum](#)

Measurement buffer stop reason.

◆ [PwrSnsrPulseUnitsEnum](#)

enum [PwrSnsrPulseUnitsEnum](#)

Enum for pulse calculation units.

Enumerator	
PwrSnsrPulseUnitsWatts	Calculates distal, mesial, and proximal using watts.
PwrSnsrPulseUnitsVolts	Calculates distal, mesial, and proximal using volts.

◆ [PwrSnsrRdgsEnableFlag](#)

enum [PwrSnsrRdgsEnableFlag](#)

Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

Enumerator	
PwrSnsrSequenceEnable	Enable sequence array capture.
PwrSnsrStartTimeEnable	Enable start time array capture.
PwrSnsrDurationEnable	Enable duration array capture.
PwrSnsrMinEnable	Enable min measurement array capture.
PwrSnsrAvgEnable	Enable average measurement capture.
PwrSnsrMaxEnable	Enable max measurement capture.

◆ [PwrSnsrStatGatingEnum](#)

enum [PwrSnsrStatGatingEnum](#)
Gating value for statistical capture.

Enumerator	
PwrSnsrStatGatingFreeRun	No gating.
PwrSnsrStatGatingMarkers	Gating is constrained to the portion of the the trace between the markers.

◆ [PwrSnsrTermActionEnum](#)

enum [PwrSnsrTermActionEnum](#)
Select the action to take when either the statistical terminalcount is reached or the terminal time has elapsed.

Enumerator	
PwrSnsrTermActionStop	Stop accumulating samples and hold the result.
PwrSnsrTermActionRestart	Clear the CCDF and begin a new one.
PwrSnsrTermActionDecimate	Divide all sample bins by 2 and continue.

◆ [PwrSnsrTriggerModeEnum](#)

enum [PwrSnsrTriggerModeEnum](#)
Trigger mode for synchronizing data acquisition with pulsed signals.

Enumerator	
PwrSnsrTriggerModeNormal	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	The power meter automatically triggers if the configured trigger does not occur within the meter's timeout period.

PwrSnsrTriggerModeAutoLevel	The power meter automatically adjusts the trigger level the trigger level to halfway between the highest and lowest power levels detected.
-----------------------------	--

◆ PwrSnsrTriggerPositionEnum

enum [PwrSnsrTriggerPositionEnum](#)

Set or return the position of the trigger event on displayed sweep.

Enumerator	
PwrSnsrTriggerPositionLeft	Left trigger position.
PwrSnsrTriggerPositionMiddle	Middle trigger position.
PwrSnsrTriggerPositionRight	Right trigger position.

◆ PwrSnsrTriggerSlopeEnum

enum [PwrSnsrTriggerSlopeEnum](#)

Values for edge trigger slope

Enumerator	
PwrSnsrTriggerSlopePositive	A negative (falling) edge passing through the trigger level triggers the power meter.
PwrSnsrTriggerSlopeNegative	A positive (rising) edge passing through the trigger level triggers the power meter.

◆ PwrSnsrTriggerSourceEnum

enum [PwrSnsrTriggerSourceEnum](#)

Trigger source used for synchronizing data acquisition.

Enumerator	
PwrSnsrTriggerSourceChannel1	Channel 1

PwrSnsrTriggerSourceExternal	EXT setting uses the signal applied to the rear MULTI I/O connector.
PwrSnsrTriggerSourceChannel2	Channel 2
PwrSnsrTriggerSourceChannel3	Channel 3
PwrSnsrTriggerSourceChannel4	Channel 4
PwrSnsrTriggerSourceChannel5	Channel 5
PwrSnsrTriggerSourceChannel6	Channel 6
PwrSnsrTriggerSourceChannel7	Channel 7
PwrSnsrTriggerSourceChannel8	Channel 8
PwrSnsrTriggerSourceChannel9	Channel 9
PwrSnsrTriggerSourceChannel10	Channel 10
PwrSnsrTriggerSourceChannel11	Channel 11
PwrSnsrTriggerSourceChannel12	Channel 12
PwrSnsrTriggerSourceChannel13	Channel 13
PwrSnsrTriggerSourceChannel14	Channel 14
PwrSnsrTriggerSourceChannel15	Channel 15
PwrSnsrTriggerSourceChannel16	Channel 16
PwrSnsrTriggerSourceIndependent	Sets each sensor in a measurment group to use its own internal trigger.

◆ PwrSnsrTriggerStatusEnum

enum [PwrSnsrTriggerStatusEnum](#)

Trigger status of the acquisition system.

Enumerator	
PwrSnsrTriggerStatusStopped	Acquisition is stopped.

PwrSnsrTriggerStatusPretrig	Acquiring data and waiting for the pre-trigger to be satisfied.
PwrSnsrTriggerStatusWaiting	Meter is armed and waiting for trigger event.
PwrSnsrTriggerStatusAcquiringNew	Acquiring new data.
PwrSnsrTriggerStatusAutoTrig	Meter is autotriggering.
PwrSnsrTriggerStatusFreerun	Trigger is in free-run mode.
PwrSnsrTriggerStatusTriggered	Meter is currently triggered.
PwrSnsrTriggerStatusRunning	Acquisition is running.

◆ PwrSnsrTrigOutModeEnum

enum [PwrSnsrTrigOutModeEnum](#)

Multi IO trigger out modes.

◆ PwrSnsrUnitsEnum

enum [PwrSnsrUnitsEnum](#)

Units returned by channel measurements.

Enumerator	
PwrSnsrUnitsdBm	dBm
PwrSnsrUnitswatts	Watts
PwrSnsrUnitsvolts	Volts
PwrSnsrUnitsDBV	dBV
PwrSnsrUnitsDBMV	dBmV
PwrSnsrUnitsDBUV	dBuV

Function Documentation

◆ PwrSnsr_Abort()

```
EXPORT int
PwrSnsr_Abort
t          (          SessionID      Vi          )
```

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.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_AcquireMeasurements()

```
EXPORT int
PwrSnsr_AcquireMeasurements
(          SessionID      Vi,
          double          Timeout,
          int             Count,
          PwrSnsrMeasBuffStopReasonEnum * StopReason,
          int *           Val
          )
```

Initiates new acquisition from the measurement buffer system (if acquisition is in the stopped state). Blocks until the number of measurements for each enabled channel is equal to count, or a time out has occurred.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timeout Maximum time in seconds to continue acquiring samples. Negative values will be treated as infinite.

Count Number of samples to acquire.

StopReason Reason acquisition stopped.

Val Number of samples acquired.

Returns

Success (0) or error code.

◆ PwrSnsr_AdvanceReadIndex()

```
EXPORT int
PwrSnsr_AdvanceReadIndex
```

```
x ( SessionID Vi )
```

Send a command to the meter to notify it the user is done reading and to advance the read index.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_Clear()

```
EXPORT int
PwrSnsr_Clear
```

```
r ( SessionID Vi )
```

Clear all data buffers. Clears averaging filters to empty.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_ClearBuffer()

```
EXPORT int
PwrSnsr_ClearBuffer
```

```
rBuffer ( SessionID Vi )
```

Sends a command to the power meter to clear all buffered readings. This method does not clear cached measurements accessible through `GetAverageMeasurements`, etc.

Parameters

Vi

The SessionID handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ `PwrSnsr_ClearError()`

```
EXPORT int
PwrSnsr_ClearError (          SessionID  Vi          )
```

This function clears the error code and error description for the given session.

Parameters

Vi

The SessionID handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ `PwrSnsr_ClearMeasurements()`

```
EXPORT int
PwrSnsr_ClearMeasurements (          SessionID  Vi          )
```

Clears cached average, min, max, duration, start time, and sequence number measurements.

Parameters

Vi

The SessionID handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_ClearUserCal()

```
EXPORT int
PwrSnsr_ClearUserCal (
                                SessionID      Vi,
                                const char *    Channel
)
```

Resets the value of fixed cal, zero, and skew to factory defaults.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_close()

```
EXPORT int
PwrSnsr_close (
                SessionID  Vi
)
```

Closes the I/O session to the instrument. Driver methods and properties that access the instrument are not accessible after Close is called.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_EnableCapturePriority()

```
EXPORT int
PwrSnsr_EnableCapturePriority (
                                SessionID      Vi,
                                const char *    Channel,
                                int             Enabled
)
```

)
Sets the 55 series power meter to a buffered capture mode and disables real time processing.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Enabled	If set to 1, enables buffered mode. If set to zero, disables capture priority(default).

Returns

Success (0) or error code.

◆ PwrSnsr_FetchAllMultiPulse()

```
EXPORT int
PwrSnsr_FetchAllMultiPulse (
    SessionID          Vi,
    const char *       Channel,
    int                 PulseInfosSize,
    PulseInfo          PulseInfos[],
    int *               PulseInfosActualSize
)
```

Return all previously acquired multiple pulse measurements. The elements in the PulseInfos array correspond to pulses on the current trace from left to right (ascending time order).

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PulseInfosSize	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()

```
EXPORT int
PwrSnsr_FetchArray
MarkerPower (
```

SessionID	Vi,
const char *	Channel,
float *	AvgPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	AvgPowerCondCode,
float *	MaxPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	MaxPowerCondCode,
float *	MinPower,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	MinPowerCondCode,
float *	PkToAvgRatio,
<u>PwrSnsrCondCode</u>	PkToAvgRatioCondC
<u>Enum</u> *	ode,
float *	Marker1Power,
<u>PwrSnsrCondCode</u>	Marker1PowerCondC
<u>Enum</u> *	ode,
float *	Marker2Power,
<u>PwrSnsrCondCode</u>	Marker2PowerCondC
<u>Enum</u> *	ode,
float *	MarkerRatio,
<u>PwrSnsrCondCode</u>	MarkerRatioCondCod
<u>Enum</u> *	e

)

Returns an array of the current marker measurements for the specified channel.

Parameters

vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

AvgPower

Average power between the markers.

AvgPowerCondCode

Condition code.

MaxPower

Maximum power between 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()**

```
EXPORT int
PwrSnsr_FetchCCDF
Percent          (          SessionID          Vi,
                   const char * Channel,
                   double Power,
                   PwrSnsrCondCode
Enum * CondCode,
                   double * Val
                   )
```

Return relative power (in dB) for a given percent on the CCDF plot.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Power	Relative power in dB
CondCode	Condition code for the measurement.
Val	Percent measurement at power.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchCCDFPower()

```
EXPORT int
PwrSnsr_FetchCCDF
Power          (
                SessionID      Vi,
                const char *   Channel,
                double         Percent,
                PwrSnsrCondCode
                Enum *             CondCode,
                double *       Val
                )
```

Return relative power (in dB) for a given percent on the CCDF plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Percent

Statistical percent to retrieve power from.

CondCode

Condition code for the measurement.

Val

relative power at percent.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchCCDFTrace()

```
EXPORT int
PwrSnsr_FetchCCDF
Trace          (
                SessionID      Vi,
                const char *   Channel,
                int             TraceBufferSize,
                float          Trace[],
                int *           TraceActualSize
                )
```

Returns the points in the CCDF trace.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TraceBufferSize**Trace****TraceActualSize****Returns**

Success (0) or error code.

◆ [PwrSnsr_FetchCursorPercent\(\)](#)

```
EXPORT int
PwrSnsr_FetchCursorPercent (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    double *           Val
)
```

Returns the percent CCDF at the cursor.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val**Returns**

Success (0) or error code.

◆ [PwrSnsr_FetchCursorPower\(\)](#)

```
EXPORT int
PwrSnsr_FetchCursorPower (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    double *           Val
)
```

Returns the power CCDF in dB at the cursor.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val

Returns

Success (0) or error code.

◆ PwrSnsr_FetchCWArray()

```
EXPORT int
PwrSnsr_FetchCWAr
ray          (
```

```
    SessionID          Vi,
    const char *       Channel,
    float *            PeakAverage,
    PwrSnsrCondCode
Enum *              PeakAverageValid,
    float *            PeakMax,
    PwrSnsrCondCode
Enum *              PeakMaxValid,
    float *            PeakMin,
    PwrSnsrCondCode
Enum *              PeakMinValid,
    float *            PeakToAvgRatio,
    PwrSnsrCondCode
Enum *              PeakToAvgRatioValid
```

```
)
```

Returns the current average, maximum, minimum powers or voltages and the peak-to-average ratio of the specified channel. Units are the same as the channel units. Note the peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PeakAverage

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()

```

EXPORT int
PwrSnsr_FetchCWPower (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *             Val
)

```

Returns the most recently acquired CW power.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val

CW power in channel units.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchDistal()

```

EXPORT int
PwrSnsr_FetchDistal (
    SessionID          Vi,

```

```

const char *      Channel,
PwrSnsrCondCode
Enum *          CondCode,
float *           Val

```

```
)
```

Returns the actual detected power of the distal level in the current channel units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val

Detected power of the distal level in the current channel units.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchDutyCycle\(\)](#)

```

EXPORT int
PwrSnsr_FetchDutyC
ycle          (
                SessionID      Vi,
                const char *    Channel,
                PwrSnsrCondCode
Enum *      IsValid,
                float *         Val
            )

```

Returns the ratio of the pulse on-time to off-time.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchEdgeDelay()

```
EXPORT int
PwrSnsr_FetchEdgeDelay (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * IsValid,
    float *            Val
)
```

Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Val

Returns

Success (0) or error code.

◆ PwrSnsr_FetchExtendedWaveform()

```
EXPORT int
PwrSnsr_FetchExtendedWaveform (
    SessionID          Vi,
    const char *       Channel,
    int                WaveformArrayBuffer
                        Size,
    float              WaveformArray[],
    int *              WaveformArrayActual
                        Size,
    int                Count
)
```

When capture priority is enabled, returns up to 100000 points of trace data based on the current timebase starting at the current trigger delay point.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
WaveformArrayBufferSize	Number of elements in the WaveformArray buffer
WaveformArray	Waveform buffer.
WaveformArrayActualSize	Number of elements updated with data.
Count	Number of points to capture.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchFallTime()

```
EXPORT int
PwrSnsr_FetchFallTime (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * IsValid,
    float *            Val
)
```

Returns the interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIEEEBottom()

```
EXPORT int
PwrSnsr_FetchIEEE (
    SessionID          Vi,
```

Bottom

```
const char *      Channel,
PwrSnsrCondCode
Enum *           IsValid,
float *          Val
```

)

Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIEEETop()

```
EXPORT int
PwrSnsr_FetchIEEETop
op          (
```

```
SessionID    Vi,
const char *  Channel,
PwrSnsrCondCode
Enum *       IsValid,
float *      Val
```

)

Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalAvg\(\)](#)

```
EXPORT int
PwrSnsr_FetchIntervalAvg (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalFilteredMax\(\)](#)

```
EXPORT int
PwrSnsr_FetchIntervalFilteredMax (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalFilteredMin()**

```
EXPORT int
PwrSnsr_FetchIntervalFilteredMin (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

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.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_FetchIntervalMax()**

```
EXPORT int
PwrSnsr_FetchIntervalMax (
    SessionID          Vi,
```

alMax

const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * CondCode,
float * Val

)

Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchIntervalMaxAvg\(\)](#)

EXPORT int
PwrSnsr_FetchInterv
alMaxAvg (

SessionID Vi,
const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * CondCode,
float * Val

)

Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIntervalMin()

```
EXPORT int
PwrSnsr_FetchIntervalMin (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIntervalMinAvg()

```
EXPORT int
PwrSnsr_FetchIntervalMinAvg (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchIntervalPkToAvg()

```
EXPORT int
PwrSnsr_FetchIntervalPkToAvg (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCode    Enum * CondCode,
    float *            Val
)
```

Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerAverage()

```
EXPORT int
PwrSnsr_FetchMarkerAverage (
    SessionID          Vi,
    const char *       Channel,
    int                 Marker,
    PwrSnsrCondCodeEnum * IsValid,
    float *             Val
)
```

For the specified marker, return the average power or voltage at the marker. The units are the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Marker

Marker number.

IsValid

Condition code.

Val

Measurement value

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerDelta()

```
EXPORT int
PwrSnsr_FetchMarkerDelta (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *             Val
)
```

Return the difference between MK1 and MK2. The units will be the same as marker units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

CondCode

Channel number. For single instruments, set this to "CH1".

Val

Condition code for the measurement.

Returns

Measurement value.

Success (0) or error code.

◆ PwrSnsr_FetchMarkerMax()

```
EXPORT int
PwrSnsr_FetchMarke
rMax      (
```

```
SessionID      Vi,
const char *   Channel,
int            Marker,
PwrSnsrCondCode
Enum *         IsValid,
float *        Val
)
```

For the specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Marker

Marker number.

IsValid**Val**

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerMin()

```
EXPORT int
PwrSnsr_FetchMarke
rMin      (
```

```
SessionID      Vi,
const char *   Channel,
```

```
int                                     Marker,  
PwrSnsrCondCode  
Enum *                               IsValid,  
float *                               Val  
)
```

For the specified marker, return the minimum power or voltage at the marker. The units are the same as the specified channel.

Parameters

- Vi** The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
- Channel** Channel number. For single instruments, set this to "CH1".
- Marker** Marker number.
- IsValid**
- Val** measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_FetchMarkerRatio\(\)](#)

```
EXPORT int  
PwrSnsr_FetchMarke  
rRatio (                               SessionID      Vi,  
                                             const char *   Channel,  
                                             PwrSnsrCondCode  
                                             Enum *         CondCode,  
                                             float *         Val  
)
```

Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units.

Parameters

- Vi** The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
- Channel** Channel number. For single instruments, set this to "CH1".
- CondCode** Condition code for the measurement. Condition code.
- Val** Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerRDelta()

```
EXPORT int
PwrSnsr_FetchMarke
rDelta          (          SessionID          Vi,
                  const char *          Channel,
                  PwrSnsrCondCode          condCode,
                  Enum *          Val
                  float *
                  )
```

Return the difference between MK2 and MK1. The units will be the same as marker units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchMarkerRRatio()

```
EXPORT int
PwrSnsr_FetchMarke
rRRatio          (          SessionID          Vi,
                  const char *          Channel,
                  PwrSnsrCondCode          CondCode,
                  Enum *          Val
                  float *
                  )
```

Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchMesial()

```
EXPORT int
PwrSnsr_FetchMesial (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCode   CondCode,
    Enum *             Val,
    float *
)
```

Returns the actual detected power of the mesial level in the current channel units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Detected power of the mesial level in the current channel units.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchOfftime()

```
EXPORT int
PwrSnsr_FetchOfftime
e (
    SessionID          Vi,
    const char *       Channel,
```

PwrSnsrCondCodeEnum *

float *

IsValid,

Val

)

Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulsewidth).

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchOvershoot()

EXPORT int

PwrSnsr_FetchOvers

hoot

(

SessionID

Vi,

const char *

Channel,

PwrSnsrCondCodeEnum *

IsValid,

float *

Val

)

Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchPeriod()

```
EXPORT int
PwrSnsr_FetchPeriod (
    SessionID
    const char *
    PwrSnsrCondCode
    Enum *
    float *
    Vi,
    Channel,
    IsValid,
    Val
)
```

Returns the interval between two successive pulses. (Reciprocal of the Pulse RepetitionFrequency)

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchPowerArray()

```
EXPORT int
PwrSnsr_FetchPower
Array (
    SessionID
    const char *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    Vi,
    Channel,
    PulsePeak,
    PulsePeakValid,
    PulseCycleAvg,
    PulseCycleAvgValid,
    PulseOnAvg,
    PulseOnValid,
    IEEETop,
)
```

<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	IEEETopValid,
float *	IEEEBottom,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	IEEEBottomValid,
float *	Overshoot,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	OvershootValid,
float *	Droop,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	DroopValid

)

Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform.

Measurements performed are: peak amplitude during the pulse, average amplitude over a full cycle of the pulse waveform, average amplitude during the pulse, IEEE top amplitude, IEEE bottom amplitude, and overshoot. Units are the same as the channel's units.

Note the pulse overshoot is returned in dB for logarithmic channel units, and percent for all other units. Also, the pulse ?ON interval used for peak

and average calculations is defined by the SENSE:PULSe:STARTGT and :ENDGT time gating settings.

A full pulse (rise and fall) must be visible on the display to make average and peak pulse power measurements, and a full cycle of the waveform must be visible to calculate average cycle amplitude.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
PulsePeak	The peak amplitude during the pulse.
PulsePeakValid	Condition code.
PulseCycleAvg	Average cycle amplitude.
PulseCycleAvgValid	Condition code.

PulseOnAvg**PulseOnValid****IEEETop**

Average power of the ON portion of the pulse.

Condition code.

The IEEE-defined top line, i.e. the portion of a pulse waveform, which represents the second nominal state of a pulse.

Condition code.

IEEETopValid**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.

Condition code.

IEEEBottomValid**Overshoot**

The difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.

Condition code.

OvershootValid**Droop**

Pulse droop.

DroopValid

Condition code.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchPRF()

EXPORT int

PwrSnsr_FetchPRF (

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

IsValid,

Val

)

Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchProximal()

```
EXPORT int
PwrSnsr_FetchProxi
mal          (          SessionID          Vi,
               const char *          Channel,
               PwrSnsrCondCode          CondCode,
               Enum *          Val
               float *
               )
```

Returns the actual detected power of the proximal level in the current channel units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val

Detected power of the proximal level in the current channel units.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchPulseCycleAvg()

```
EXPORT int
PwrSnsr_FetchPulse
CycleAvg      (          SessionID          Vi,
                     const char *          Channel,
                     PwrSnsrCondCode          IsValid,
                     Enum *          Val
                     float *
                     )
```

Returns the average power of the entire pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.
Returns	
Success (0) or error code.	

◆ PwrSnsr_FetchPulseOnAverage()

```
EXPORT int
PwrSnsr_FetchPulse
OnAverage      (
```

```
SessionID      Vi,
const char *   Channel,
PwrSnsrCondCode
Enum *        IsValid,
float *        Val
```

```
)
```

Average power of the ON portion of the pulse.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.
Returns	
Success (0) or error code.	

◆ PwrSnsr_FetchPulsePeak()

```
EXPORT int
PwrSnsr_FetchPulse
Peak      (
```

```
SessionID      Vi,
const char *   Channel,
PwrSnsrCondCode
Enum *        IsValid,
```

float * Val

)

Returns the peak amplitude during the pulse.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchRiseTime()

```
EXPORT int
PwrSnsr_FetchRiseTime
me (
```

SessionID Vi,
const char * Channel,
[PwrSnsrCondCode](#)
[Enum](#) * IsValid,
float * Val

)

Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchStatMeasurementArray()

```
EXPORT int
PwrSnsr_FetchStatM
easurementArray (
```

```
SessionID          Vi,
const char *       Channel,
double *           Pavg,
PwrSnsrCondCode
Enum *            PavgCond,
double *           Ppeak,
PwrSnsrCondCode
Enum *            PpeakCond,
double *           Pmin,
PwrSnsrCondCode
Enum *            PminCond,
double *           PkToAvgRatio,
PwrSnsrCondCode
Enum *            PkToAvgRatioCond,
double *           CursorPwr,
PwrSnsrCondCode
Enum *            CursorPwrCond,
double *           CursorPct,
PwrSnsrCondCode
Enum *            CursorPctCond,
double *           SampleCount,
PwrSnsrCondCode
Enum *            SampleCountCond,
double *           SecondsRun,
PwrSnsrCondCode
Enum *            SecondsRunCond
```

```
)
```

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.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel	Channel number. For single instruments, set this to "CH1".
Pavg	Long term average power in channel units.
PavgCond	Condition code.
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()

```

EXPORT int
PwrSnsr_FetchTimeA
rray          (
    SessionID
    const char *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    float *
    PwrSnsrCondCode
    Enum *
    Vi,
    Channel,
    Frequency,
    FrequencyValid,
    Period,
    PeriodValid,
    Width,
    WidthValid,
    Offtime,
    OfftimeValid,

```

float *	DutyCycle,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	DutyCycleValid,
float *	Risetime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	RisetimeValid,
float *	Falltime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	FalltimeValid,
float *	EdgeDelay,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	EdgeDelayValid,
float *	Skew,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	SkewValid

)

Returns an array of the current automatic timing measurements performed on a periodic pulse waveform.

Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window. Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions. It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements. Note that the timing of the edge transitions is defined by the settings of the SENSE:PULSE:DISTal, :MESIal and :PROXimal settings; see the descriptions Forthose commands. Units are the same as the channel's units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel	Channel number. For single instruments, set this to "CH1".
Frequency	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid	Condition code.
Period	The interval between two successive pulses.
PeriodValid	Condition code.
Width	The interval between the first and second signal crossings of the mesial line.
WidthValid	Condition code.
Offtime	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid	Condition code.
DutyCycle	The ratio of the pulse on-time to period.
DutyCycleValid	Condition code.
Risetime	The interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.
RisetimeValid	Condition code.
Falltime	The interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.
FalltimeValid	Condition code.
EdgeDelay	Time offset from the trigger reference to the first mesial transition level of either slope on the waveform.
EdgeDelayValid	Condition code.
Skew	The trigger offset between the assigned trigger channel and this channel.
SkewValid	Condition code.
Returns	
	Success (0) or error code.

◆ PwrSnsr_FetchWaveform()

```
EXPORT int
PwrSnsr_FetchWaveform (
```

```
SessionID
const char *
int
```

```
Vi,
Channel,
WaveformArrayBuffer
Size,
```

```
float WaveformArray[],
int * WaveformArrayActualSize
)

```

Returns a previously acquired waveform for this channel. The acquisition must be made prior to calling this method. Call this method separately for each channel.

Parameters

- Vi** The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
- Channel** Channel number. For single instruments, set this to "CH1".
- WaveformArrayBufferSize** Size in bytes of the Waveform buffer.
- WaveformArray** The array contains the average waveform. Units for the individual array elements are in the channel units setting.
- WaveformArrayActualSize** Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchWaveformMinMax()

```
EXPORT int
PwrSnsr_FetchWaveformMinMax (
    SessionID
    const char *
    int
    float
    int *
    int
    float
    int *
    int
    float
    int *
    Vi,
    Channel,
    MinWaveformBufferSize,
    MinWaveform[],
    MinWaveformActualSize,
    MaxWaveformBufferSize,
    MaxWaveform[],
    MaxWaveformActualSize,
    WaveformArrayBufferSize,
    WaveformArray[],
    WaveformArrayActualSize
)

```


)
Returns the previously acquired minimum and maximum waveforms for this specified channel. The acquisition must be made prior to calling this method. Call this method separately for each channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
MinWaveformBufferSize	Size in bytes of the MinWaveform buffer.
MinWaveform	This array contains the min waveform. Units for the individual array elements are in the channel units setting.
MinWaveformActualSize	Size in bytes of the data written to MinWaveform.
MaxWaveformBufferSize	Size in bytes of the MaxWaveform buffer.
MaxWaveform	This array contains the max waveform. Units for the individual array elements are in the channel units setting.
MaxWaveformActualSize	Size in bytes of the data written to MaxWaveform.
WaveformArrayBufferSize	Size in bytes of the Waveform buffer.
WaveformArray	The array contains the average waveform. Units for the individual array elements are in the channel units setting.
WaveformArrayActualSize	Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

◆ PwrSnsr_FetchWidth()

```
EXPORT int
PwrSnsr_FetchWidth (
    SessionID
    const char *
    PwrSnsrCondCode
    Enum *
    float *
    Vi,
    Channel,
    IsValid,
    Val
)
```

Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
IsValid	Condition code.
Val	Measurement return value.

Returns

Success (0) or error code.

◆ PwrSnsr_FindResources()

```
EXPORT int
PwrSnsr_FindResources (
    const char * Delimiter,
    int ValBufferSize,
    char Val[]
)
```

Returns a delimited string of available resources. These strings can be used in the initialize function to open a session to an instrument.

Parameters

Delimiter	The string used to delimit the list of resources ie. " ", " ", ":", etc.
ValBufferSize	Number of elements in Val.
Val	The return string.

Returns

Success (0) or error code.

◆ PwrSnsr_GetAcqStatusArray()

```
EXPORT int
PwrSnsr_GetAcqStatusArray (
    SessionID Vi,
    const char * Channel,
    int * SweepLength,
```

```
double *      SampleRate,
double *      SweepRate,
double *      SweepTime,
double *      StartTime,
int *         StatusWord
```

```
)
```

Returns data about the status of the acquisition system.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

SweepLength

Returns the number of points in the trace.

SampleRate

Returns the sample rate.

SweepRate

Returns the number of triggered sweeps per second.

SweepTime

Returns the sweep time for the trace.

StartTime

Returns the start time relative to the trigger.

StatusWord

Internal use - acquisition system status word.

Returns

Success (0) or error code.

◆ PwrSnsr_GetAttenuation()

```
EXPORT int
PwrSnsr_GetAttenuati
on          (          SessionID      Vi,
                  const char *      Channel,
                  float *            Attenuation
                  )
```

Attenuation in dB for the sensor.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Attenuation

Returns

Success (0) or error code.

◆ **PwrSnsr_GetAverage()**

```
EXPORT int
PwrSnsr_GetAverage (          SessionID          Vi,
                             const char *         Channel,
                             int *                Average
                             )
```

Get the number of traces averaged together to form the measurement result on the selected channel.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel Channel number. For single instruments, set this to "CH1".

Average

Returns

Success (0) or error code.

◆ **PwrSnsr_GetBandwidth()**

```
EXPORT int
PwrSnsr_GetBandwidth (          SessionID          Vi,
                               const char *         Channel,
                               PwrSnsrBandwidthE
                               num *                Bandwidth
                               )
```

Get the sensor video bandwidth for the selected sensor.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel Channel number. For single instruments, set this to "CH1".

Bandwidth

Returns

Success (0) or error code.

◆ `PwrSnsr_GetBufferedAverageMeasurements()`

```
EXPORT int
PwrSnsr_GetBuffered
AverageMeasurement
s (
    SessionID
    const char *
    int
    float
    int *
    Vi,
    Channel,
    ValBufferSize,
    Val[],
    ValActualSize
)
```

Get the average power measurements that were captured during the last call to `AcquireMeasurements`.

Parameters**Vi**

The SessionID handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ValBufferSize

Buffer size of Val.

Val

Array of average measurements.

ValActualSize

Actual size of Val.

Returns

Success (0) or error code.

◆ `PwrSnsr_GetBufferedMeasurementsAvailable()`

```
EXPORT int
PwrSnsr_GetBuffered
MeasurementsAvailab
le (
    SessionID
    int *
    Vi,
    MeasurementsAvailab
le
)
```

Gets the number of measurements available in the power meter's internal buffer. Note: The number of readings that have been acquired may be more or less.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MeasurementsAvailable

The number of measurements available in the 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()**

```
EXPORT int
PwrSnsr_GetCalFact
or          (          SessionID          Vi,
              const char *          Channel,
              float *          CalFactor
              )
```

Get the frequency calibration factor currently in use on the selected channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CalFactor**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetCalFactors()**

```
EXPORT int
PwrSnsr_GetCalFact
ors          (          SessionID          Vi,
                    const char *          Channel,
                    float *          MaxFrequency,
                    float *          MinFrequency,
                    int          FrequencyListBufferSize,
                    )
```

float	FrequencyList[],
int *	FrequencyListActualSize,
int	CalFactorListBufferSize,
float	CalFactorList[],
int *	CalFactorListActualSize,
<u>PwrSnsrBandwidthE</u>	<u>num</u>
	Bandwidth

)

Query information associated with calibration factors.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MaxFrequency

Maximum RF frequency measureable by this channel.

MinFrequency

Minimum RF frequency measureable by this channel.

FrequencyListBufferSize

Number of elements in FrequencyList.

FrequencyList

List of frequencies correlated to the cal factors.

FrequencyListActualSize

Actual number of elements returned in FrequencyList.

CalFactorListBufferSize

Number of elements in CalFactorList.

CalFactorList

List of cal factors correlated to the frequencies.

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\(\)](#)

EXPORT int

PwrSnsr_GetCapture (

SessionID

const char *

Vi,

Channel,

int *

Capture

)

Get whether statistical capture is enabled.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Capture

Returns

Success (0) or error code.

◆ PwrSnsr_GetCCDFTraceCount()

```
EXPORT int
PwrSnsr_GetCCDFTraceCount (
                                SessionID      Vi,
                                const char *    Channel,
                                int *          TraceCount
                                )
```

Get the number of points in the CCDF trace plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TraceCount

Returns

Success (0) or error code.

◆ PwrSnsr_GetChannelByIndex()

```
EXPORT int
PwrSnsr_GetChannelByIndex (
                                SessionID      Vi,
                                int            BuffSize,
                                char           Channel[]
                                )
```


int

Index

)

Gets the channel name by zero index. Note: SCPI commands use a one-based index.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Buffer size for Channel

Channel

Channel number buffer

Index

the index of the channel

Returns

Success (0) or error code.

◆ PwrSnsr_GetChannelCount()

EXPORT int

PwrSnsr_GetChannel

Count

(

SessionID

Vi,

int *

Count

)

Get number of channels.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Count

Number of channels

Returns

Success (0) or error code.

◆ PwrSnsr_GetChanTraceCount()

EXPORT int

PwrSnsr_GetChanTr

aceCount

(

SessionID

Vi,

const char *

int *

Channel,

TraceCount

)

Get the number of points in the CCDF trace plot.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TraceCount

The number of points in the CCDF trace plot.

Returns

Success (0) or error code.

◆ PwrSnsr_GetContinuousCapture()

```
EXPORT int
PwrSnsr_GetContinu
ousCapture      (          SessionID      Vi,
                  int *          ContinuousCapture
                  )
```

Get whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ContinuousCapture

True if AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Returns

Success (0) or error code.

◆ PwrSnsr_GetCurrentTemp()

```
EXPORT int
PwrSnsr_GetCurrent
Temp      (          SessionID      Vi,
                  const char *    Channel,
                  double *        CurrentTemp
                  )
```

Get current sensor internal temperature in degrees C.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CurrentTemp**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetDiagStatusArray()**

```
EXPORT int
PwrSnsr_GetDiagStat
usArray      (
```

```
SessionID    Vi,
const char *  Channel,
float *       DetectorTemp,
float *       CpuTemp,
float *       MioVoltage,
float *       VccInt10,
float *       VccAux18,
float *       Vcc50,
float *       Vcc25,
float *       Vcc33
```

```
)
```

Returns diagnostic data.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

DetectorTemp

Temperature in degrees C at the RF detector.

CpuTemp

Temperature of the CPU in degrees C.

MioVoltage

Voltage at the Multi I/O port.

VccInt10

Vcc 10 voltage.

VccAux18

Vcc Aux 18 voltage.

Vcc50

Vcc 50 voltage.

Vcc25

Vcc 25 voltage.

Vcc33

Vcc 33 voltage.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetDistal()**

```
EXPORT int
PwrSnsr_GetDistal (
    SessionID
    const char *
    float *
    Vi,
    Channel,
    Distal
)
```

Get the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Distal**Returns**

Success (0) or error code.

◆ **PwrSnsr_GetDongleSerialNumber()**

```
EXPORT int
PwrSnsr_GetDongleSerialN
umber (
    long *
    val
)
```

Get the hardware license serial number.

Parameters**val**

Serial number of the license dongle

Returns

Success (0) or error code.

◆ **PwrSnsr_GetDuration()**

```
EXPORT int
PwrSnsr_GetDuration (
    SessionID
    Vi,
```

float *

Duration

)

Get the time duration samples are captured during each timed mode acquisition.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Duration

The duration in seconds samples are captured during each timed mode acquisition.

Returns

Success (0) or error code.

◆ PwrSnsr_GetDurations()

```
EXPORT int
PwrSnsr_GetDuration
s          (          SessionID          Vi,
              const char *          Channel,
              int          ValBufferSize,
              float          Val[],
              int *          ValActualSize
              )
```

Get the duration entries in seconds that were captured during the last call to AcquireMeasurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ValBufferSize

Size of the buffer.

Val

Array of measurement durations in seconds.

ValActualSize

Actual size of the returned buffer.

Returns

Success (0) or error code.

◆ PwrSnsr_GetEnabled()

```
EXPORT int
PwrSnsr_GetEnabled (
                                SessionID
                                const char *
                                int *
                                Vi,
                                Channel,
                                Enabled
                                )
```

Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Enabled

Boolean. 1 for enabled; 0 for disabled.

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndDelay()

```
EXPORT int
PwrSnsr_GetEndDelay (
                                SessionID
                                float *
                                Vi,
                                EndDelay
                                )
```

Get delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

EndDelay

The delay time added to the detected end of a burst for analysis.

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndGate()

```
EXPORT int
PwrSnsr_GetEndGate
(
    SessionID Vi,
    const char * Channel,
    float * EndGate
)
```

Get the point on a pulse, which is used to define the end of the pulse's active interval.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

EndGate

Returns

Success (0) or error code.

◆ PwrSnsr_GetEndQual()

```
EXPORT int
PwrSnsr_GetEndQual
(
    SessionID Vi,
    float * EndQual
)
```

Get the minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

EndQual

The minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Returns

Success (0) or error code.

◆ PwrSnsr_GetError()

```

EXPORT int
PwrSnsr_GetError (
    SessionID
    int *
    int
    char
    Vi,
    ErrorCode,
    ErrorDescriptionBuffer
    rSize,
    ErrorDescription[]
)

```

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.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ErrorCode

ErrorDescriptionBufferSize

ErrorDescription

Returns

Success (0) or error code.

◆ PwrSnsr_GetExpirationDate()

```

EXPORT int
PwrSnsr_Get
ExpirationDate (
    int *
    Date
)

```

Get the hardware license expiration date.

Parameters

Date

expiration date in the format YYYYMMDD

Returns

Success (0) or error code.

◆ PwrSnsr_GetExternalSkew()

```

EXPORT int
PwrSnsr_GetExternal
Skew (
    SessionID
    const char *
    float *
    Vi,
    Channel,
    External
)

```


)
Gets the skew in seconds for the external trigger.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

External

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_GetFactoryCalDate()

```
EXPORT int
PwrSnsr_GetFactory
CalDate          (          SessionID          Vi,
                    const char *          Channel,
                    int          FactoryCalDateBuffer
                    char          Size,
                    FactoryCalDate[])
                    )
```

The date (YYYYmmDD) the last time the sensor was calibrated at the factory.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

FactoryCalDateBufferSize

Size of FactoryCalDate in bytes.

FactoryCalDate

Returns

Success (0) or error code.

◆ PwrSnsr_GetFetchLatency()

```
EXPORT int
PwrSnsr_GetFetchLa
tency          (          SessionID          Vi,
```

int *

Latency

)

Get the period the library waits to update fetch measurements in ms.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Latency

Fetch latency in ms.

Returns

Success (0) or error code.

◆ PwrSnsr_GetFilterState()

```
EXPORT int
PwrSnsr_GetFilterSta
te          (          SessionID          Vi,
              const char *          Channel,
              PwrSnsrFilterStateE
              num *          FilterState
              )
```

Get the current setting of the integration filter on the selected channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ManufactureDateBufferSize**ManufactureDate**

Returns

Success (0) or error code.

◆ PwrSnsr_GetFilterTime()

```
EXPORT int
PwrSnsr_GetFilterTi
me          (          SessionID          Vi,
              const char *          Channel,
              float *          FilterTime
              )
```

)
Get the current length of the integration filter on the selected channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

FilterTime

Returns

Success (0) or error code.

◆ PwrSnsr_GetFirmwareVersion()

```
EXPORT int
PwrSnsr_GetFirmwar
eVersion          (          SessionID          Vi,
                    const char * Channel,
                    int FirmwareVersionBuffer
                    char rSize,
                    FirmwareVersion[])
                    )
```

Returns the firmware version of the power meter associated with this channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

FirmwareVersionBufferSize

Size of the FirmwareVersion buffer.

FirmwareVersion

Buffer to hold the firmware version.

Returns

Success (0) or error code.

◆ PwrSnsr_GetFpgaVersion()

```
EXPORT int
PwrSnsr_GetFpgaVer
sion              (          SessionID          Vi,
```

```

    )
    Get the sensor FPGA version.
Parameters
    Vi
    Channel
    ValBufferSize
    Val
Returns
    Success (0) or error code.

```

```

const char *
int
char
Channel,
ValBufferSize,
Val[]

```

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel number. For single instruments, set this to "CH1".

Size of Val in bytes

Buffer for storing the version

◆ PwrSnsr_GetFrequency()

```

EXPORT int
PwrSnsr_GetFrequency
(
    SessionID
    const char *
    float *
    Vi,
    Channel,
    Frequency
)
Get the RF frequency for the current sensor.
Parameters
    Vi
    Channel
    Frequency
Returns
    Success (0) or error code.

```

```

SessionID
const char *
float *
Vi,
Channel,
Frequency

```

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel number. For single instruments, set this to "CH1".

RF Frequency in Hz.

◆ PwrSnsr_GetGateMode()

```

EXPORT int
PwrSnsr_GetGateMo (
    SessionID
    Vi,

```

de

[PwrSnsrMeasBuffGateEnum](#) *

GateMode

)

Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval. The gate signal may be internally or externally generated in several different ways.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

GateMode

Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code.

◆ PwrSnsr_GetGating()

EXPORT int

PwrSnsr_GetGating (

SessionID

const char *

Vi,

Channel,

[PwrSnsrStatGatingEnum](#) *

Gating

)

Get whether statistical capture is enabled.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1". whether the statical capture is gated by markers or free-running.

Gating

Returns

Success (0) or error code.

◆ PwrSnsr_GetHorizontalOffset()

```
EXPORT int
PwrSnsr_GetHorizontalOffset (
    SessionID
    const char *
    double *
    Vi,
    Channel,
    HorizontalOffset
)
```

Get the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative).

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

HorizontalOffset

Returns

Success (0) or error code.

◆ PwrSnsr_GetHorizontalScale()

```
EXPORT int
PwrSnsr_GetHorizontalScale (
    SessionID
    const char *
    double *
    Vi,
    Channel,
    HorizontalScale
)
```

Get the statistical mode horizontal scale in dB/Div.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

HorizontalScale

Returns

Success (0) or error code.

◆ PwrSnsr_GetImpedance()

```
EXPORT int
PwrSnsr_GetImpedan
ce (
    SessionID
    const char *
    float *
    Vi,
    Channel,
    Impedance
)
```

Input impedance of the sensor.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Impedance

Returns

Success (0) or error code.

◆ PwrSnsr_GetInitiateContinuous()

```
EXPORT int
PwrSnsr_GetInitiateC
ontinuous (
    SessionID
    int *
    Vi,
    InitiateContinuous
)
```

Get the data acquisition mode for single or free-run measurements.

If INITiate:CONTInuous is set to ON, the instrument immediately begins taking measurements (Modulated, CW and Statistical Modes), or arms its trigger and takes a measurement each time a trigger occurs (Pulse Mode). If set to OFF, the measurement will begin (or be armed) as soon as the INITiate command is issued, and will stop once the measurement criteria (averaging, filtering or sample count) has been satisfied. Note that INITiate:IMMediate and READ commands are invalid when INITiate:CONTInuous is set to ON; however, by convention this situation does not result in a SCPI error.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

InitiateContinuous

Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code.

◆ PwrSnsr_GetInternalSkew()

```
EXPORT int
PwrSnsr_GetInternal
Skew          (          SessionID          Vi,
                const char * Channel,
                float * InternalSkew
                )
```

Gets the skew in seconds for the internal trigger.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

InternalSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsAvailable()

```
EXPORT int
PwrSnsr_GetIsAvaila
ble          (          SessionID          Vi,
                const char * Channel,
                int * IsAvailable
                )
```

Returns true if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsAvailable

True if modulated/CW measurement system is available. Will always return false if measurement buffer is enabled.

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsAvgSensor()

```
EXPORT int
PwrSnsr_GetIsAvgSensor (
    SessionID          Vi,
    const char *       Channel,
    int *              IsAvgSensor
)
```

Returns true if sensor is average responding (not peak detecting).

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsAvgSensor

True if sensor is average responding.

Returns

Success (0) or error code.

◆ PwrSnsr_GetIsRunning()

```
EXPORT int
PwrSnsr_GetIsRunning (
    SessionID          Vi,
    const char *       Channel,
    int *              IsRunning
)
```

Returns true if modulated/CW measurements are actively running.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsRunning

True if modulated/CW measurements are actively running.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetManufactureDate()**

```
EXPORT int
PwrSnsr_GetManufactureDate (
                                SessionID      Vi,
                                const char *    Channel,
                                int             ManufactureDateBufferSize,
                                char           ManufactureDate[]
)
```

Date the sensor was manufactured in the following format YYYYmmDD.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ManufactureDateBufferSize

Size of ManufactureDate in bytes.

ManufactureDate

Return value.

Returns

Success (0) or error code.

◆ **PwrSnsr_GetMarkerPixelPosition()**

```
EXPORT int
PwrSnsr_GetMarkerPixelPosition (
                                SessionID      Vi,
                                int             MarkerNumber,
                                int *          PixelPosition
)
```

Get the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber**PixelPosition****Returns**

Success (0) or error code.

◆ PwrSnsr_GetMarkerTimePosition()

```
EXPORT int
PwrSnsr_GetMarkerTimePosition (
                                SessionID      Vi,
                                int             MarkerNumber,
                                float *        TimePosition
)
```

Get the time (x-axis-position) of the selected marker relative to the trigger.

Note 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.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber**TimePosition****Returns**

Success (0) or error code.

◆ PwrSnsr_GetMaxFreqHighBandwidth()

```
EXPORT int
PwrSnsr_GetMaxFreqHighBandwidth (
                                SessionID      Vi,
                                const char *    Channel,
                                float *        MaxFreqHighBandwidth
)
```

Maximum frequency carrier the sensor can measure in high bandwidth.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MaxFreqHighBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMaxFreqLowBandwidth()

```
EXPORT int
PwrSnsr_GetMaxFreqLowBandwidth (
                                SessionID      Vi,
                                const char *    Channel,
                                float *         MaxFreqLowBandwidth
                                )
```

Maximum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MaxFreqLowBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMaxMeasurements()

```
EXPORT int
PwrSnsr_GetMaxMeasurements (
                                SessionID      Vi,
                                const char *    Channel,
                                int             ValBufferSize,
                                float           Val[],
                                )
```

int * ValActualSize

)

Get the maximum power measurements that were captured during the last call to AcquireMeasurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

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()

```
EXPORT int
PwrSnsr_GetMaxTimebase (
    SessionID Vi,
    float * MaxTimebase
)
```

Gets the maximum timebase setting available.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MaxTimebase

Returns

Success (0) or error code.

◆ PwrSnsr_GetMeasBuffEnabled()

```
EXPORT int
PwrSnsr_GetMeasBuffEnabled (
    SessionID Vi,
    int * MeasBuffEnabled
)
```

Get whether the measurement buffer has been enabled.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MeasBuffEnabled

True if measurement buffer is enabled.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMeasurementsAvailable()

```
EXPORT int
PwrSnsr_GetMeasurementsAvailable (
                                SessionID      Vi,
                                const char *    Channel,
                                int *           Val
                                )
```

Get the number of measurement entries available that were captured during AcquireMeasurements().

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Val

Number of measurement entries available.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMemChanArchive()

```
EXPORT int
PwrSnsr_GetMemChanArchive (
                                SessionID      Vi,
                                const char *    memChan,
                                int             ValBufferSize,
                                char            Val[]
                                )
```

Returns an XML document containing settings and readings obtained using the SaveToMemoryChannel method.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
MemChan	The name of the memory channel to get the archive from.
ValBufferSize	
Val	XML document containing settings and readings.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMesial()

```
EXPORT int
PwrSnsr_GetMesial (          SessionID      Vi,
                             const char *   Channel,
                             float *        Mesial
                           )
```

Get the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Mesial	

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinFreqHighBandwidth()

```
EXPORT int
PwrSnsr_GetMinFreq
HighBandwidth (          SessionID      Vi,
                             const char *   Channel,
```

```

                                float *
                                MinFreqHighBandwidth
                                h
                                )

```

Minimum frequency of RF the sensor can measure in high bandwidth.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MinFreqHighBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinFreqLowBandwidth()

```

EXPORT int
PwrSnsr_GetMinFreq
LowBandwidth (
                                SessionID
                                const char *
                                Vi,
                                Channel,
                                MinFreqLowBandwidth
                                h
                                float *
                                )

```

Minimum frequency carrier the sensor can measure in low bandwidth.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MinFreqLowBandwidth

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinimumSupportedFirmware()

```

EXPORT int
PwrSnsr_Get
MinimumSupp
ortedFirmware (
                                int *
                                Version
                                )

```


Gets the minimum supported firmware as an integer. Format is YYYYMMDD.

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinimumTrig()

```
EXPORT int
PwrSnsr_GetMinimumTrig (
    SessionID          Vi,
    const char *        Channel,
    float *             MinimumTrig
)
```

Minimum internal trigger level in dBm.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MinimumTrig

Returns

Success (0) or error code.

◆ PwrSnsr_GetMinMeasurements()

```
EXPORT int
PwrSnsr_GetMinMeasurements (
    SessionID          Vi,
    const char *        Channel,
    int                 ValBufferSize,
    float               Val[],
    int *               ValActualSize
)
```

Get the minimum power measurements that were captured during the last call to AcquireMeasurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ValBufferSize

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()

```
EXPORT int
PwrSnsr_GetModel (
    SessionID      Vi,
    const char *   Channel,
    int            ModelBufferSize,
    char           Model[]
)
```

Gets the model of the meter connected to the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ModelBufferSize

Size of the buffer..

Model

Buffer where the model is read into.

Returns

Success (0) or error code.

◆ PwrSnsr_GetNumberOfCals()

```
EXPORT int
PwrSnsr_Get
NumberOfCal
s (long * val)
```

Get the number of calibrations left on the license.

Parameters**val**

Number of cals left.

Returns

Success (0) or error code.

◆ PwrSnsr_GetOffsetdB()

```
EXPORT int
PwrSnsr_GetOffsetdB
(
    SessionID          Vi,
    const char *       Channel,
    float *             OffsetdB
)
```

Get a measurement offset in dB for the selected sensor.

This setting is used to compensate for external couplers, attenuators or amplifiers in the RF signal path ahead of the power sensor.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

OffsetdB

Returns

Success (0) or error code.

◆ PwrSnsr_GetOverRan()

```
EXPORT int
PwrSnsr_GetOverRan
(
    SessionID          Vi,
    int *              OverRan
)
```

Get flag indicating whether the power meter's internal buffer filled up before being emptied.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

OverRan

True if the power meter's internal buffer filled up before being emptied.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakHoldDecay()

```
EXPORT int
PwrSnsr_GetPeakHoldDecay (
                                SessionID
                                const char *
                                int *
                                Vi,
                                Channel,
                                EnvelopeAverage
                                )
```

Get the number of min/max traces averaged together to form the peak hold measurement results on the selected channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

EnvelopeAverage

Out parameter value.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakHoldTracking()

```
EXPORT int
PwrSnsr_GetPeakHoldTracking (
                                SessionID
                                const char *
                                int *
                                Vi,
                                Channel,
                                EnvelopeTracking
                                )
```

Returns whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

EnvelopeTracking

Out boolean parameter value.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPeakPowerMax()

```
EXPORT int
PwrSnsr_GetPeakPowerMax (
                                SessionID      Vi,
                                const char *    Channel,
                                float *        PeakPowerMax
                                )
```

Maximum power level the sensor can measure.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PeakPowerMax**Returns**

Success (0) or error code.

◆ PwrSnsr_GetPeakPowerMin()

```
EXPORT int
PwrSnsr_GetPeakPowerMin (
                                SessionID      Vi,
                                const char *    Channel,
                                float *        PeakPowerMin
                                )
```

Minimum power level the sensor can measure.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PeakPowerMin

Returns

Success (0) or error code.

◆ PwrSnsr_GetPercentPosition()

```
EXPORT int
PwrSnsr_GetPercent
Position      (
                                SessionID      Vi,
                                const char *    Channel,
                                double *        PercentPosition
                                )
```

Get the cursor percent on the CCDF plot.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1". Channel Channel number. For single instruments, set this to 1.

PercentPosition**Returns**

Success (0) or error code.

◆ PwrSnsr_GetPeriod()

```
EXPORT int
PwrSnsr_GetPeriod  (
                                SessionID      Vi,
                                float *        Period
                                )
```

Get the period each timed mode acquisition (measurement buffer) is started.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Period

The period in seconds each timed mode acquisition is started.

Returns

Success (0) or error code.

◆ PwrSnsr_GetPowerPosition()

```
EXPORT int
PwrSnsr_GetPowerPosition (          SessionID          Vi,
                                   const char *          Channel,
                                   double *              PowerPosition
                                   )
```

Get the cursor power in dB on the CCDF plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PowerPosition

Returns

Success (0) or error code.

◆ PwrSnsr_GetProximal()

```
EXPORT int
PwrSnsr_GetProximal (          SessionID          Vi,
                              const char *          Channel,
                              float *              Proximal
                              )
```

Get the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Proximal

Returns

Success (0) or error code.

◆ PwrSnsr_GetPulseUnits()

```
EXPORT int
PwrSnsr_GetPulseUnits (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrPulseUnits Enum * Units
)
```

Get the units for entering the pulse distal, mesial and proximal levels.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PwrSnsrPulseUnitsEnum

Returns

Success (0) or error code.

◆ PwrSnsr_GetRdgsEnableFlag()

```
EXPORT int
PwrSnsr_GetRdgsEnableFlag (
    SessionID          Vi,
    int *              Flag
)
```

Get the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Flag

Bit masked value indicating which measurement arrays will be queried (see PwrSnsrRdgsEnableFlag).

Returns

Success (0) or error code.

◆ PwrSnsr_GetReadingPeriod()

```
EXPORT int
PwrSnsr_GetReadingPeriod (
                                SessionID
                                const char *
                                float *
                                Vi,
                                Channel,
                                ReadingPeriod
)
```

Returns the period (rate) in seconds per new filtered reading.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ReadingPeriod

The period (rate) in seconds per new filtered reading.

Returns

Success (0) or error code.

◆ PwrSnsr_GetReturnCount()

```
EXPORT int
PwrSnsr_GetReturnCount (
                                SessionID
                                int *
                                Vi,
                                ReturnCount
)
```

Get the return count for each measurement query.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ReturnCount

The return count for each measurement query.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSequenceNumbers()

```

EXPORT int
PwrSnsr_GetSequenceNumbers (
    SessionID
    const char *
    int
    long long
    int *
    Vi,
    Channel,
    ValBufferSize,
    Val[],
    ValActualSize
)

```

Get the sequence number entries that were captured during the last call to AcquireMeasurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

ValBufferSize

Size of the buffer.

Val

Array of sequence numbers.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSerialNumber()

```

EXPORT int
PwrSnsr_GetSerialNumber (
    SessionID
    const char *
    int
    char
    Vi,
    Channel,
    SerialNumberBufferSize,
    SerialNumber[]
)

```

Gets the serial number of the sensor.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

SerialNumberBufferSize

Size in bytes of Serial number.

SerialNumber

Out parameter. ASCII string serial number.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSessionCount()

```
EXPORT int
PwrSnsr_GetSession
Count          (          SessionID          Vi,
                  int *          SessionCount
                  )
```

Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

SessionCount

Get the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSlaveSkew()

```
EXPORT int
PwrSnsr_GetSlaveSk
ew          (          SessionID          Vi,
                  const char *          Channel,
                  float *          SlaveSkew
                  )
```

Gets the skew in seconds for the slave trigger.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

SlaveSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartDelay()

```
EXPORT int
PwrSnsr_GetStartDelay (
                                SessionID      Vi,
                                float *        StartDelay
)
```

Get delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartDelay

Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartGate()

```
EXPORT int
PwrSnsr_GetStartGate (
                                SessionID      Vi,
                                const char *    Channel,
                                float *        StartGate
)
```

Get the point on a pulse, which is used to define the beginning of the pulse's active interval.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

StartGate**Returns**

Success (0) or error code.

◆ PwrSnsr_GetStartMode()

```
EXPORT int
PwrSnsr_GetStartMode (
    SessionID Vi,
    PwrSnsrMeasBuffStartModeEnum * StartMode
)
```

Get the mode used to start acquisition of buffer entries.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartMode

Mode used to start acquisition of buffer entries.

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartQual()

```
EXPORT int
PwrSnsr_GetStartQual (
    SessionID Vi,
    float * StartQual
)
```

Get the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartQual

The minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Returns

Success (0) or error code.

◆ PwrSnsr_GetStartTimes()

```
EXPORT int
PwrSnsr_GetStartTimes (
    SessionID          Vi,
    const char *       Channel,
    int                 ValBufferSize,
    double              Val[],
    int *               ValActualSize
)
```

Get the start time entries in seconds that were captured during the last call to AcquireMeasurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of start times.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_GetSweepTime()

```
EXPORT int
PwrSnsr_GetSweepTime (
    SessionID          Vi,
    const char *       Channel,
    float *             SweepTime
)
```

Get sweep time for the trace in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".

SweepTime

Sweep time for the trace in seconds.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTempComp()

```
EXPORT int
PwrSnsr_GetTempComp (
    SessionID          Vi,
    const char *       Channel,
    int *              TempComp
)
```

Get the state of the peak sensor temperature compensation system.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TempComp

Boolean. 1 for on; 0 for off.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTermAction()

```
EXPORT int
PwrSnsr_GetTermAction (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrTermActionEnum * TermAction
)
```

Get the termination action for statistical capturing.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermAction**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTermCount()

```
EXPORT int
PwrSnsr_GetTermCo
unt          (          SessionID          Vi,
               const char *          Channel,
               double *          TermCount
               )
```

Get the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermCount**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTermTime()

```
EXPORT int
PwrSnsr_GetTermTi
me          (          SessionID          Vi,
               const char *          Channel,
               int *          TermTime
               )
```

Get the termination time in seconds for statistical capturing. After the time has elapsed, the action determined by TermAction is taken.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermTime**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTimebase()

```
EXPORT int
PwrSnsr_GetTimebase
(
    SessionID      Vi,
    float *        Timebase
)
```

Get the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 (or max timebase) sec in a 1-2-5 sequence,.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timebase**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTimedOut()

```
EXPORT int
PwrSnsr_GetTimedOut
(
    SessionID      Vi,
    int *          TimedOut
)
```

Check if the last measurement buffer session timed out.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

TimedOut

True if the last measurement buffer session timed out.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTimeOut()

```
EXPORT int
PwrSnsr_GetTimeOut
t                (                SessionID                Vi,
                                long *                Val
                )
```

Returns the time out value for I/O in milliseconds.

Parameters

Vi	The SessionID handle that you obtain from 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()

```
EXPORT int
PwrSnsr_GetTimePer
Point                (                SessionID                Vi,
                                const char *                Channel,
                                float *                TimePerPoint
                )
```

Get time spacing for each waveform point in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
TimePerPoint	Time spacing for each waveform point in seconds.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTimespan()

```
EXPORT int
PwrSnsr_GetTimespan (
                                SessionID      Vi,
                                float *        Timespan
)
```

Get the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division.

Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timespan

Returns

Success (0) or error code.

◆ PwrSnsr_GetTraceStartTime()

```
EXPORT int
PwrSnsr_GetTraceStartTime (
                                SessionID      Vi,
                                const char *    Channel,
                                float *        TraceStartTime
)
```

Get time offset (start time) of the trace in seconds. May be negative, indicating pre-trigger information.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TraceStartTime

Time offset (start time) of the trace in seconds. May be negative, indicating pre-trigger information.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigDelay()

```
EXPORT int
PwrSnsr_GetTrigDelay
(
                                SessionID      Vi,
                                float *        Delay
)
```

Return the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position.

Positive values cause the actual trigger to occur after the trigger condition is met. This places the trigger event to the left of the trigger point on the display, and is useful for viewing events during a pulse, some fixed delay time after the rising edge trigger. Negative trigger delay places the trigger event to the right of the trigger point on the display, and is useful for looking at events before the trigger edge.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Delay

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigHoldoff()

```
EXPORT int
PwrSnsr_GetTrigHoldoff
(
                                SessionID      Vi,
                                float *        Holdoff
)
```

Return the trigger holdoff time in seconds.

Trigger holdoff is used to disable the trigger for a specified amount of time after each trigger event. The holdoff time starts immediately after each valid trigger edge, and will not permit any new triggers until the time has expired. When the holdoff time is up, the trigger re-arms, and the next valid trigger event (edge) will cause a new sweep. This feature is used to help synchronize the power meter with burst waveforms such as a TDMA or GSM frame. The

trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Holdoff

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigHoldoffMode()

```
EXPORT int
PwrSnsr_GetTrigHoldoffMode (
                                SessionID      Vi,
                                PwrSnsrHoldoffModeEnum* HoldoffMode
)
```

Returns the holdoff mode to normal or gap holdoff.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

HoldoffMode

holdoff mode.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigLevel()

```
EXPORT int
PwrSnsr_GetTrigLevel (
                                SessionID      Vi,
                                float*         Level
)
```

Return the trigger level for synchronizing data acquisition with a pulsed input signal.

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The

trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Level

Trigger level in dBm.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigMode()

```
EXPORT int
PwrSnsr_GetTrigMod
e                (                SessionID                Vi,
                                PwrSnsrTriggerMod
                                eEnum *                Mode
                                )
```

Return the trigger mode for synchronizing data acquisition with pulsed signals.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Mode

Trigger mode.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigPosition()

```
EXPORT int
PwrSnsr_GetTrigPosi
tion                (                SessionID                Vi,
                                PwrSnsrTriggerPosi
                                tionEnum *                Position
                                )
```

Return the position of the trigger event on displayed sweep.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session. Trigger position.

Position**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigSlope()

```
EXPORT int
PwrSnsr_GetTrigSlop
e                (                SessionID                Vi,
                                PwrSnsrTriggerSlop
                                eEnum *                Slope
                                )
```

Return the trigger slope or polarity.

When set to positive, trigger events will be generated when a signal rising edge crosses the trigger level threshold. When negative, trigger events are generated on the falling edge of the pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Slope**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigSource()

```
EXPORT int
PwrSnsr_GetTrigSour
ce                (                SessionID                Vi,
                                PwrSnsrTriggerSou
                                rceEnum *                Source
                                )
```

Set the signal the power meter monitors for a trigger. It can be channel external input, or independent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Source**Returns**

Success (0) or error code.

◆ PwrSnsr_GetTrigStatus()

```
EXPORT int
PwrSnsr_GetTrigStat
us          (          SessionID          Vi,
              PwrSnsrTriggerStat
              usEnum *          Status
              )
```

The status of the triggering system. Update rate is controlled by FetchLatency setting.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Status

Status of the trigger.

Returns

Success (0) or error code.

◆ PwrSnsr_GetTrigVernier()

```
EXPORT int
PwrSnsr_GetTrigVern
ier          (          SessionID          Vi,
                    float *          Vernier
                    )
```

Return the fine position of the trigger event on the power sweep.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Vernier

Trigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code.

◆ PwrSnsr_GetUnits()

```
EXPORT int  
PwrSnsr_GetUnits    (          SessionID          Vi,  
                        const char *          Channel,  
                        PwrSnsrUnitsEnum          Units  
                        *)  
)
```

Get units for the selected channel.

Voltage is calculated with reference to the sensor input impedance. Note that for ratiometric results, logarithmic units will always return dBr (dB relative) while linear units return percent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Units

Returns

Success (0) or error code.

◆ PwrSnsr_GetVerticalCenter()

```
int EXPORT  
PwrSnsr_GetVertical  
Center    (          SessionID          Vi,  
                        const char *          Channel,  
                        float *          VerticalCenter  
                        *)  
)
```

Gets vertical center based on current units: <arg> = (range varies by units)

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical center in units

Returns

Success (0) or error code.

◆ **PwrSnsr_GetVerticalScale()**

```
int EXPORT
PwrSnsr_GetVertical
Scale          (          SessionID          Vi,
                  const char *          Channel,
                  float *          VerticalScale
                  )
```

Gets vertical scale based on current units: <arg> = (range varies by units)

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical scale in units

Returns

Success (0) or error code.

◆ **PwrSnsr_GetWriteProtection()**

```
EXPORT int
PwrSnsr_GetWritePr
otection        (          SessionID          Vi,
                  int *          WriteProtection
                  )
```

Get whether the measurement buffer is set to overwrite members that have not been read by the user.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

WriteProtection

Returns true if the measurement buffer is allowed to overwrite members that have not been read by the user.

Returns

Success (0) or error code.

◆ PwrSnsr_init()

```
EXPORT int  
PwrSnsr_init ( char * ResourceName,  
               SessionID * Vi  
               )
```

Initialize a communication session with a supported USB power sensor.

Parameters**ResourceName**

Name of the resource. The resource descriptor is in the following format:

USB::[VID]::[PID]::[Serial Number]::BTN

Where serial number is the USB power meter's serial number in decimal format, and the VID and PID are in hexadecimal format.

e.g. 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 more than one descriptor separated by a semicolon.

Channel assignment is determined by the order in the list, in other words CH1 would be the first listed resource, CH2 the second resource, etc.

e.g. Define a synthetic peak power meter using serial number 1234 for CH1 and serial number 4242 for CH2:

USB::0x1BFE::0x5500::1234::BTN;USB::0x1BFE::0x5500::4242::BTN

Returns

Success (0) or error code.

◆ PwrSnsr_InitiateAquisition()

```
EXPORT int  
PwrSnsr_InitiateAquisition ( SessionID Vi  
                             )
```

Starts a single measurement cycle when INITiate:CONTinuous is set to OFF.

In Modulated Mode, the measurement will complete once the power has been integrated for the full FILTER time. In Pulse Mode, enough trace sweeps must be triggered to satisfy the AVERaging setting. In Statistical Mode, acquisition stops once the terminal condition(s) are met. In each case, no reading will be returned until the measurement is complete. This command is not valid when INITiate:CONTinuous is ON, however, by convention this situation does not result in a SCPI error

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_IsLicenseDongleConnected()

```
EXPORT int
PwrSnsr_IsLicenseDongleConnected (int * val)
```

Get whether the hardware license dongle is connected.

Parameters

val

Boolean. 1 for connected or 0 for not connected.

Returns

Success (0) or error code.

◆ PwrSnsr_LoadMemChanFromArchive()

```
EXPORT int
PwrSnsr_LoadMemChanFromArchive (SessionID
                                const char *
                                const char *
                                Vi,
                                memChan,
                                ArchiveContent)
```

Loads the named memory channel using the given archive. If the memory channel does not exist, one is created.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MemChan

Memory channel name. Must have the form MEM1...n, where n is the number of measurement channels. In single channel configurations, this parameter should always be "MEM1".

ArchiveContent

An xml document containing settings and readings obtained using the SaveToMemoryChannel method. An archive can be obtained using the GetMemChanArchive method.

Returns

Success (0) or error code.

◆ PwrSnsr_MeasurePower()

```
EXPORT int
PwrSnsr_MeasurePower (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return average power using a default instrument configuration in Modulated Mode and dBm units. Instrument remains stopped in Modulated Mode after a measurement.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.

Val

Average power in dBm

Returns

Success (0) or error code.

◆ PwrSnsr_MeasureVoltage()

```
EXPORT int
PwrSnsr_MeasureVol
tage          (          SessionID          Vi,
                  const char *          Channel,
                  PwrSnsrCondCode          CondCode,
                  Enum *          Val
                  float *
          )
```

Return average voltage using a default instrument configuration in Modulated Mode and volts units. Instrument remains stopped in Modulated Mode after a measurement.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement.
Val	Average voltage in linear volts.

Returns

Success (0) or error code.

◆ PwrSnsr_QueryAverageMeasurements()

```
EXPORT int
PwrSnsr_QueryAvera
geMeasurements    (          SessionID          Vi,
                  const char *          Channel,
                  int          ValBufferSize,
                  float          Val[],
                  int *          ValActualSize
          )
```

Query the power meter for all buffered average power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
-----------	---

Channel

Channel number. For single instruments, set this to "CH1".

ValBufferSize

Size of the buffer in elements.

Val

Array of average power measurements.

ValActualSize

Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_QueryDurations()

```

EXPORT int
PwrSnsr_QueryDurations (
    SessionID          Vi,
    const char *       Channel,
    int                 ValBufferSize,
    float               Val[],
    int *               ValActualSize
)

```

Query the power meter for all buffered measurement durations in seconds.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, 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()

```

EXPORT int
PwrSnsr_QueryMaxMeasurements (
    SessionID          Vi,
    const char *       Channel,
    int                 ValBufferSize,
    float               Val[]
)

```

```

                                int *          ValActualSize
                                )

```

Query the power meter for all buffered maximum power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of max measurements.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_QueryMinMeasurements()

```

EXPORT int
PwrSnsr_QueryMinM
easurements      (
                                SessionID      Vi,
                                const char *    Channel,
                                int             ValBufferSize,
                                float           Val[],
                                int *          ValActualSize
                                )

```

Query the power meter for all buffered minimum power measurements.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of min measurements.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_QuerySequenceNumbers()

```
EXPORT int
PwrSnsr_QuerySequenceNumbers (
    SessionID          Vi,
    const char *       Channel,
    int                ValBufferSize,
    long long          Val[],
    int *              ValActualSize
)
```

Query the power meter for all buffered sequence numbers.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
ValBufferSize	Size of the buffer.
Val	Array of sequence numbers.
ValActualSize	Actual size of the returned array in elements.

Returns

Success (0) or error code.

◆ PwrSnsr_QueryStartTimes()

```
EXPORT int
PwrSnsr_QueryStartTimes (
    SessionID          Vi,
    const char *       Channel,
    int                ValBufferSize,
    float              Val[],
    int *              ValActualSize
)
```

Query the power meter for all buffered start times in seconds.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
-----------	---

Channel	Channel number. For single instruments, set this to "CH1".
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()

```
EXPORT int
PwrSnsr_ReadByteAr
ray          (
                SessionID      Vi,
                const char *    Channel,
                int              Count,
                int              ValBufferSize,
                unsigned char    Val[],
                int *             ValActualSize
            )
```

Reads byte array from the meter.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Count	Maximum count of bytes to return.

ValBufferSize

Size of the buffer.

Val

Byte array from the USB.

ValActualSize

Actual size of the returned array in bytes.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadControl()**

```

EXPORT int
PwrSnsr_ReadControl(
    SessionID          Vi,
    const char *       Channel,
    int                Count,
    int                ValBufferSize,
    unsigned char *    Val[],
    int *              ValActualSize
)

```

Reads a control transfer on the USB.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Count

Maximum count to return.

ValBufferSize

Size of the buffer.

Val

Byte array from a USB control transfer.

ValActualSize

Actual size of the returned array in bytes.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadCWArray()**

```

EXPORT int
PwrSnsr_ReadCWArray(
    SessionID          Vi,
    const char *       Channel,
    float *            PeakAverage,

```

PwrSnsrCondCode

Enum * PeakAverageValid,
float * PeakMax,

PwrSnsrCondCode

Enum * PeakMaxValid,
float * PeakMin,

PwrSnsrCondCode

Enum * PeakMinValid,
float * PeakToAvgRatio,

PwrSnsrCondCode

Enum * PeakToAvgRatioValid

)

Returns the current average, maximum, minimum powers or voltages and the peak-to-average ratio of the specified channel. Units are the same as the channel's units. Note the peak-to-average ratio and marker ratio are returned in dB for logarithmic channel units, and percent for all other channel units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

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()

```
EXPORT int
PwrSnsr_ReadCWPower
(
```

```
SessionID
const char *
```

```
Vi,
Channel,
```

PwrSnsrCondCodeEnum *

float *

IsValid,

Val

)

Initiates a CW power acquisition and returns the result in channel units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

IsValid

Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadDutyCycle()

EXPORT int

PwrSnsr_ReadDutyC

ycle

(

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

CondCode,

Val

)

Returns the ratio of the pulse on-time to off-time.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadEdgeDelay()

```
EXPORT int
PwrSnsr_ReadEdgeDelay (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *             Val
)
```

Returns time offset from the trigger reference to the first mesial transition level of either slope on the waveform.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadFallTime()

```
EXPORT int
PwrSnsr_ReadFallTime (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *             Val
)
```

Returns the interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle
-----------	---

Channel	identifies a particular instrument session. 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()

```
EXPORT int
PwrSnsr_ReadIEEEBottom (
    SessionID          Vi,
    const char *        Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *              Val
)
```

Returns the IEEE-define base line, i.e. The two portions of a pulse waveform which represent the first nominal state from which a pulse departs and to which it ultimately returns.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIEEETop()

```
EXPORT int
PwrSnsr_ReadIEEETop (
    SessionID          Vi,
    const char *        Channel,
```


PwrSnsrCondCodeEnum *

float *

CondCode,

Val

)

Returns the IEEE-defined top line, i.e. the portion of a pulse waveform which represents the second nominal state of a pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalAvg()

EXPORT int

PwrSnsr_ReadInterva

lAvg

(

SessionID

Vi,

const char *

Channel,

PwrSnsrCondCodeEnum *

CondCode,

float *

Val

)

Return the average power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalFilteredMax\(\)](#)

```
EXPORT int
PwrSnsr_ReadIntervalFilteredMax (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return the maximum filtered power or voltage in the time interval between marker 1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalFilteredMin\(\)](#)

```
EXPORT int
PwrSnsr_ReadIntervalFilteredMin (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

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.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.
Returns	
	Success (0) or error code.

◆ PwrSnsr_ReadIntervalMax()

```
EXPORT int
PwrSnsr_ReadIntervalMax
(
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Return the maximum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
CondCode	Condition code for the measurement. Condition code.
Val	Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadIntervalMaxAvg()

```
EXPORT int
PwrSnsr_ReadIntervalMaxAvg
(
    SessionID          Vi,
```

const char *	Channel,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	CondCode,
float *	Val

)

Return maximum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadIntervalMin\(\)](#)

```
EXPORT int
PwrSnsr_ReadInterva
IMin      (
```

SessionID	Vi,
const char *	Channel,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	CondCode,
float *	Val

)

Return the minimum instantaneous power or voltage in the time interval between marker1 and marker 2. The units will be the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadIntervalMinAvg()**

```
EXPORT int
PwrSnsr_ReadIntervalMinAvg (
    SessionID Vi,
    const char * Channel,
    PwrSnsrCondCode
Enum * CondCode,
    float * Val
)
```

Return minimum of the average power trace between MK1 and MK2. The units will be the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadIntervalPkToAvg()**

```
EXPORT int
PwrSnsr_ReadIntervalPkToAvg (
    SessionID Vi,
    const char * Channel,
    PwrSnsrCondCode
Enum * CondCode,
    float * Val
)
```

Return the peak-to-average ratio of the power or voltage between marker 1 and marker 2. The units are dB for logarithmic channel units or percent for linear channel units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadMarkerAverage()**

```
EXPORT int
PwrSnsr_ReadMarker
Average          (          SessionID          Vi,
                      const char *          Channel,
                      int          Marker,
                      PwrSnsrCondCode
Enum *          CondCode,
                      float *          Val
                      )
```

For the specified marker, return the average power or voltage at the marker. The units are the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Marker

Marker number.

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerDelta\(\)](#)

```
EXPORT int
PwrSnsr_ReadMarker
Delta          (          SessionID          Vi,
                   const char *          Channel,
                   PwrSnsrCondCode
Enum *          CondCode,
                   float *          Val
                   )
```

Return the difference between MK1 and MK2. The units will be the same as marker units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerMax\(\)](#)

```
EXPORT int
PwrSnsr_ReadMarker
Max          (          SessionID          Vi,
                   const char *          Channel,
                   int          Marker,
                   PwrSnsrCondCode
Enum *          CondCode,
                   float *          Val
                   )
```

For the specified marker, return the maximum power or voltage at the marker. The units are the same as the specified channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

Channel number. For single instruments, set this to "CH1".

Marker

Marker number.

CondCodeCondition code for the measurement.
Condition code.**Val**

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerMin\(\)](#)

```

EXPORT int
PwrSnsr_ReadMarker
Min          (          SessionID          Vi,
               const char *          Channel,
               int          Marker,
               PwrSnsrCondCode
               Enum *          CondCode,
               float *          Val
               )

```

For the specified marker, return the minimum power or voltage at the marker. The units are the same as the specified channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Marker

Marker number.

CondCodeCondition code for the measurement.
Condition code.**Val**

measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadMarkerRatio\(\)](#)


```

EXPORT int
PwrSnsr_ReadMarker
Ratio          (
                SessionID          Vi,
                const char *       Channel,
                PwrSnsrCondCode
Enum *        CondCode,
                float *            Val
                )

```

Return the ratio of MK1 to MK2. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerRDelta()

```

EXPORT int
PwrSnsr_ReadMarker
RDelta        (
                SessionID          Vi,
                const char *       Channel,
                PwrSnsrCondCode
Enum *        CondCode,
                float *            Val
                )

```

Return the difference between MK2 and MK1. The units will be the same as marker units.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadMarkerRRatio()

```
EXPORT int
PwrSnsr_ReadMarker
RRatio          (          SessionID          Vi,
                  const char *          Channel,
                  PwrSnsrCondCode          CondCode,
                  Enum *          Val
                  float *
                  )
```

Return the ratio of MK2 to MK1. The units will be dB for logarithmic units or percent for linear units.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel Channel number. For single instruments, set this to "CH1".

CondCode Condition code for the measurement. Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadOfftime()

```
EXPORT int
PwrSnsr_ReadOfftim
e          (          SessionID          Vi,
                  const char *          Channel,
                  PwrSnsrCondCode          CondCode,
                  Enum *          Val
                  float *
                  )
```

Returns the time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadOvershoot()**

```
EXPORT int
PwrSnsr_ReadOvershoot (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Returns the difference between the distortion following a major transition and the IEEE top line in dB or percent, depending on the channel units.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadPeriod()**

```
EXPORT int
PwrSnsr_ReadPeriod (
    SessionID          Vi,
```

```

                                const char *      Channel,
                                PwrSnsrCondCode
                                Enum *          CondCode,
                                float *        Val
                                )

```

Returns the interval between two successive pulses.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel Channel number. For single instruments, set this to "CH1".

CondCode Condition code for the measurement. Condition code.

Val Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadPowerArray\(\)](#)

```

EXPORT int
PwrSnsr_ReadPower
Array          (
                                SessionID          Vi,
                                const char *      Channel,
                                float *          PulsePeak,
                                PwrSnsrCondCode
                                Enum *          PulsePeakValid,
                                float *          PulseCycleAvg,
                                PwrSnsrCondCode
                                Enum *          PulseCycleAvgValid,
                                float *          PulseOnAvg,
                                PwrSnsrCondCode
                                Enum *          PulseOnValid,
                                float *          IEEETop,
                                PwrSnsrCondCode
                                Enum *          IEEETopValid,
                                float *          IEEEBottom,
                                PwrSnsrCondCode
                                Enum *          IEEEBottomValid,
                                float *          Overshoot,

```

PwrSnsrCondCodeEnum *

float *

OvershootValid,

Droop,

PwrSnsrCondCodeEnum *

DroopValid

)

Returns an array of the current automatic amplitude measurements performed on a periodic pulse waveform.

Measurements performed are: peak amplitude during the pulse, average amplitude over a full cycle of the

pulse waveform, average amplitude during the pulse, IEEE top amplitude, IEEE bottom amplitude, and overshoot.

Units are the same as the channel's units. Note the pulse overshoot is returned in dB for logarithmic channel units,

and percent for all other units. Also, the pulse ON interval used for peak and average calculations is

defined by the SENSE:PULSE:STARTGT and :ENDGT time gating settings.

A full pulse (rise and fall) must be visible on the display to make average and peak pulse power measurements,

and a full cycle of the waveform must be visible to calculate average cycle amplitude.

Parameters**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
Overshoot

Condition code.

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()

```
EXPORT int
PwrSnsr_ReadPRF (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCode   CondCode,
    Enum *             Val,
    float *
)
```

Returns the number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadPulseCycleAvg()

```
EXPORT int
PwrSnsr_ReadPulse
CycleAvg (
    SessionID          Vi,
    const char *       Channel,
```

PwrSnsrCondCodeEnum *

float *

CondCode,

Val

)

Returns the average power of the entire pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadPulseOnAverage()

EXPORT int

PwrSnsr_ReadPulse

OnAverage

(

SessionID

const char *

PwrSnsrCondCodeEnum *

float *

Vi,

Channel,

CondCode,

Val

)

Average power of the ON portion of the pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadPulsePeak\(\)](#)

```
EXPORT int
PwrSnsr_ReadPulsePeak (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Returns the peak amplitude during the pulse.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ [PwrSnsr_ReadRiseTime\(\)](#)

```
EXPORT int
PwrSnsr_ReadRiseTime (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCodeEnum * CondCode,
    float *            Val
)
```

Returns the interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement.
Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadSCPI()

```
EXPORT int
PwrSnsr_ReadSCPI (          SessionID      Vi,
                           int              ValueBufferSize,
                           long *          ValueActualSize,
                           char            Value[],
                           int              Timeout
                           )
```

Read a SCPI string response from the instrument.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ValueBufferSize

Number of elements in Value.

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()

```
EXPORT int
PwrSnsr_ReadSCPIB
ytes          (          SessionID      Vi,
                           int              ValueBufferSize,
                           char            Value[],
                           long *          ValueActualSize,
```

```

                                int                               Timeout
                                )

```

Read a SCPI byte array response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
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()

```

EXPORT int
PwrSnsr_ReadSCPIFromNamedParser (
                                SessionID           Vi,
                                const char *         name,
                                int                   ValueBufferSize,
                                long *               ValueActualSize,
                                char                  Value[],
                                int                   Timeout
                                )

```

Read a SCPI string response from the instrument.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
name	Name of the parser. If parser doesn't exist, returns PWR_SNSR_ERROR_NULL_POINTER. PwrSnsr_SendSCPIToNamedParser can be used to create a named parser.

ValueBufferSize

ValueActualSize

Value

Timeout

Returns

Success (0) or error code.

Number of elements in Value.

Number of elements actually written to Value.

The string returned from the instrument SCPI interface.

Time out in milliseconds for the read operation. Use -1 for infinite and -2 to use the existing time out value.

◆ PwrSnsr_ReadTimeArray()

```
EXPORT int
PwrSnsr_ReadTimeA
rray (
```

SessionID	Vi,
const char *	Channel,
float *	Frequency,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	FrequencyValid,
float *	Period,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	PeriodValid,
float *	Width,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	WidthValid,
float *	Offtime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	OfftimeValid,
float *	DutyCycle,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	DutyCycleValid,
float *	Risetime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	RisetimeValid,
float *	Falltime,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	FalltimeValid,
float *	EdgeDelay,
<u>PwrSnsrCondCode</u>	
<u>Enum</u> *	EdgeDelayValid,
float *	Skew,

PwrSnsrCondCode
Enum*

SkewValid

)

Returns an array of the current automatic timing measurements performed on a periodic pulse waveform.

Measurements performed are: the frequency, period, width, offtime and duty cycle of the pulse waveform, and the risetime and falltime of the edge transitions. For each of the measurements to be performed, the appropriate items to be measured must within the trace window. Pulse frequency, period, offtime and duty cycle measurements require that an entire cycle of the pulse waveform (minimum of three edge transitions) be present. Pulse width measurement requires that at least one full pulse is visible, and is most accurate if the pulse width is at least 0.4 divisions. Risetime and falltime measurements require that the edge being measured is visible, and will be most accurate if the transition takes at least 0.1 divisions. It is always best to have the power meter set on the fastest timebase possible that meets the edge visibility restrictions. Set the trace averaging as high as practical to reduce fluctuations and noise in the pulse timing measurements. Note that the timing of the edge transitions is defined by the settings of the SENSE:PULSE:DISTal, :MESIal and :PROXimal settings; see the descriptions For those commands. Units are the same as the channel's units.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
Frequency	The number of cycles of a repetitive signal that take place in one second (Pulse Repetition Frequency).
FrequencyValid	Condition code.
Period	The interval between two successive pulses.
PeriodValid	Condition code.
Width	The interval between the first and second signal crossings of the mesial line.
WidthValid	Condition code.
Offtime	The time a repetitive pulse is off. (Equal to the pulse period minus the pulse width).
OfftimeValid	Condition code.

DutyCycle

The ratio of the pulse on-time to period.

DutyCycleValid

Condition code.

Risetime

The interval between the first signal crossing of the proximal line to the first signal crossing of the distal line.

RisetimeValid

Condition code.

Falltime

The interval between the last signal crossing of the distal line to the last signal crossing of the proximal line.

FalltimeValid

Condition code.

EdgeDelay

Time offset from the trigger reference to the first mesial transition level of either slope on the waveform.

EdgeDelayValid

Condition code.

Skew

The trigger offset between the assigned trigger channel and this channel.

SkewValid

Condition code.

Returns

Success (0) or error code.

◆ PwrSnsr_ReadWaveform()

```

EXPORT int
PwrSnsr_ReadWaveform(
    SessionID          Vi,
    const char *       Channel,
    WaveformArrayBuffer Size,
    WaveformArray[]    WaveformArray,
    WaveformArrayActualSize
    int *
)

```

Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the waveform for this channel. Call FetchWaveform to obtain the waveforms for other channels.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

WaveformArrayBufferSize

Size in bytes of the Waveform buffer.

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_ReadWaveformMinMax()

```
EXPORT int
PwrSnsr_ReadWaveformMinMax (
    SessionID          Vi,
    const char *       Channel,
    int                 MinWaveformBufferSize,
    float               MinWaveform[],
    int *               MinWaveformActualSize,
    int                 MaxWaveformBufferSize,
    float               MaxWaveform[],
    int *               MaxWaveformActualSize,
    WaveformArrayBufferSize,
    WaveformArray[],
    WaveformArrayActualSize,
    int *
)
```

Initiates an acquisition on all enabled channels, waits (up to MaxTime) for the acquisition to complete, and returns the min/max waveforms for this channel. Call FetchMinMaxWaveform to obtain the min/max waveforms for other channels.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

MinWaveformBufferSize

Size in bytes of the MinWaveform buffer.

MinWaveform

This array contains the min waveform. Units for the individual array elements are in the channel units setting.

MinWaveformActualSize

Size in bytes of the data written to MinWaveform.

MaxWaveformBufferSize

Size in bytes of the MaxWaveform buffer.

MaxWaveform

This array contains the max waveform. Units for the individual array elements are in the channel units setting.

MaxWaveformActualSize

Size in bytes of the data written to MaxWaveform.

WaveformArrayBufferSize

Size in bytes of the Waveform buffer.

WaveformArray

The array contains the average waveform. Units for the individual array elements are in the channel units setting.

WaveformArrayActualSize

Size in bytes of the data written to WaveformArray.

Returns

Success (0) or error code.

◆ **PwrSnsr_ReadWidth()**

```
EXPORT int
PwrSnsr_ReadWidth (
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrCondCode    CondCode,
    Enum *              Val,
    float *
)
```

Returns the pulse width, i.e. the interval between the first and second signal crossings of the mesial line.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CondCode

Condition code for the measurement. Condition code.

Val

Measurement value.

Returns

Success (0) or error code.

◆ PwrSnsr_reset()

```
EXPORT int
PwrSnsr_reset (          SessionID      Vi          )
```

Places the instrument in a known state.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_ResetContinuousCapture()

```
EXPORT int
PwrSnsr_ResetContinuous
Capture (          SessionID      Vi          )
```

Sets a flag indicating to restart continuous capture. This method allows the user to restart continuous acquisition. Has no effect if ContinuousCapture is set to false.

Parameters

Vi The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_SaveToMemoryChannel()

```
EXPORT int
PwrSnsr_SaveToMemoryChannel (          SessionID      Vi,
                                     const char *      memChan,
                                     const char *      ChannelName
                                     )
```


Saves the given channel to a memory channel. If the memory channel does not exist, a new one is created.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

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()

```
EXPORT int
PwrSnsr_SaveUserCal (
                                SessionID      Vi,
                                const char *    Channel
)
```

Instructs power meter to save the value of fixed cal, zero, and skew values.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_self_test()

```
EXPORT int
PwrSnsr_self_test (
                                SessionID      Vi,
                                int *          TestResult
)
```

Performs an instrument self test, waits for the instrument to complete the test, and queries the instrument for the results. If the instrument passes the test, `TestResult` is 0.

Parameters

Vi

The `SessionID` handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.
Error or success code.

TestResult

Returns

Success (0) or error code.

◆ `PwrSnsr_SendSCPIBytes()`

```
EXPORT int
PwrSnsr_SendSCPIB
ytes          (
```

```
SessionID
int
char
```

```
Vi,
CommandBufferSize,
Command[]
```

```
)
```

Send a SCPI command as a byte array.

Parameters

Vi

The `SessionID` handle that you obtain from the `PwrSnsr_init` function. The handle identifies a particular instrument session.
Number of elements in `Command`.
Command to send.

CommandBufferSize

Command

Returns

Success (0) or error code.

◆ `PwrSnsr_SendSCPICommand()`

```
EXPORT int
PwrSnsr_SendSCPIC
ommand        (
```

```
SessionID
const char *
```

```
Vi,
Command
```

```
)
```

Send a SCPI command to the instrument.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Command**Returns**

Success (0) or error code.

◆ PwrSnsr_SendSCPIToNamedParser()

```
EXPORT int
PwrSnsr_SendSCPIToNamedParser (          SessionID      Vi,
                                         const char *    name,
                                         const char *    Command
                                         )
```

Send a SCPI command to the instrument using a named SCPI parser.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

name

Name of the parser. Creates a new parser if the name is not already used.

Command**Returns**

Success (0) or error code.

◆ PwrSnsr_SetAverage()

```
EXPORT int
PwrSnsr_SetAverage (          SessionID      Vi,
                              const char *    Channel,
                              int             Average
                              )
```

Set the number of traces averaged together to form the measurement result on the selected channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Average**Returns**

Success (0) or error code.

◆ PwrSnsr_SetBandwidth()

```
EXPORT int
PwrSnsr_SetBandwidth(
    SessionID          Vi,
    const char *       Channel,
    PwrSnsrBandwidthEnum Bandwidth
)
```

Set the sensor video bandwidth for the selected sensor.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Bandwidth**Returns**

Success (0) or error code.

◆ PwrSnsr_SetCalFactor()

```
EXPORT int
PwrSnsr_SetCalFactor(
    SessionID          Vi,
    const char *       Channel,
    float              CalFactor
)
```

Set the frequency calibration factor currently in use on the selected channel.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

CalFactor**Returns**

Success (0) or error code.

◆ PwrSnsr_SetCapture()

```
EXPORT int
PwrSnsr_SetCapture (          SessionID      Vi,
                             const char *    Channel,
                             int             Capture
                             )
```

Set whether statistical capture is enabled.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Capture**Returns**

Success (0) or error code.

◆ PwrSnsr_SetCCDFTraceCount()

```
EXPORT int
PwrSnsr_SetCCDFTraceCount (          SessionID      Vi,
                                     const char *    Channel,
                                     int             TraceCount
                                     )
```

Set the number of points (1 - 16384) in the CCDF trace plot.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TraceCount**Returns**

Success (0) or error code.

◆ PwrSnsr_SetContinuousCapture()

```
EXPORT int
PwrSnsr_SetContinuousCapture (          SessionID      Vi,
                                     int                  ContinuousCapture
                                     )
```

Set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ContinuousCapture

True to set whether AcquireMeasurements will stop the measurement buffer session or continue capturing measurement buffer entries after being called.

Returns

Success (0) or error code.

◆ PwrSnsr_SetDistal()

```
EXPORT int
PwrSnsr_SetDistal (          SessionID      Vi,
                             const char *   Channel,
                             float          Distal
                             )
```

Set the pulse amplitude percentage, which is used to define the end of a rising edge or beginning of a falling edge transition.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Distal**Returns**

Success (0) or error code.

◆ PwrSnsr_SetDuration()

```
EXPORT int
PwrSnsr_SetDuration (          SessionID      Vi,
                               float           Duration
                               )
```

Set the duration samples are captured during each timed mode acquisition.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Duration

The duration samples are captured during each timed mode acquisition in seconds.

Returns

Success (0) or error code.

◆ PwrSnsr_SetEnabled()

```
EXPORT int
PwrSnsr_SetEnabled (          SessionID      Vi,
                               const char *   Channel,
                               int            Enabled
                               )
```

Get the measurement state of the selected channel. When the value is true, the channel performs measurements; when the value is false, the channel is disabled and no measurements are performed.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Enabled

Boolean. 1 for enable; 0 for disable.

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndDelay()

```
EXPORT int
PwrSnsr_SetEndDelay (
                                SessionID      Vi,
                                float           EndDelay
)
```

Set delay time added to the detected end of a burst for analysis. Typically negative. Typically used to exclude the falling edge of a burst.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
EndDelay	Delay time added to the detected end of a burst for analysis.

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndGate()

```
EXPORT int
PwrSnsr_SetEndGate (
                                SessionID      Vi,
                                const char *    Channel,
                                float           EndGate
)
```

Set the point on a pulse, which is used to define the end of the pulse's active interval.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EndGate	

Returns

Success (0) or error code.

◆ PwrSnsr_SetEndQual()

```
EXPORT int
PwrSnsr_SetEndQual (                SessionID      Vi,
                                   float              EndQual
                                   )
```

Set the minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

EndQual

The minimum amount of time power remains below the trigger point to be counted as the end of a burst.

Returns

Success (0) or error code.

◆ PwrSnsr_SetExternalSkew()

```
EXPORT int
PwrSnsr_SetExternal
Skew                (                SessionID      Vi,
                                   const char *      Channel,
                                   float              External
                                   )
```

Sets the skew in seconds for the external trigger.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

External

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ [PwrSnsr_SetFetchLatency\(\)](#)

```
EXPORT int
PwrSnsr_SetFetchLatency (          SessionID      Vi,
                                int                Latency
                                )
```

Set the period the library waits to update fetch measurements in ms.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Latency

Fetch latency in ms.

Returns

Success (0) or error code.

◆ [PwrSnsr_SetFilterState\(\)](#)

```
EXPORT int
PwrSnsr_SetFilterState (          SessionID      Vi,
                                const char *    Channel,
                                PwrSnsrFilterStateEnum FilterState
                                )
```

Set the current setting of the integration filter on the selected channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

FilterState

Returns

Success (0) or error code.

◆ [PwrSnsr_SetFilterTime\(\)](#)

```
EXPORT int
PwrSnsr_SetFilterTime
e (                               SessionID           Vi,
                                const char *         Channel,
                                float                 FilterTime
                                )
```

Set the current length of the integration filter on the selected channel.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

FilterTime

Returns

Success (0) or error code.

◆ PwrSnsr_SetFrequency()

```
EXPORT int
PwrSnsr_SetFrequency
cy (                               SessionID           Vi,
                                const char *         Channel,
                                float                 Frequency
                                )
```

Set the RF frequency for the current sensor, and apply the appropriate frequency calibration factor from the sensor internal table.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Frequency

RF Frequency in Hz.

Returns

Success (0) or error code.

◆ PwrSnsr_SetGateMode()

```
EXPORT int
PwrSnsr_SetGateMode (
    SessionID Vi,
    PwrSnsrMeasBufferGateEnum GateMode
)
```

Each Measurement Buffer Entry is controlled by a buffer gate that defines the start and end of the entry time interval.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

GateMode

Buffer gate mode that defines the start and end of the entry time interval.

Returns

Success (0) or error code.

◆ PwrSnsr_SetGating()

```
EXPORT int
PwrSnsr_SetGating (
    SessionID Vi,
    const char * Channel,
    PwrSnsrStatGatingEnum Gating
)
```

Set whether the statical capture is gated by markers or free-running.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Gating

Returns

Success (0) or error code.

◆ PwrSnsr_SetHorizontalOffset()

```
EXPORT int
PwrSnsr_SetHorizontalOffset (
    SessionID
    const char *
    double
    Vi,
    Channel,
    HorizontalOffset
)
```

Set the statistical mode horizontal scale offset in dB. The offset value will appear at the leftmost edge of the scale with units dBr (decibels relative).

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

HorizontalOffset

Returns

Success (0) or error code.

◆ PwrSnsr_SetHorizontalScale()

```
EXPORT int
PwrSnsr_SetHorizontalScale (
    SessionID
    const char *
    double
    Vi,
    Channel,
    HorizontalScale
)
```

Set the statistical mode horizontal scale in dB/Div.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

HorizontalScale

Returns

Success (0) or error code.

◆ PwrSnsr_SetInitiateContinuous()

```
EXPORT int
PwrSnsr_SetInitiateC
ontinuous          (          SessionID          Vi,
                    int          InitiateContinuous
                    )
```

Set the data acquisition mode for single or free-run measurements.

If INITiate:CONTinuous is set to ON, the instrument immediately begins taking measurements (Modulated, CW and Statistical Modes), or arms its trigger and takes a measurement each time a trigger occurs (Pulse Mode). If set to OFF, the measurement will begin (or be armed) as soon as the INITiate command is issued, and will stop once the measurement criteria (averaging, filtering or sample count) has been satisfied. Note that INITiate:IMMediate and READ commands are invalid when INITiate:CONTinuous is set to ON; however, by convention this situation does not result in a SCPI error.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

InitiateContinuous

Boolean. 0 for off or 1 for on.

Returns

Success (0) or error code.

◆ PwrSnsr_SetInternalSkew()

```
EXPORT int
PwrSnsr_SetInternalS
kew          (          SessionID          Vi,
                  const char *          Channel,
                  float          InternalSkew
                  )
```

Sets the skew in seconds for the internal trigger.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

InternalSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_SetMarkerPixelPosition()

```
EXPORT int
PwrSnsr_SetMarkerPixelPosition (
                                SessionID      Vi,
                                int             MarkerNumber,
                                int             PixelPosition
)
```

Set the horizontal pixel position (X-axis-position) of the selected vertical marker. There are 501 pixel positions numbered from 0 to 500 inclusive.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MarkerNumber**PixelPosition****Returns**

Success (0) or error code.

◆ PwrSnsr_SetMarkerTimePosition()

```
EXPORT int
PwrSnsr_SetMarkerTimePosition (
                                SessionID      Vi,
                                int             MarkerNumber,
                                float           TimePosition
)
```

Set the time (x-axis-position) of the selected marker relative to the trigger.

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.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

identifies a particular instrument session.

MarkerNumber

TimePosition

Returns

Success (0) or error code.

◆ PwrSnsr_SetMeasBuffEnabled()

```
EXPORT int
PwrSnsr_SetMeasBuf
fEnabled          (          SessionID          Vi,
                    int          MeasBuffEnabled
                    )
```

Enable or disable the measurement buffer. Disabling the measurement buffer enables modulated/CW measurements. Conversely, enabling it disables modulated/CW measurements.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

MeasBuffEnabled

True to enable measurement buffer, false to disable.

Returns

Success (0) or error code.

◆ PwrSnsr_SetMesial()

```
EXPORT int
PwrSnsr_SetMesial (          SessionID          Vi,
                           const char *        Channel,
                           float               Mesial
                           )
```

Set the pulse amplitude percentage, which is used to define the midpoint of a rising edge or end of a falling edge transition.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Mesial**Returns**

Success (0) or error code.

◆ PwrSnsr_SetOffsetdB()

```
EXPORT int
PwrSnsr_SetOffsetdB
(
    SessionID      Vi,
    const char *   Channel,
    float          OffsetdB
)
```

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.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

OffsetdB**Returns**

Success (0) or error code.

◆ PwrSnsr_SetPeakHoldDecay()

```
EXPORT int
PwrSnsr_SetPeakHoldDecay
(
    SessionID      Vi,
    const char *   Channel,
    int            PeakHoldDecay
)
```

Set the number of min/max traces averaged together to form the peak hold measurement results on the selected channel.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeAverage	Peak hold decay value.
Returns	
Success (0) or error code.	

◆ PwrSnsr_SetPeakHoldTracking()

```
EXPORT int
PwrSnsr_SetPeakHoldTracking (
                                SessionID      Vi,
                                const char *    Channel,
                                int             EnvelopeTracking
)
```

Sets whether peak hold decay automatically tracks trace averaging. If set to true, the peak hold decay and trace averaging values are the same. If set to false, peak hold decay is independent.

Parameters

Vi	The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.
Channel	Channel number. For single instruments, set this to "CH1".
EnvelopeTracking	Boolean value. True to set peak hold tracking on.

Returns

Success (0) or error code.

◆ PwrSnsr_SetPercentPosition()

```
EXPORT int
PwrSnsr_SetPercentPosition (
                                SessionID      Vi,
                                const char *    Channel,
                                double          PercentPosition
)
```

Set the cursor percent on the CCDF plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

PercentPosition

Returns

Success (0) or error code.

◆ PwrSnsr_SetPeriod()

```
EXPORT int
PwrSnsr_SetPeriod (          SessionID      Vi,
                             float          Period
                             )
```

Set the period each timed mode acquisition (measurement buffer) is started.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Period

The period in seconds each timed mode acquisition is started.

Returns

Success (0) or error code.

◆ PwrSnsr_SetPowerPosition()

```
EXPORT int
PwrSnsr_SetPowerP
osition          (          SessionID      Vi,
                             const char *  Channel,
                             double        PowerPosition
                             )
```

Set the cursor power in dB on the CCDF plot.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

Channel number. For single instruments, set this to "CH1".

PowerPosition**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetProximal()**

```
EXPORT int
PwrSnsr_SetProximal (
    SessionID
    const char *
    float
    Vi,
    Channel,
    Proximal
)
```

Set the pulse amplitude percentage, which is used to define the beginning of a rising edge or end of a falling edge transition.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Proximal**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetPulseUnits()**

```
EXPORT int
PwrSnsr_SetPulseUnits (
    SessionID
    const char *
    PwrSnsrPulseUnitsEnum
    Vi,
    Channel,
    PwrSnsrPulseUnitsEnum
)
```

Set the units for entering the pulse distal, mesial and proximal levels.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

Channel number. For single instruments, set this to "CH1".

PwrSnsrPulseUnitsEnum**Returns**

Success (0) or error code.

◆ PwrSnsr_SetRdgsEnableFlag()

```
EXPORT int  
PwrSnsr_SetRdgsEn  
ableFlag          (          SessionID          Vi,  
                   int          Flag  
                   )
```

Set the flag indicating which measurement buffer arrays will be read when calling PwrSnsr_AcquireMeasurements.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Flag

Bit masked value indicating which measurement arrays will be queried (see PwrSnsrRdgsEnableFlag).

Returns

Success (0) or error code.

◆ PwrSnsr_SetReturnCount()

```
EXPORT int  
PwrSnsr_SetReturnC  
ount              (          SessionID          Vi,  
                   int          ReturnCount  
                   )
```

Set the return count for each measurement query.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

ReturnCount

The return count for each measurement query.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetSessionCount()**

```
EXPORT int
PwrSnsr_SetSession
Count          (          SessionID          Vi,
                                     int          SessionCount
                                     )
```

Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

SessionCount

Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetSessionTimeout()**

```
EXPORT int
PwrSnsr_SetSession
Timeout        (          SessionID          Vi,
                                     float          Seconds
                                     )
```

Set the count of elements for this measurement buffer session. Set to 0 for the meter to continuously acquire measurements.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Seconds

Set the time out value. Values less than or equal to 0 will be treated as infinite. Valid range : 0.001 to 1000

Returns

Success (0) or error code.

◆ PwrSnsr_SetSlaveSkew()

```
EXPORT int
PwrSnsr_SetSlaveSkew (
    SessionID          Vi,
    const char *       Channel,
    float               SlaveSkew
)
```

Sets the skew in seconds for the slave trigger.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

SlaveSkew

Trigger skew in seconds (-1e-6 to 1e-6).

Returns

Success (0) or error code.

◆ PwrSnsr_SetStartDelay()

```
EXPORT int
PwrSnsr_SetStartDelay (
    SessionID          Vi,
    float               StartDelay
)
```

Set delay time added to the detected beginning of a burst for analysis. Typically used to exclude the rising edge of a burst.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartDelay

Delay time in seconds added to the detected beginning of a burst for analysis.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetStartGate()**

```
EXPORT int
PwrSnsr_SetStartGate
(
    SessionID          Vi,
    const char *       Channel,
    float              StartGate
)
```

Set the point on a pulse, which is used to define the beginning of the pulse's active interval.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

StartGate**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetStartMode()**

```
EXPORT int
PwrSnsr_SetStartMode
(
    SessionID          Vi,
    PwrSnsrMeasBuffStartModeEnum StartMode
)
```

Set the mode used to start acquisition of buffer entries.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartMode

Mode used to start acquisition of buffer entries.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetStartQual()**

```
EXPORT int
PwrSnsr_SetStartQual (
    SessionID float Vi,
    StartQual
)
```

Set the minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

StartQual

The minimum amount of time power remains above the trigger point to be counted as the beginning of a burst.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTempComp()**

```
EXPORT int
PwrSnsr_SetTempComp (
    SessionID const char * Vi,
    int Channel,
    TempComp
)
```

Set the state of the peak sensor temperature compensation system.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TempComp

Boolean. 1 for on; 0 for off.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermAction()

```
EXPORT int
PwrSnsr_SetTermAct
ion          (          SessionID          Vi,
               const char *          Channel,
               PwrSnsrTermAction          TermAction
               Enum
               )
```

Set the termination action for statistical capturing.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermAction

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermCount()

```
EXPORT int
PwrSnsr_SetTermCo
unt          (          SessionID          Vi,
               const char *          Channel,
               double          TermCount
               )
```

Set the termination count for statistical capturing. After the sample count has been reached, the action determined by TermAction is taken.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermCount

Returns

Success (0) or error code.

◆ PwrSnsr_SetTermTime()

```
EXPORT int
PwrSnsr_SetTermTime
(
    SessionID
    const char *
    int
    Vi,
    Channel,
    TermTime
)
```

Set the termination time in seconds (1 - 3600) for statistical capturing. After the time has elapsed, the action determined by TermAction is taken.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

TermTime**Returns**

Success (0) or error code.

◆ PwrSnsr_SetTimebase()

```
EXPORT int
PwrSnsr_SetTimebase
(
    SessionID
    float
    Vi,
    Timebase
)
```

Set the Pulse Mode timebase in seconds/division. (10 divisions = 1 trace) Value = 5e-9 to 10e-3 sec (or max timebase) in a 1-2-5 sequence,.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timebase**Returns**

Success (0) or error code.

◆ [PwrSnsr_SetTimeOut\(\)](#)

```
EXPORT int
PwrSnsr_SetTimeOut (          SessionID      Vi,
                              long            Milliseconds
                              )
```

Sets the time out in milliseconds for I/O.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Milliseconds

Time out in milliseconds. Use -1 for infinite time out.

Returns

Success (0) or error code.

◆ [PwrSnsr_SetTimespan\(\)](#)

```
EXPORT int
PwrSnsr_SetTimespan (          SessionID      Vi,
                              float            Timespan
                              )
```

Set the horizontal time span of the trace in pulse mode. Time span = 10* Time/Division.
Value = 5e-8 to 100e-3 sec in a 1-2-5 sequence.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Timespan

Returns

Success (0) or error code.

◆ [PwrSnsr_SetTrigDelay\(\)](#)

```
EXPORT int
PwrSnsr_SetTrigDelay (
                                SessionID    Vi,
                                float         Delay
)
```

Sets the trigger delay time in seconds with respect to the trigger for the trigger display location in the LEFT position.

Positive values cause the actual trigger to occur after the trigger condition is met. This places the trigger event to the left of the trigger point on the display, and is useful for viewing events during a pulse, some fixed delay time after the rising edge trigger. Negative trigger delay places the trigger event to the right of the trigger point on the display, and is useful for looking at events before the trigger edge.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Delay

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigHoldoff()

```
EXPORT int
PwrSnsr_SetTrigHoldoff (
                                SessionID    Vi,
                                float         Holdoff
)
```

Sets the trigger holdoff time in seconds.

Trigger holdoff is used to disable the trigger for a specified amount of time after each trigger event. The holdoff time starts immediately after each valid trigger edge, and will not permit any new triggers until the time has expired. When the holdoff time is up, the trigger re-arms, and the next valid trigger event (edge) will cause a new sweep. This feature is used to help synchronize the power meter with burst waveforms such as a TDMA or GSM frame. The trigger holdoff resolution is 10 nanoseconds, and it should be set to a time that is just slightly shorter than the frame repetition interval.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Holdoff**Returns**

Success (0) or error code.

◆ **PwrSnsr_SetTrigHoldoffMode()**

```
EXPORT int
PwrSnsr_SetTrigHoldoffMode (
                                SessionID      Vi,
                                PwrSnsrHoldoffModeEnum HoldoffMode
                                )
```

Sets the holdoff mode to normal or gap holdoff.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

HoldoffMode

Holdoff mode.

Returns

Success (0) or error code.

◆ **PwrSnsr_SetTrigLevel()**

```
EXPORT int
PwrSnsr_SetTrigLevel (
                                SessionID      Vi,
                                float          Level
                                )
```

Set the trigger level for synchronizing data acquisition with a pulsed input signal.

The internal trigger level entered should include any global offset and will also be affected by the frequency cal factor. The available internal trigger level range is sensor dependent. The trigger level is set and returned in dBm. This setting is only valid for normal and auto trigger modes.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Level

Trigger level in dBm.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigMode()

```
EXPORT int
PwrSnsr_SetTrigMod
e                (                SessionID                Vi,
                                PwrSnsrTriggerMod
                                eEnum                Mode
                                )
```

Set the trigger mode for synchronizing data acquisition with pulsed signals.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Mode

Trigger mode.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigOutMode()

```
EXPORT int
PwrSnsr_SetTrigOut
Mode                (                SessionID                Vi,
                                const char *                Channel,
                                int                Mode
                                )
```

Sets the trigger out/mult io mode. Setting trigger mode overrides this command.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Mode

Trigger out/multi IO mode

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigPosition()

```
EXPORT int
PwrSnsr_SetTrigPosi
tion          (          SessionID          Vi,
                  PwrSnsrTriggerPosi
                  tionEnum          Position
                )
```

Set the position of the trigger event on displayed sweep.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Position

Trigger position.

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigSlope()

```
EXPORT int
PwrSnsr_SetTrigSlop
e          (          SessionID          Vi,
                  PwrSnsrTriggerSlop
                  eEnum          Slope
                )
```

Sets the trigger slope or polarity.

When set to positive, trigger events will be generated when a signal rising edge crosses the trigger level threshold. When negative, trigger events are generated on the falling edge of the pulse.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

identifies a particular instrument session.

Slope

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigSource()

```
EXPORT int
PwrSnsr_SetTrigSource (
                                SessionID      Vi,
                                PwrSnsrTriggerSourceEnum Source
)
```

Get the signal the power meter monitors for a trigger. It can be channel external input, or independent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Source

Returns

Success (0) or error code.

◆ PwrSnsr_SetTrigVernier()

```
EXPORT int
PwrSnsr_SetTrigVernier (
                                SessionID      Vi,
                                float Vernier
)
```

Set the fine position of the trigger event on the power sweep.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Vernier

Trigger position -30.0 to 30.0 (0.0 = left, 5.0 = middle, 10.0 = Right).

Returns

Success (0) or error code.

◆ PwrSnsr_SetUnits()

```
EXPORT int
PwrSnsr_SetUnits    (
                                SessionID      Vi,
                                const char *    Channel,
                                PwrSnsrUnitsEnum Units
                                )
```

Set units for the selected channel.

Voltage is calculated with reference to the sensor input impedance. Note that for ratiometric results, logarithmic units will always return dBr (dB relative) while linear units return percent.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Units

Returns

Success (0) or error code.

◆ PwrSnsr_SetVerticalCenter()

```
int EXPORT
PwrSnsr_SetVertical
Center    (
                                SessionID      Vi,
                                const char *    Channel,
                                float           VerticalCenter
                                )
```

Sets vertical center based on current units: <arg> = (range varies by units)

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical center in units

Returns

Success (0) or error code.

◆ `PwrSnsr_SetVerticalScale()`

```
int EXPORT
PwrSnsr_SetVertical
Scale          (          SessionID          Vi,
                  const char *          Channel,
                  float          VerticalScale
                  )
```

Sets vertical scale based on current units: <arg> = (range varies by units)

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

VerticalCenter

Vertical scale in units

Returns

Success (0) or error code.

◆ `PwrSnsr_SetWriteProtection()`

```
EXPORT int
PwrSnsr_SetWritePr
otection          (          SessionID          Vi,
                  int          WriteProtection
                  )
```

Set whether to allow the measurement buffer to overwrite entries that have not been read by the user.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

WriteProtection

Set false to allow the measurement buffer to overwrite entries that have not been read by the user.

Returns

Success (0) or error code.

◆ PwrSnsr_StartAcquisition()

```
EXPORT int
PwrSnsr_Start
Acquisition (          SessionID      Vi          )
```

Starts measurement buffer acquisition. This method allows the user to send a command to the power meter to begin buffering measurements without waiting for all measurements to be completed. Alternately, you can call the AcquireReadings method to start buffering measurements and wait for them to be read from the meter.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ PwrSnsr_StatModeReset()

```
EXPORT int
PwrSnsr_StatModeR
eset (          SessionID      Vi,
          const char *      Channel
        )
```

Resets statistical capturing mode by clearing the buffers and restarting the acquisition timer.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_Status()

```
EXPORT int
PwrSnsr_Status      (
                                SessionID      Vi,
                                PwrSnsrAcquisition
                                StatusEnum *    Val
                                )
```

Returns whether an acquisition is in progress, complete, or if the status is unknown.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Val

Status out parameter.

Returns

Success (0) or error code.

◆ [PwrSnsr_StopAcquisition\(\)](#)

```
EXPORT int
PwrSnsr_Stop
Acquisition      (
                                SessionID      Vi
                                )
```

Sends a command to stop the measurement buffer from acquiring readings.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Returns

Success (0) or error code.

◆ [PwrSnsr_Write\(\)](#)

```
EXPORT int
PwrSnsr_Write      (
                                SessionID      Vi,
                                const char *    Channel,
                                int             DataBufferSize,
                                unsigned char   Data[]
                                )
```

Write a byte array to the meter.

Parameters

Vi

The SessionID handle that you obtain from the PwrSnsr_init function. The handle

Channel

identifies a particular instrument session.

DataBufferSize

Channel number. For single instruments, set this to "CH1".

Data

Size of the buffer in bytes.

Returns

Data to send.

Success (0) or error code.

◆ PwrSnsr_Zero()

```
EXPORT int
PwrSnsr_Zero      (      SessionID      Vi,
                      const char *      Channel
                      )
```

Performs a zero offset null adjustment.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Returns

Success (0) or error code.

◆ PwrSnsr_ZeroQuery()

```
EXPORT int
PwrSnsr_ZeroQuery (      SessionID      Vi,
                      const char *      Channel,
                      int *              Val
                      )
```

Performs a zero offset null adjustment and returns true if successful.

Parameters**Vi**

The SessionID handle that you obtain from the PwrSnsr_init function. The handle identifies a particular instrument session.

Channel

Channel number. For single instruments, set this to "CH1".

Val

Boolean value for operation success or failure.

Returns

Success (0) or error code.

Generated by  1.8.15

2.2 Globals

Power Sensor Library

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- PulseInfo : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_ALREADY_INITIALIZED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_INVALID_SESSION_HANDLE : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_NOT_INITIALIZED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_NULL_POINTER : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_OPERATION_PENDING : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_OUT_OF_MEMORY : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_RESET_FAILED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_RESOURCE_UNKNOWN : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_UNEXPECTED_RESPONSE : [PwrSnsrLib.h](#)
- PWR_SNSR_INV_PARAMETER : [PwrSnsrLib.h](#)
- PWR_SNSR_IO_GENERAL : [PwrSnsrLib.h](#)
- PWR_SNSR_IO_TIMEOUT : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_ACCESS : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_BUSY : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_INTERRUPTED : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_INVALID_PARAM : [PwrSnsrLib.h](#)

- PWR_SNSR_LIBUSB_ERROR_IO : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NO_DEVICE : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NO_MEM : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NOT_FOUND : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_OTHER : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_OVERFLOW : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_PIPE : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_TIMEOUT : [PwrSnsrLib.h](#)
- PWR_SNSR_MODEL_NOT_SUPPORTED : [PwrSnsrLib.h](#)
- PwrSnsr_Abort() : [PwrSnsrLib.h](#)
- PwrSnsr_AcquireMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_AdvanceReadIndex() : [PwrSnsrLib.h](#)
- PwrSnsr_Clear() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearBuffer() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearError() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearUserCal() : [PwrSnsrLib.h](#)
- PwrSnsr_close() : [PwrSnsrLib.h](#)
- PwrSnsr_EnableCapturePriority() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchAllMultiPulse() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchArrayMarkerPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFPercent() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFTrace() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCursorPercent() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCursorPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCWArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCWPower() : [PwrSnsrLib.h](#)

- PwrSnsr_FetchDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchDutyCycle() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchEdgeDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchExtendedWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchFallTime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIEEEBottom() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIEEETop() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalFilteredMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalFilteredMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMaxAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMinAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalPkToAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchOfftime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchOvershoot() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPowerArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPRF() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchProximal() : [PwrSnsrLib.h](#)

- PwrSnsr_FetchPulseCycleAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPulseOnAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPulsePeak() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchRiseTime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchStatMeasurementArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchTimeArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchWaveformMinMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchWidth() : [PwrSnsrLib.h](#)
- PwrSnsr_FindResources() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAcqStatusArray() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAttenuation() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBufferedAverageMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBufferedMeasurementsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCalFactor() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCalFactors() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCCDFTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChannelByIndex() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChannelCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChanTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCurrentTemp() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDiagStatusArray() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDongleSerialNumber() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDuration() : [PwrSnsrLib.h](#)

- PwrSnsr_GetDurations() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndGate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndQual() : [PwrSnsrLib.h](#)
- PwrSnsr_GetError() : [PwrSnsrLib.h](#)
- PwrSnsr_GetExpirationDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetExternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFactoryCalDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFetchLatency() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFilterState() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFilterTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFirmwareVersion() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFpgaVersion() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFrequency() : [PwrSnsrLib.h](#)
- PwrSnsr_GetGateMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetGating() : [PwrSnsrLib.h](#)
- PwrSnsr_GetHorizontalOffset() : [PwrSnsrLib.h](#)
- PwrSnsr_GetHorizontalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_GetImpedance() : [PwrSnsrLib.h](#)
- PwrSnsr_GetInitiateContinuous() : [PwrSnsrLib.h](#)
- PwrSnsr_GetInternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsAvgSensor() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsRunning() : [PwrSnsrLib.h](#)
- PwrSnsr_GetManufactureDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMarkerPixelPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMarkerTimePosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxFreqHighBandwidth() : [PwrSnsrLib.h](#)

- PwrSnsr_GetMaxFreqLowBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMeasBuffEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMeasurementsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMemChanArchive() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinFreqHighBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinFreqLowBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinimumSupportedFirmware() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinimumTrig() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetModel() : [PwrSnsrLib.h](#)
- PwrSnsr_GetNumberOfCals() : [PwrSnsrLib.h](#)
- PwrSnsr_GetOffsetdB() : [PwrSnsrLib.h](#)
- PwrSnsr_GetOverRan() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakHoldDecay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakHoldTracking() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakPowerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakPowerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPercentPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPowerPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetProximal() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPulseUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_GetRdgsEnableFlag() : [PwrSnsrLib.h](#)
- PwrSnsr_GetReadingPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_GetReturnCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSequenceNumbers() : [PwrSnsrLib.h](#)

- PwrSnsr_GetSerialNumber() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSessionCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSlaveSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartGate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartQual() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartTimes() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSweepTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTempComp() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermAction() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimedOut() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimeOut() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimePerPoint() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimespan() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTraceStartTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigHoldoff() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigHoldoffMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigLevel() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigSlope() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigSource() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigStatus() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigVernier() : [PwrSnsrLib.h](#)

- PwrSnsr_GetUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_GetVerticalCenter() : [PwrSnsrLib.h](#)
- PwrSnsr_GetVerticalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_GetWriteProtection() : [PwrSnsrLib.h](#)
- PwrSnsr_init() : [PwrSnsrLib.h](#)
- PwrSnsr_InitiateAquisition() : [PwrSnsrLib.h](#)
- PwrSnsr_IsLicenseDongleConnected() : [PwrSnsrLib.h](#)
- PwrSnsr_LoadMemChanFromArchive() : [PwrSnsrLib.h](#)
- PwrSnsr_MeasurePower() : [PwrSnsrLib.h](#)
- PwrSnsr_MeasureVoltage() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryAverageMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryDurations() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryMaxMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryMinMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QuerySequenceNumbers() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryStartTimes() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadArrayMarkerPower() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadByteArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadControl() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadCWArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadCWPower() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadDutyCycle() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadEdgeDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadFallTime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIEEEBottom() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIEEETop() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalFilteredMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalFilteredMin() : [PwrSnsrLib.h](#)

- PwrSnsr_ReadIntervalMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMaxAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMin() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMinAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalPkToAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadOfftime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadOvershoot() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPowerArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPRF() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulseCycleAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulseOnAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulsePeak() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadRiseTime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPI() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPIBytes() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPIFromNamedParser() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadTimeArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWaveformMinMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWidth() : [PwrSnsrLib.h](#)
- PwrSnsr_reset() : [PwrSnsrLib.h](#)

- PwrSnsr_ResetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SaveToMemoryChannel() : [PwrSnsrLib.h](#)
- PwrSnsr_SaveUserCal() : [PwrSnsrLib.h](#)
- PwrSnsr_self_test() : [PwrSnsrLib.h](#)
- PwrSnsr_SendSCPIBytes() : [PwrSnsrLib.h](#)
- PwrSnsr_SendSCPICommand() : [PwrSnsrLib.h](#)
- PwrSnsr_SendSCPIToNamedParser() : [PwrSnsrLib.h](#)
- PwrSnsr_SetAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_SetBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCalFactor() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCCDFTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SetDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_SetDuration() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndGate() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndQual() : [PwrSnsrLib.h](#)
- PwrSnsr_SetExternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFetchLatency() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFilterState() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFilterTime() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFrequency() : [PwrSnsrLib.h](#)
- PwrSnsr_SetGateMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetGating() : [PwrSnsrLib.h](#)
- PwrSnsr_SetHorizontalOffset() : [PwrSnsrLib.h](#)
- PwrSnsr_SetHorizontalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_SetInitiateContinuous() : [PwrSnsrLib.h](#)

- PwrSnsr_SetInternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMarkerPixelPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMarkerTimePosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMeasBuffEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_SetOffsetdB() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPeakHoldDecay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPeakHoldTracking() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPercentPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPowerPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetProximal() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPulseUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_SetRdgsEnableFlag() : [PwrSnsrLib.h](#)
- PwrSnsr_SetReturnCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSessionCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSessionTimeout() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSlaveSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartGate() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartQual() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTempComp() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermAction() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermTime() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimeOut() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimespan() : [PwrSnsrLib.h](#)

- PwrSnsr_SetTrigDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigHoldoff() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigHoldoffMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigLevel() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigOutMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigSlope() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigSource() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigVernier() : [PwrSnsrLib.h](#)
- PwrSnsr_SetUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_SetVerticalCenter() : [PwrSnsrLib.h](#)
- PwrSnsr_SetVerticalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_SetWriteProtection() : [PwrSnsrLib.h](#)
- PwrSnsr_StartAcquisition() : [PwrSnsrLib.h](#)
- PwrSnsr_StatModeReset() : [PwrSnsrLib.h](#)
- PwrSnsr_Status() : [PwrSnsrLib.h](#)
- PwrSnsr_StopAcquisition() : [PwrSnsrLib.h](#)
- PwrSnsr_Write() : [PwrSnsrLib.h](#)
- PwrSnsr_Zero() : [PwrSnsrLib.h](#)
- PwrSnsr_ZeroQuery() : [PwrSnsrLib.h](#)
- PwrSnsrAcqComplete : [PwrSnsrLib.h](#)
- PwrSnsrAcqInProgress : [PwrSnsrLib.h](#)
- PwrSnsrAcqStatusUnknown : [PwrSnsrLib.h](#)
- PwrSnsrAcquisitionStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrAvgEnable : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthEnum : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthHigh : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthLow : [PwrSnsrLib.h](#)

- PwrSnsrCondCodeEnum : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeError : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeMeasurementStopped : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeNormal : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeOverrange : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeUnderrange : [PwrSnsrLib.h](#)
- PwrSnsrDurationEnable : [PwrSnsrLib.h](#)
- PwrSnsrErrorCodesEnum : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateAuto : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateEnum : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateOff : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateOn : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeGap : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeNormal : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberEnum : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberMarker1 : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberMarker2 : [PwrSnsrLib.h](#)
- PwrSnsrMaxEnable : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffGateEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffStartModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffStopReasonEnum : [PwrSnsrLib.h](#)
- PwrSnsrMinEnable : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsEnum : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsVolts : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsWatts : [PwrSnsrLib.h](#)
- PwrSnsrRdgsEnableFlag : [PwrSnsrLib.h](#)
- PwrSnsrSequenceEnable : [PwrSnsrLib.h](#)
- PwrSnsrStartTimeEnable : [PwrSnsrLib.h](#)

- PwrSnsrStatGatingEnum : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingFreeRun : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingMarkers : [PwrSnsrLib.h](#)
- PwrSnsrTermActionDecimate : [PwrSnsrLib.h](#)
- PwrSnsrTermActionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTermActionRestart : [PwrSnsrLib.h](#)
- PwrSnsrTermActionStop : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeAuto : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeAutoLevel : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeNormal : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionLeft : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionMiddle : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionRight : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopeNegative : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopePositive : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel1 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel10 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel11 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel12 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel13 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel14 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel15 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel16 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel2 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel3 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel4 : [PwrSnsrLib.h](#)

- PwrSnsrTriggerSourceChannel5 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel6 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel7 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel8 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel9 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceExternal : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceIndependent : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusAcquiringNew : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusAutoTrig : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusFreerun : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusPretrig : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusRunning : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusStopped : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusTriggered : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusWaiting : [PwrSnsrLib.h](#)
- PwrSnsrTrigOutModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrUnitsdBm : [PwrSnsrLib.h](#)
- PwrSnsrUnitsDBMV : [PwrSnsrLib.h](#)
- PwrSnsrUnitsDBUV : [PwrSnsrLib.h](#)
- PwrSnsrUnitsDBV : [PwrSnsrLib.h](#)
- PwrSnsrUnitsEnum : [PwrSnsrLib.h](#)
- PwrSnsrUnitsvolts : [PwrSnsrLib.h](#)
- PwrSnsrUnitswatts : [PwrSnsrLib.h](#)

2.2.2 Functions

Power Sensor Library

- PwrSnsr_Abort() : [PwrSnsrLib.h](#)
- PwrSnsr_AcquireMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_AdvanceReadIndex() : [PwrSnsrLib.h](#)
- PwrSnsr_Clear() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearBuffer() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearError() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_ClearUserCal() : [PwrSnsrLib.h](#)
- PwrSnsr_close() : [PwrSnsrLib.h](#)
- PwrSnsr_EnableCapturePriority() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchAllMultiPulse() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchArrayMarkerPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFPercent() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCCDFTrace() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCursorPercent() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCursorPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCWArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchCWPower() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchDutyCycle() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchEdgeDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchExtendedWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchFallTime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIEEEBottom() : [PwrSnsrLib.h](#)

- PwrSnsr_FetchIEEETop() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalFilteredMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalFilteredMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMaxAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalMinAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchIntervalPkToAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMarkerRRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchOfftime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchOvershoot() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPowerArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPRF() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchProximal() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPulseCycleAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPulseOnAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchPulsePeak() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchRiseTime() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchStatMeasurementArray() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchTimeArray() : [PwrSnsrLib.h](#)

- PwrSnsr_FetchWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchWaveformMinMax() : [PwrSnsrLib.h](#)
- PwrSnsr_FetchWidth() : [PwrSnsrLib.h](#)
- PwrSnsr_FindResources() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAcqStatusArray() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAttenuation() : [PwrSnsrLib.h](#)
- PwrSnsr_GetAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBufferedAverageMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetBufferedMeasurementsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCalFactor() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCalFactors() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCCDFTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChannelByIndex() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChannelCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetChanTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_GetCurrentTemp() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDiagStatusArray() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDongleSerialNumber() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDuration() : [PwrSnsrLib.h](#)
- PwrSnsr_GetDurations() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndGate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetEndQual() : [PwrSnsrLib.h](#)
- PwrSnsr_GetError() : [PwrSnsrLib.h](#)

- PwrSnsr_GetExpirationDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetExternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFactoryCalDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFetchLatency() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFilterState() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFilterTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFirmwareVersion() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFpgaVersion() : [PwrSnsrLib.h](#)
- PwrSnsr_GetFrequency() : [PwrSnsrLib.h](#)
- PwrSnsr_GetGateMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetGating() : [PwrSnsrLib.h](#)
- PwrSnsr_GetHorizontalOffset() : [PwrSnsrLib.h](#)
- PwrSnsr_GetHorizontalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_GetImpedance() : [PwrSnsrLib.h](#)
- PwrSnsr_GetInitiateContinuous() : [PwrSnsrLib.h](#)
- PwrSnsr_GetInternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsAvgSensor() : [PwrSnsrLib.h](#)
- PwrSnsr_GetIsRunning() : [PwrSnsrLib.h](#)
- PwrSnsr_GetManufactureDate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMarkerPixelPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMarkerTimePosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxFreqHighBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxFreqLowBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMaxTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMeasBuffEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMeasurementsAvailable() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMemChanArchive() : [PwrSnsrLib.h](#)

- PwrSnsr_GetMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinFreqHighBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinFreqLowBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinimumSupportedFirmware() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinimumTrig() : [PwrSnsrLib.h](#)
- PwrSnsr_GetMinMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_GetModel() : [PwrSnsrLib.h](#)
- PwrSnsr_GetNumberOfCals() : [PwrSnsrLib.h](#)
- PwrSnsr_GetOffsetdB() : [PwrSnsrLib.h](#)
- PwrSnsr_GetOverRan() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakHoldDecay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakHoldTracking() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakPowerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeakPowerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPercentPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPowerPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetProximal() : [PwrSnsrLib.h](#)
- PwrSnsr_GetPulseUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_GetRdgsEnableFlag() : [PwrSnsrLib.h](#)
- PwrSnsr_GetReadingPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_GetReturnCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSequenceNumbers() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSerialNumber() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSessionCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSlaveSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartGate() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartMode() : [PwrSnsrLib.h](#)

- PwrSnsr_GetStartQual() : [PwrSnsrLib.h](#)
- PwrSnsr_GetStartTimes() : [PwrSnsrLib.h](#)
- PwrSnsr_GetSweepTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTempComp() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermAction() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermCount() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTermTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimedOut() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimeOut() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimePerPoint() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTimespan() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTraceStartTime() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigHoldoff() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigHoldoffMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigLevel() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigMode() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigSlope() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigSource() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigStatus() : [PwrSnsrLib.h](#)
- PwrSnsr_GetTrigVernier() : [PwrSnsrLib.h](#)
- PwrSnsr_GetUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_GetVerticalCenter() : [PwrSnsrLib.h](#)
- PwrSnsr_GetVerticalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_GetWriteProtection() : [PwrSnsrLib.h](#)
- PwrSnsr_init() : [PwrSnsrLib.h](#)
- PwrSnsr_InitiateAquisition() : [PwrSnsrLib.h](#)

- PwrSnsr_IsLicenseDongleConnected() : [PwrSnsrLib.h](#)
- PwrSnsr_LoadMemChanFromArchive() : [PwrSnsrLib.h](#)
- PwrSnsr_MeasurePower() : [PwrSnsrLib.h](#)
- PwrSnsr_MeasureVoltage() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryAverageMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryDurations() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryMaxMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryMinMeasurements() : [PwrSnsrLib.h](#)
- PwrSnsr_QuerySequenceNumbers() : [PwrSnsrLib.h](#)
- PwrSnsr_QueryStartTimes() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadArrayMarkerPower() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadByteArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadControl() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadCWArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadCWPower() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadDutyCycle() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadEdgeDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadFallTime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIEEEBottom() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIEEETop() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalFilteredMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalFilteredMin() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMaxAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMin() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalMinAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadIntervalPkToAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerAverage() : [PwrSnsrLib.h](#)

- PwrSnsr_ReadMarkerDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerMin() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRDelta() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadMarkerRRatio() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadOfftime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadOvershoot() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPowerArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPRF() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulseCycleAvg() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulseOnAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadPulsePeak() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadRiseTime() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPI() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPIBytes() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadSCPIFromNamedParser() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadTimeArray() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWaveform() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWaveformMinMax() : [PwrSnsrLib.h](#)
- PwrSnsr_ReadWidth() : [PwrSnsrLib.h](#)
- PwrSnsr_reset() : [PwrSnsrLib.h](#)
- PwrSnsr_ResetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SaveToMemoryChannel() : [PwrSnsrLib.h](#)
- PwrSnsr_SaveUserCal() : [PwrSnsrLib.h](#)
- PwrSnsr_self_test() : [PwrSnsrLib.h](#)
- PwrSnsr_SendSCPIBytes() : [PwrSnsrLib.h](#)
- PwrSnsr_SendSCPICommand() : [PwrSnsrLib.h](#)

- PwrSnsr_SendSCPIToNamedParser() : [PwrSnsrLib.h](#)
- PwrSnsr_SetAverage() : [PwrSnsrLib.h](#)
- PwrSnsr_SetBandwidth() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCalFactor() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SetCCDFTraceCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetContinuousCapture() : [PwrSnsrLib.h](#)
- PwrSnsr_SetDistal() : [PwrSnsrLib.h](#)
- PwrSnsr_SetDuration() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndGate() : [PwrSnsrLib.h](#)
- PwrSnsr_SetEndQual() : [PwrSnsrLib.h](#)
- PwrSnsr_SetExternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFetchLatency() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFilterState() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFilterTime() : [PwrSnsrLib.h](#)
- PwrSnsr_SetFrequency() : [PwrSnsrLib.h](#)
- PwrSnsr_SetGateMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetGating() : [PwrSnsrLib.h](#)
- PwrSnsr_SetHorizontalOffset() : [PwrSnsrLib.h](#)
- PwrSnsr_SetHorizontalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_SetInitiateContinuous() : [PwrSnsrLib.h](#)
- PwrSnsr_SetInternalSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMarkerPixelPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMarkerTimePosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMeasBuffEnabled() : [PwrSnsrLib.h](#)
- PwrSnsr_SetMesial() : [PwrSnsrLib.h](#)
- PwrSnsr_SetOffsetdB() : [PwrSnsrLib.h](#)

- PwrSnsr_SetPeakHoldDecay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPeakHoldTracking() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPercentPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPeriod() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPowerPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetProximal() : [PwrSnsrLib.h](#)
- PwrSnsr_SetPulseUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_SetRdgsEnableFlag() : [PwrSnsrLib.h](#)
- PwrSnsr_SetReturnCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSessionCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSessionTimeout() : [PwrSnsrLib.h](#)
- PwrSnsr_SetSlaveSkew() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartGate() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetStartQual() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTempComp() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermAction() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermCount() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTermTime() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimebase() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimeOut() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTimespan() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigDelay() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigHoldoff() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigHoldoffMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigLevel() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigMode() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigOutMode() : [PwrSnsrLib.h](#)

- PwrSnsr_SetTrigPosition() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigSlope() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigSource() : [PwrSnsrLib.h](#)
- PwrSnsr_SetTrigVernier() : [PwrSnsrLib.h](#)
- PwrSnsr_SetUnits() : [PwrSnsrLib.h](#)
- PwrSnsr_SetVerticalCenter() : [PwrSnsrLib.h](#)
- PwrSnsr_SetVerticalScale() : [PwrSnsrLib.h](#)
- PwrSnsr_SetWriteProtection() : [PwrSnsrLib.h](#)
- PwrSnsr_StartAcquisition() : [PwrSnsrLib.h](#)
- PwrSnsr_StatModeReset() : [PwrSnsrLib.h](#)
- PwrSnsr_Status() : [PwrSnsrLib.h](#)
- PwrSnsr_StopAcquisition() : [PwrSnsrLib.h](#)
- PwrSnsr_Write() : [PwrSnsrLib.h](#)
- PwrSnsr_Zero() : [PwrSnsrLib.h](#)
- PwrSnsr_ZeroQuery() : [PwrSnsrLib.h](#)

Generated by  1.8.15

2.2.3 Typedefs

Power Sensor Library

- PulseInfo : [PwrSnsrLib.h](#)
- PwrSnsrAcquisitionStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthEnum : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeEnum : [PwrSnsrLib.h](#)
- PwrSnsrErrorCodesEnum : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateEnum : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberEnum : [PwrSnsrLib.h](#)

- PwrSnsrMeasBuffGateEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffStartModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffStopReasonEnum : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsEnum : [PwrSnsrLib.h](#)
- PwrSnsrRdgsEnableFlag : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingEnum : [PwrSnsrLib.h](#)
- PwrSnsrTermActionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrTrigOutModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrUnitsEnum : [PwrSnsrLib.h](#)

Generated by  1.8.15

2.2.4 Enumerations

Power Sensor Library

- PwrSnsrAcquisitionStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthEnum : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeEnum : [PwrSnsrLib.h](#)
- PwrSnsrErrorCodesEnum : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateEnum : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffGateEnum : [PwrSnsrLib.h](#)
- PwrSnsrMeasBuffStartModeEnum : [PwrSnsrLib.h](#)

- PwrSnsrMeasBuffStopReasonEnum : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsEnum : [PwrSnsrLib.h](#)
- PwrSnsrRdgsEnableFlag : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingEnum : [PwrSnsrLib.h](#)
- PwrSnsrTermActionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopeEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceEnum : [PwrSnsrLib.h](#)
- PwrSnsrTriggerStatusEnum : [PwrSnsrLib.h](#)
- PwrSnsrTrigOutModeEnum : [PwrSnsrLib.h](#)
- PwrSnsrUnitsEnum : [PwrSnsrLib.h](#)

Generated by  1.8.15

2.2.5 Enumerator

Power Sensor Library

- PWR_SNSR_ERROR_ALREADY_INITIALIZED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_INVALID_SESSION_HANDLE : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_NOT_INITIALIZED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_NULL_POINTER : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_OPERATION_PENDING : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_OUT_OF_MEMORY : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_RESET_FAILED : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_RESOURCE_UNKNOWN : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_STATUS_NOT_AVAILABLE : [PwrSnsrLib.h](#)
- PWR_SNSR_ERROR_UNEXPECTED_RESPONSE : [PwrSnsrLib.h](#)

- PWR_SNSR_INV_PARAMETER : [PwrSnsrLib.h](#)
- PWR_SNSR_IO_GENERAL : [PwrSnsrLib.h](#)
- PWR_SNSR_IO_TIMEOUT : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_ACCESS : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_BUSY : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_INTERRUPTED : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_INVALID_PARAM : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_IO : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NO_DEVICE : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NO_MEM : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NOT_FOUND : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_NOT_SUPPORTED : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_OTHER : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_OVERFLOW : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_PIPE : [PwrSnsrLib.h](#)
- PWR_SNSR_LIBUSB_ERROR_TIMEOUT : [PwrSnsrLib.h](#)
- PWR_SNSR_MODEL_NOT_SUPPORTED : [PwrSnsrLib.h](#)
- PwrSnsrAcqComplete : [PwrSnsrLib.h](#)
- PwrSnsrAcqInProgress : [PwrSnsrLib.h](#)
- PwrSnsrAcqStatusUnknown : [PwrSnsrLib.h](#)
- PwrSnsrAvgEnable : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthHigh : [PwrSnsrLib.h](#)
- PwrSnsrBandwidthLow : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeError : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeMeasurementStopped : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeNormal : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeOverrange : [PwrSnsrLib.h](#)
- PwrSnsrCondCodeUnderrange : [PwrSnsrLib.h](#)
- PwrSnsrDurationEnable : [PwrSnsrLib.h](#)

- PwrSnsrFilterStateAuto : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateOff : [PwrSnsrLib.h](#)
- PwrSnsrFilterStateOn : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeGap : [PwrSnsrLib.h](#)
- PwrSnsrHoldoffModeNormal : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberMarker1 : [PwrSnsrLib.h](#)
- PwrSnsrMarkerNumberMarker2 : [PwrSnsrLib.h](#)
- PwrSnsrMaxEnable : [PwrSnsrLib.h](#)
- PwrSnsrMinEnable : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsVolts : [PwrSnsrLib.h](#)
- PwrSnsrPulseUnitsWatts : [PwrSnsrLib.h](#)
- PwrSnsrSequenceEnable : [PwrSnsrLib.h](#)
- PwrSnsrStartTimeEnable : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingFreeRun : [PwrSnsrLib.h](#)
- PwrSnsrStatGatingMarkers : [PwrSnsrLib.h](#)
- PwrSnsrTermActionDecimate : [PwrSnsrLib.h](#)
- PwrSnsrTermActionRestart : [PwrSnsrLib.h](#)
- PwrSnsrTermActionStop : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeAuto : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeAutoLevel : [PwrSnsrLib.h](#)
- PwrSnsrTriggerModeNormal : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionLeft : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionMiddle : [PwrSnsrLib.h](#)
- PwrSnsrTriggerPositionRight : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopeNegative : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSlopePositive : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel1 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel10 : [PwrSnsrLib.h](#)
- PwrSnsrTriggerSourceChannel11 : [PwrSnsrLib.h](#)

- PwrSnsrTriggerSourceChannel12 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel13 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel14 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel15 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel16 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel2 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel3 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel4 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel5 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel6 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel7 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel8 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceChannel9 : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceExternal : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerSourceIndependent : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusAcquiringNew : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusAutoTrig : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusFreerun : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusPretrig : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusRunning : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusStopped : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusTriggered : [PwrSnsrLib.h](#)
 - PwrSnsrTriggerStatusWaiting : [PwrSnsrLib.h](#)
 - PwrSnsrUnitsdBm : [PwrSnsrLib.h](#)
 - PwrSnsrUnitsDBMV : [PwrSnsrLib.h](#)
 - PwrSnsrUnitsDBUV : [PwrSnsrLib.h](#)
 - PwrSnsrUnitsDBV : [PwrSnsrLib.h](#)
 - PwrSnsrUnitsvolts : [PwrSnsrLib.h](#)
 - PwrSnsrUnitswatts : [PwrSnsrLib.h](#)
-

Generated by  1.8.15

- P -

PulseInfo 16, 261

PwrSnsrLib.h 22, 264