

AMDTPowerProfileAPI

Generated by Doxygen 1.6.1

Mon Aug 3 02:55:22 2015

Contents

1	CodeXL Power Profiler API	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Module Documentation	9
5.1	Power Profiling	9
5.1.1	Detailed Description	11
5.1.2	Enumeration Type Documentation	11
5.1.2.1	AMDTPwrProfileMode	11
5.1.2.2	AMDTPwrDeviceType	12
5.1.2.3	AMDTPwrCategory	12
5.1.2.4	AMDTPwrAggregation	13
5.1.2.5	AMDTPwrUnit	13
5.1.2.6	AMDTPwrProfileState	13
5.1.2.7	AMDTPwrSampleValueOption	14
5.1.2.8	AMDTPwrApuPStates	14
5.1.3	Function Documentation	15
5.1.3.1	AMDTPwrProfileInitialize	15
5.1.3.2	AMDTPwrGetSystemTopology	15
5.1.3.3	AMDTPwrGetDeviceCounters	16

5.1.3.4	AMDTPwrGetCounterDesc	17
5.1.3.5	AMDTPwrEnableCounter	17
5.1.3.6	AMDTPwrDisableCounter	18
5.1.3.7	AMDTPwrEnableAllCounters	19
5.1.3.8	AMDTPwrGetMinimalTimerSamplingPeriod	19
5.1.3.9	AMDTPwrSetTimerSamplingPeriod	20
5.1.3.10	AMDTPwrStartProfiling	20
5.1.3.11	AMDTPwrStopProfiling	21
5.1.3.12	AMDTPwrPauseProfiling	21
5.1.3.13	AMDTPwrResumeProfiling	22
5.1.3.14	AMDTPwrGetProfilingState	22
5.1.3.15	AMDTPwrProfileClose	22
5.1.3.16	AMDTPwrSetSampleValueOption	23
5.1.3.17	AMDTPwrGetSampleValueOption	23
5.1.3.18	AMDTPwrReadAllEnabledCounters	24
5.1.3.19	AMDTPwrReadCounterHistogram	25
5.1.3.20	AMDTPwrGetTimerSamplingPeriod	25
5.1.3.21	AMDTPwrIsCounterEnabled	26
5.1.3.22	AMDTPwrGetNumEnabledCounters	26
5.1.3.23	AMDTPwrGetApuPstateInfo	27
5.1.3.24	AMDTPwrGetCounterHierarchy	27
6	Data Structure Documentation	29
6.1	AMDTPwrApuPstate Struct Reference	29
6.1.1	Detailed Description	29
6.1.2	Field Documentation	29
6.1.2.1	m_state	29
6.1.2.2	m_isBoosted	29
6.1.2.3	m_frequency	30
6.2	AMDTPwrApuPstateList Struct Reference	31
6.2.1	Detailed Description	31
6.2.2	Field Documentation	31
6.2.2.1	m_cnt	31
6.2.2.2	m_stateInfo	31

6.3	AMDTPwrCounterDesc Struct Reference	32
6.3.1	Detailed Description	32
6.3.2	Field Documentation	32
6.3.2.1	m_counterID	32
6.3.2.2	m_deviceId	32
6.3.2.3	m_name	33
6.3.2.4	m_description	33
6.3.2.5	m_category	33
6.3.2.6	m_aggregation	33
6.3.2.7	m_minValue	33
6.3.2.8	m_maxValue	33
6.3.2.9	m_units	33
6.4	AMDTPwrCounterHierarchy Struct Reference	34
6.4.1	Detailed Description	34
6.4.2	Field Documentation	34
6.4.2.1	m_counter	34
6.4.2.2	m_parent	34
6.4.2.3	m_childCnt	34
6.4.2.4	m_pChildList	34
6.5	AMDTPwrCounterValue Struct Reference	35
6.5.1	Detailed Description	35
6.5.2	Field Documentation	35
6.5.2.1	m_counterID	35
6.5.2.2	m_counterValue	35
6.6	AMDTPwrDevice Struct Reference	36
6.6.1	Detailed Description	36
6.6.2	Field Documentation	36
6.6.2.1	m_type	36
6.6.2.2	m_deviceID	36
6.6.2.3	m_pName	36
6.6.2.4	m_pDescription	36
6.6.2.5	m_isAccessible	37
6.6.2.6	m_pFirstChild	37
6.6.2.7	m_pNextDevice	37

6.7	AMDTPwrHistogram Struct Reference	38
6.7.1	Detailed Description	38
6.7.2	Field Documentation	38
6.7.2.1	m_counterId	38
6.7.2.2	m_numOfBins	38
6.7.2.3	m_pRange	38
6.7.2.4	m_pBins	38
6.8	AMDTPwrSample Struct Reference	39
6.8.1	Detailed Description	39
6.8.2	Field Documentation	39
6.8.2.1	m_systemTime	39
6.8.2.2	m_elapsedTimeMs	39
6.8.2.3	m_recordId	39
6.8.2.4	m_numOfValues	40
6.8.2.5	m_counterValues	40
6.9	AMDTPwrSystemTime Struct Reference	41
6.9.1	Detailed Description	41
6.9.2	Field Documentation	41
6.9.2.1	m_second	41
6.9.2.2	m_microSecond	41
7	File Documentation	43
7.1	AMDTDefinitions.h File Reference	43
7.1.1	Detailed Description	45
7.1.2	Define Documentation	45
7.1.2.1	AMDT_STATUS_OK	45
7.1.2.2	AMDT_ERROR_FAIL	45
7.1.2.3	AMDT_ERROR_INVALIDARG	45
7.1.2.4	AMDT_ERROR_OUTOFMEMORY	45
7.1.2.5	AMDT_ERROR_UNEXPECTED	45
7.1.2.6	AMDT_ERROR_ACCESSDENIED	45
7.1.2.7	AMDT_ERROR_HANDLE	46
7.1.2.8	AMDT_ERROR_ABORT	46
7.1.2.9	AMDT_ERROR_NOTIMPL	46

7.1.2.10	AMDT_ERROR_NOFILE	46
7.1.2.11	AMDT_ERROR_INVALIDPATH	46
7.1.2.12	AMDT_ERROR_INVALIDDATA	46
7.1.2.13	AMDT_ERROR_NOTAVAILABLE	46
7.1.2.14	AMDT_ERROR_NODATA	46
7.1.2.15	AMDT_ERROR_LOCKED	47
7.1.2.16	AMDT_ERROR_TIMEOUT	47
7.1.2.17	AMDT_STATUS_PENDING	47
7.1.2.18	AMDT_ERROR_NOTSUPPORTED	47
7.1.2.19	AMDT_ERROR_DRIVER_ALREADY_- INITIALIZED	47
7.1.2.20	AMDT_ERROR_DRIVER_UNAVAILABLE . . .	47
7.1.2.21	AMDT_WARN_SMU_DISABLED	47
7.1.2.22	AMDT_WARN_IGPU_DISABLED	48
7.1.2.23	AMDT_ERROR_DRIVER_UNINITIALIZED . . .	48
7.1.2.24	AMDT_ERROR_INVALID_DEVICEID	48
7.1.2.25	AMDT_ERROR_INVALID_COUNTERID	48
7.1.2.26	AMDT_ERROR_COUNTER_ALREADY_- ENABLED	48
7.1.2.27	AMDT_ERROR_NO_WRITE_PERMISSION . . .	48
7.1.2.28	AMDT_ERROR_COUNTER_NOT_ENABLED . .	48
7.1.2.29	AMDT_ERROR_TIMER_NOT_SET	49
7.1.2.30	AMDT_ERROR_PROFILE_DATAFILE_NOT_SET	49
7.1.2.31	AMDT_ERROR_PROFILE_ALREADY_STARTED	49
7.1.2.32	AMDT_ERROR_PROFILE_NOT_STARTED . . .	49
7.1.2.33	AMDT_ERROR_PROFILE_NOT_PAUSED	49
7.1.2.34	AMDT_ERROR_PROFILE_DATA_NOT_- AVAILABLE	49
7.1.2.35	AMDT_ERROR_PLATFORM_NOT_SUPPORTED	49
7.1.2.36	AMDT_ERROR_INTERNAL	50
7.1.2.37	AMDT_DRIVER_VERSION_MISMATCH	50
7.1.2.38	AMDT_ERROR_BIOS_VERSION_NOT_- SUPPORTED	50
7.1.2.39	AMDT_ERROR_PROFILE_ALREADY_- CONFIGURED	50

7.1.2.40	AMDT_ERROR_PROFILE_NOT_CONFIGURED	50
7.1.2.41	AMDT_ERROR_PROFILE_SESSION_EXISTS .	50
7.1.2.42	AMDT_ERROR_SMU_ACCESS_FAILED	50
7.1.2.43	AMDT_ERROR_COUNTERS_NOT_ENABLED .	51
7.1.2.44	AMDT_ERROR_PREVIOUS_SESSION_NOT_- CLOSED	51
7.1.2.45	AMDT_ERROR_COUNTER_NOHIERARCHY .	51
7.1.2.46	AMDT_ERROR_COUNTER_NOT_ACCESSIBLE	51
7.1.3	Typedef Documentation	51
7.1.3.1	AMDTResult	51
7.2	AMDTPowerProfileApi.h File Reference	52
7.2.1	Detailed Description	53
7.3	AMDTPowerProfileDataTypes.h File Reference	54
7.3.1	Detailed Description	55
7.3.2	Define Documentation	55
7.3.2.1	AMDT_PWR_ALL_DEVICES	55
7.3.2.2	AMDT_MAX_PSTATES	56
7.3.3	Typedef Documentation	56
7.3.3.1	AMDTPwrDeviceId	56
8	Example Documentation	57
8.1	CollectAllCounters.cpp	57

Chapter 1

CodeXL Power Profiler API

The AMDTPwrProfileAPI is a powerful library to help analyze the energy efficiency of systems based on AMD CPUs, APU's and Discrete GPU's.

This API:

- Provides counters to read the power, thermal and frequency characteristics of APU/dGPU and their subcomponents.
- Supports AMD APU's (Kaveri, Temash, Mullins, Carrizo), Discrete GPU's (Tonga, Iceland, Bonaire, Hawaii and other newer graphics cards)
- Supports AMD FirePro discrete GPU cards (W9100, W8100, W7100, W5100 and other newer graphics cards).
- Supports Microsoft Windows as a dynamically loaded library or as a static library.
- Supports Linux as a shared library or as a static library.
- Manages memory automatically - no allocation and free required.

Using this API, counter values can be read at regular sampling interval. Before any profiling done, the [AMDTPwrProfileInitialize\(\)](#) API must be called. When all the profiling is finished, the [AMDTPwrProfileClose\(\)](#) API must be called. Upon successful completion all the APIs will return AMDT_STATUS_OK, otherwise they return appropriate error codes.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Power Profiling	9
---------------------------	---

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

AMDTPwrApuPstate	29
AMDTPwrApuPstateList	31
AMDTPwrCounterDesc	32
AMDTPwrCounterHierarchy	34
AMDTPwrCounterValue	35
AMDTPwrDevice	36
AMDTPwrHistogram	38
AMDTPwrSample	39
AMDTPwrSystemTime	41

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

AMDDefinitions.h (Basic data type definitions and error codes used by the AMD CodeXL Power Profiler APIs)	43
AMDTPowerProfileApi.h (AMD Power Profiler APIs to configure, control and collect the power profile counters)	52
AMDTPowerProfileDataTypes.h (Data types and structure definitions used by AMD CodeXL Power Profiler APIs)	54

Chapter 5

Module Documentation

5.1 Power Profiling

AMDT Power Profiler APIs.

Data Structures

- struct [AMDTPwrDevice](#)
- struct [AMDTPwrCounterDesc](#)
- struct [AMDTPwrCounterValue](#)
- struct [AMDTPwrSystemTime](#)
- struct [AMDTPwrSample](#)
- struct [AMDTPwrApuPstate](#)
- struct [AMDTPwrApuPstateList](#)
- struct [AMDTPwrCounterHierarchy](#)
- struct [AMDTPwrHistogram](#)

Enumerations

- enum [AMDTPwrProfileMode](#) { [AMDTPWR_PROFILE_MODE_ONLINE](#) }
- enum [AMDTPwrDeviceType](#) {
 [AMDTPWR_DEVICE_SYSTEM](#), [AMDTPWR_DEVICE_PACKAGE](#),
 [AMDTPWR_DEVICE_CPU_COMPUTE_UNIT](#), [AMDTPWR_DEVICE_CPU_CORE](#),
 [AMDTPWR_DEVICE_INTERNAL_GPU](#), [AMDTPWR_DEVICE_EXTERNAL_GPU](#), [AMDTPWR_DEVICE_SVI2](#), [AMDTPWR_DEVICE_CNT](#) }
• enum [AMDTPwrCategory](#) {
 [AMDTPWR_CATEGORY_POWER](#), [AMDTPWR_CATEGORY_FREQUENCY](#),
 [AMDTPWR_CATEGORY_TEMPERATURE](#), [AMDTPWR_CATEGORY_VOLTAGE](#),

```

    AMDT_PWR_CATEGORY_CURRENT, AMDT_PWR_CATEGORY_DVFS,
    AMDT_PWR_CATEGORY_PROCESS, AMDT_PWR_CATEGORY_TIME,
    AMDT_PWR_CATEGORY_COUNT, AMDT_PWR_CATEGORY_CNT }
• enum AMDTPwrAggregation { AMDT_PWR_VALUE_SINGLE, AMDT_
  PWR_VALUE_CUMULATIVE,      AMDT_PWR_VALUE_HISTOGRAM,
  AMDT_PWR_VALUE_CNT }
• enum AMDTPwrUnit {
  AMDT_PWR_UNIT_TYPE_COUNT,      AMDT_PWR_UNIT_TYPE_
  PERCENT, AMDT_PWR_UNIT_TYPE_RATIO, AMDT_PWR_UNIT_
  TYPE_MILLI_SECOND,

  AMDT_PWR_UNIT_TYPE_JOULE,      AMDT_PWR_UNIT_TYPE_WATT,
  AMDT_PWR_UNIT_TYPE_VOLT,      AMDT_PWR_UNIT_TYPE_MILLI_
  AMPERE,

  AMDT_PWR_UNIT_TYPE_MEGA_HERTZ, AMDT_PWR_UNIT_TYPE_
  CENTIGRADE, AMDT_PWR_UNIT_TYPE_CNT }
• enum AMDTPwrProfileState {
  AMDT_PWR_PROFILE_STATE_UNINITIALIZED,      AMDT_PWR_
  PROFILE_STATE_IDLE,      AMDT_PWR_PROFILE_STATE_RUNNING,
  AMDT_PWR_PROFILE_STATE_PAUSED,

  AMDT_PWR_PROFILE_STATE_STOPPED,      AMDT_PWR_PROFILE_
  STATE_ABORTED, AMDT_PWR_PROFILE_STATE_CNT }
• enum AMDTSampleValueOption { AMDT_PWR_SAMPLE_VALUE_
  INSTANTANEOUS, AMDT_PWR_SAMPLE_VALUE_LIST, AMDT_PWR_
  SAMPLE_VALUE_AVERAGE,      AMDT_PWR_SAMPLE_VALUE_CNT
  }
• enum AMDTApuPStates {
  AMDT_PWR_PSTATE_PB0, AMDT_PWR_PSTATE_PB1, AMDT_PWR_
  PSTATE_PB2, AMDT_PWR_PSTATE_PB3,

  AMDT_PWR_PSTATE_PB4, AMDT_PWR_PSTATE_PB5, AMDT_PWR_
  PSTATE_PB6, AMDT_PWR_PSTATE_P0,

  AMDT_PWR_PSTATE_P1, AMDT_PWR_PSTATE_P2, AMDT_PWR_
  PSTATE_P3, AMDT_PWR_PSTATE_P4,

  AMDT_PWR_PSTATE_P5, AMDT_PWR_PSTATE_P6, AMDT_PWR_
  PSTATE_P7 }

```

Functions

- `AMDTResult AMDTPwrProfileInitialize (AMDTPwrProfileMode profile-Mode)`
- `AMDTResult AMDTPwrGetSystemTopology (AMDTPwrDevice **ppTopology)`
- `AMDTResult AMDTPwrGetDeviceCounters (AMDTPwrDeviceId deviceId, AMDTUInt32 *pNumCounters, AMDTPwrCounterDesc **ppCounterDescs)`
- `AMDTResult AMDTPwrGetCounterDesc (AMDTUInt32 counterId, AMDTPwrCounterDesc *pCounterDesc)`

- [AMDTReturn AMDTPwrEnableCounter](#) (AMDTUInt32 counterId)
- [AMDTReturn AMDTPwrDisableCounter](#) (AMDTUInt32 counterId)
- [AMDTReturn AMDTPwrEnableAllCounters](#) ()
- [AMDTReturn AMDTPwrGetMinimalTimerSamplingPeriod](#) (AMDTUInt32 *pIntervalMilliSec)
- [AMDTReturn AMDTPwrSetTimerSamplingPeriod](#) (AMDTUInt32 interval)
- [AMDTReturn AMDTPwrStartProfiling](#) ()
- [AMDTReturn AMDTPwrStopProfiling](#) ()
- [AMDTReturn AMDTPwrPauseProfiling](#) ()
- [AMDTReturn AMDTPwrResumeProfiling](#) ()
- [AMDTReturn AMDTPwrGetProfilingState](#) (AMDTPwrProfileState *pState)
- [AMDTReturn AMDTPwrProfileClose](#) ()
- [AMDTReturn AMDTPwrSetSampleValueOption](#) (AMDTSampleValueOption opt)
- [AMDTReturn AMDTPwrGetSampleValueOption](#) (AMDTSampleValueOption *pOpt)
- [AMDTReturn AMDTPwrReadAllEnabledCounters](#) (AMDTUInt32 *pNumOfSamples, AMDTPwrSample **ppData)
- [AMDTReturn AMDTPwrReadCounterHistogram](#) (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTPwrHistogram **ppData)
- [AMDTReturn AMDTPwrGetTimerSamplingPeriod](#) (AMDTUInt32 *pIntervalMilliSec)
- [AMDTReturn AMDTPwrIsCounterEnabled](#) (AMDTUInt32 counterId)
- [AMDTReturn AMDTPwrGetNumEnabledCounters](#) (AMDTUInt32 *pCount)
- [AMDTReturn AMDTPwrGetApuPstateInfo](#) (AMDTPwrApuPstateList *pList)
- [AMDTReturn AMDTPwrGetCounterHierarchy](#) (AMDTUInt32 counterId, AMDTPwrCounterHierarchy *pInfo)

5.1.1 Detailed Description

AMDT Power Profiler APIs.

5.1.2 Enumeration Type Documentation

5.1.2.1 enum AMDTPwrProfileMode

Following power profile modes are supported.

Enumerator:

AMDT_PWR_PROFILE_MODE_ONLINE Power profile mode is online

Definition at line 41 of file AMDTPowerProfileDataTypes.h.

5.1.2.2 enum AMDTDeviceType

Each package (processor node) and its sub-components and dGPUs are considered as devices here. Following are the various types of devices supported by power profiler.

Enumerator:

- AMDT_PWR_DEVICE_SYSTEM*** Dummy root node. All the top-level devices like CPU,APU,dGPU are its children
- AMDT_PWR_DEVICE_PACKAGE*** In a multi-node system, each node will be a separate package
- AMDT_PWR_DEVICE_CPU_COMPUTE_UNIT*** Each CPU Compute-Unit within a package
- AMDT_PWR_DEVICE_CPU_CORE*** Each CPU core within a CPU Compute-Unit
- AMDT_PWR_DEVICE_INTERNAL_GPU*** Integrated GPU within a AMD APU
- AMDT_PWR_DEVICE_EXTERNAL_GPU*** Each AMD dGPU connected in the system
- AMDT_PWR_DEVICE_SVI2*** Serial Voltage Interface 2
- AMDT_PWR_DEVICE_CNT*** Total device count

Definition at line 50 of file AMDTPowerProfileDataTypes.h.

5.1.2.3 enum AMDTPwrCategory

Following is the list of counter category supported by power profiler.

Enumerator:

- AMDT_PWR_CATEGORY_POWER*** Instantaneous power
- AMDT_PWR_CATEGORY_FREQUENCY*** Frequency
- AMDT_PWR_CATEGORY_TEMPERATURE*** Temperature in centigrade
- AMDT_PWR_CATEGORY_VOLTAGE*** Voltage
- AMDT_PWR_CATEGORY_CURRENT*** Current
- AMDT_PWR_CATEGORY_DVFS*** P-State, C-State
- AMDT_PWR_CATEGORY_PROCESS*** PID, TID
- AMDT_PWR_CATEGORY_TIME*** Time
- AMDT_PWR_CATEGORY_COUNT*** Generic count value
- AMDT_PWR_CATEGORY_CNT*** Total category count

Definition at line 65 of file AMDTPowerProfileDataTypes.h.

5.1.2.4 enum AMDTPwrAggregation

Following is the list of aggregation types supported by power profiler.

Enumerator:

AMDT_PWR_VALUE_SINGLE Single instantaneous value

AMDT_PWR_VALUE_CUMULATIVE Cumulative value

AMDT_PWR_VALUE_HISTOGRAM Histogram value

AMDT_PWR_VALUE_CNT Total power value

Definition at line 82 of file AMDTPowerProfileDataTypes.h.

5.1.2.5 enum AMDTPwrUnit

Various unit types for the output values for the counter types.

Enumerator:

AMDT_PWR_UNIT_TYPE_COUNT Count index

AMDT_PWR_UNIT_TYPE_PERCENT Percentage

AMDT_PWR_UNIT_TYPE_RATIO Ratio

AMDT_PWR_UNIT_TYPE_MILLI_SECOND Time in milli seconds

AMDT_PWR_UNIT_TYPE_JOULE Energy consumption

AMDT_PWR_UNIT_TYPE_WATT Power consumption

AMDT_PWR_UNIT_TYPE_VOLT Voltage

AMDT_PWR_UNIT_TYPE_MILLI_AMPERE Current

AMDT_PWR_UNIT_TYPE_MEGA_HERTZ Frequency type unit

AMDT_PWR_UNIT_TYPE_CENTIGRADE Temperature type unit

AMDT_PWR_UNIT_TYPE_CNT Total power unit

Definition at line 93 of file AMDTPowerProfileDataTypes.h.

5.1.2.6 enum AMDTPwrProfileState

States of Power profiler.

Enumerator:

AMDT_PWR_PROFILE_STATE_UNINITIALIZED Profiler is not initialized

AMDT_PWR_PROFILE_STATE_IDLE Profiler is initialized

AMDT_PWR_PROFILE_STATE_RUNNING Profiler is running

AMDT_PWR_PROFILE_STATE_PAUSED Profiler is paused

AMDT_PWR_PROFILE_STATE_STOPPED Profiler is Stopped
AMDT_PWR_PROFILE_STATE_ABORTED Profiler is aborted
AMDT_PWR_PROFILE_STATE_CNT Total number of profiler states

Definition at line 111 of file AMDTPowerProfileDataTypes.h.

5.1.2.7 enum AMDTSampleValueOption

Options to retrieve the profiled counter data using AMDTPwrReadAllEnabledCounters function

Enumerator:

AMDT_PWR_SAMPLE_VALUE_INSTANTANEOUS Default. The latest/instantaneous
AMDT_PWR_SAMPLE_VALUE_LIST List of sampled counter values
AMDT_PWR_SAMPLE_VALUE_AVERAGE Average of the sampled counter
AMDT_PWR_SAMPLE_VALUE_CNT Maximum Sample value count

Definition at line 125 of file AMDTPowerProfileDataTypes.h.

5.1.2.8 enum AMDTApuPStates

P-States can be either hardware or software P-States. Hardware P-States are also known as Boosted P-States. These are defined as AMDT_PWR_PSTATES_PBx. The Software P-States are defined as AMDT_PWR_PSTATES_Px, where x is the P-State number. Hardware(Boosted) P-States are not software visible.

Enumerator:

AMDT_PWR_PSTATE_PB0 Boosted P-State 0
AMDT_PWR_PSTATE_PB1 Boosted P-State 1
AMDT_PWR_PSTATE_PB2 Boosted P-State 2
AMDT_PWR_PSTATE_PB3 Boosted P-State 3
AMDT_PWR_PSTATE_PB4 Boosted P-State 4
AMDT_PWR_PSTATE_PB5 Boosted P-State 5
AMDT_PWR_PSTATE_PB6 Boosted P-State 6
AMDT_PWR_PSTATE_P0 Software P-State 0
AMDT_PWR_PSTATE_P1 Software P-State 1
AMDT_PWR_PSTATE_P2 Software P-State 2
AMDT_PWR_PSTATE_P3 Software P-State 3
AMDT_PWR_PSTATE_P4 Software P-State 4
AMDT_PWR_PSTATE_P5 Software P-State 5
AMDT_PWR_PSTATE_P6 Software P-State 6
AMDT_PWR_PSTATE_P7 Software P-State 7

Definition at line 139 of file AMDTPowerProfileDataTypes.h.

5.1.3 Function Documentation

5.1.3.1 `AMDTResult AMDTPwrProfileInitialize (AMDTPwrProfileMode profileMode)`

This API loads and initializes the AMDT Power Profile drivers. This API should be the first one to be called.

Parameters:

← *profileMode*,: Client should select any one of the predefined profile modes that are defined in [AMDTPwrProfileMode](#).

Returns:

The status of initialization request

Return values:

AMDT_STATUS_OK,: Success

AMDT_ERROR_INVALIDARG,: An invalid profileMode parameter was passed

AMDT_ERROR_DRIVER_UNAVAILABLE,: Driver not available

AMDT_ERROR_DRIVER_ALREADY_INITIALIZED,: Already initialized

AMDT_DRIVER_VERSION_MISMATCH,: Mismatch between the expected and installed driver versions

AMDT_ERROR_PLATFORM_NOT_SUPPORTED,: Platform not supported

AMDT_WARN_SMU_DISABLED,: SMU is disabled and hence power and thermal values provided by SMU will not be available

AMDT_WARN_IGPU_DISABLED,: Internal GPU is disabled

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed.

Examples:

[CollectAllCounters.cpp](#).

5.1.3.2 `AMDTResult AMDTPwrGetSystemTopology (AMDTPwrDevice ** ppTopology)`

This API provides device tree that represents the current system topology relevant to power profiler. The nodes (a processor package or a dGPU) and as well as their sub-components are considered as devices. Each device in the tree points to their siblings and children, if any.

Parameters:

→ *ppTopology*,: Device tree

Returns:

The status of system topology request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as ppTopology parameter

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_OUTOFMEMORY,: Failed to allocate required memory

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.3 AMDTResult AMDTPwrGetDeviceCounters (AMDTPwrDeviceId deviceId, AMDTUInt32 *pNumCounters, AMDTPwrCounterDesc **ppCounterDescs)

This API provides the list of supported counters for the given device id. If the device id is [AMDT_PWR_ALL_DEVICES](#), then counters for all the available devices will be returned. The pointer returned will be valid till the client calls [AMDTPwrProfileClose\(\)](#) function.

Parameters:

← **deviceId**,: The deviceId provided by [AMDTPwrGetSystemTopology\(\)](#) function or AMDT_PWR_ALL_DEVICES to represent all the devices returned by [AMDTPwrGetSystemTopology\(\)](#)

→ **pNumCounters**,: Number of counters supported by the device

→ **ppCounterDescs**,: Description of each counter supported by the device

Returns:

The status of device counter details request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as ppCounterDescs or pNumCounters parameters

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_INVALID_DEVICEID,: invalid deviceId parameter was passed

AMDT_ERROR_OUTOFMEMORY,: Failed to allocate required memory

AMDT_ERROR_FAIL,: An internal error occurred

Examples:

[CollectAllCounters.cpp](#).

5.1.3.4 **AMDTResult AMDTPwrGetCounterDesc** (AMDTUInt32 *counterId*, AMDTPwrCounterDesc * *pCounterDesc*)

This API provides the description for the given counter Index.

Parameters:

- ← *counterId*,: Counter index
- *pCounterDesc*,: Description of the counter which index is counterId

Returns:

The status of counter description request

Return values:

- AMDT_STATUS_OK**,: On Success
- AMDT_ERROR_INVALIDARG**,: NULL pointer was passed as pCounterDesc parameter
- AMDT_ERROR_DRIVER_UNINITIALIZED**,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful
- AMDT_ERROR_INVALID_COUNTERID**,: Invalid counterId parameter was passed
- AMDT_ERROR_FAIL**,: An internal error occurred

Examples:

[CollectAllCounters.cpp](#).

5.1.3.5 **AMDTResult AMDTPwrEnableCounter** (AMDTUInt32 *counterId*)

This API will enable the counter to be sampled. This API cannot be used once profile is started.

- If histogram/cumulative counters are enabled along with simple counters, then it is expected that the [AMDTPwrReadAllEnabledCounters\(\)](#) API is regularly called to read the simple counters value. Only then the values for histogram/cumulative counters will be aggregated and the [AMDTPwrReadCounterHistogram\(\)](#) API will return the correct values.
- If only the histogram/cumulative counters are enabled, calling [AMDTPwrReadCounterHistogram\(\)](#) is sufficient to get the values for the enabled histogram/cumulative counters.

Parameters:

- ← *counterId*,: Counter index

Returns:

The status of counter enable request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_INVALID_COUNTERID,: Invalid counterId parameter was passed

AMDT_ERROR_COUNTER_ALREADY_ENABLED,: Specified counter is already enabled

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Counters cannot be enabled on the fly when the profile is already started

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed

AMDT_ERROR_COUNTER_NOT_ACCESSIBLE,: Counter is not accessible

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.6 AMDTResult AMDTPwrDisableCounter (AMDTUInt32 counterId)

This API will disable the counter to be sampled from the active list. This API cannot be used once profile is started.

Parameters:

← **counterId,:** Counter index

Returns:

The status of counter disable request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_INVALID_COUNTERID,: Invalid counterId parameter was passed

AMDT_ERROR_COUNTER_NOT_ENABLED,: Specified counter is not enabled

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Counters cannot be disabled on the fly when the profile run is already started

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.7 AMDTResult AMDTPwrEnableAllCounters ()

This API will enable all the simple counters. This will NOT enable the histogram counters. This API cannot be used once profile is started.

Returns:

The status of enabling all the supported counters request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_COUNTER_ALREADY_ENABLED,: Some of the counters are already enabled

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Counters cannot be enabled on the fly when the profile is already started

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed

Examples:

[CollectAllCounters.cpp](#).

5.1.3.8 AMDTResult AMDTPwrGetMinimalTimerSamplingPeriod (AMDTUInt32 * *pIntervalMilliSec*)

This API provides the minimum sampling interval which can be set by the client.

Parameters:

→ *pIntervalMilliSec*,: The sampling interval in milli-second

Returns:

The status of retrieving the minimum supported sampling interval request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as *pIntervalMilliSec* parameter

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.9 AMDTResult AMDTPwrSetTimerSamplingPeriod (AMDTUInt32 *interval*)

This API will set the driver to periodically sample the counter values and store them in a buffer. This cannot be called once the profile run is started.

Parameters:

← *interval*,: sampling period in millisecond

Returns:

The status of sampling time set request

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: Invalid interval value was passed as IntervalMilliSec parameter

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Timer interval cannot be changed when the profile is already started

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed

AMDT_ERROR_FAIL,: An internal error occurred

Examples:

[CollectAllCounters.cpp](#).

5.1.3.10 AMDTResult AMDTPwrStartProfiling ()

This API will start the profiling and the driver will collect the data at regular interval specified by [AMDTPwrSetTimerSamplingPeriod\(\)](#). This has to be called after enabling the required counters by using [AMDTPwrEnableCounter\(\)](#) or [AMDTPwrEnableAllCounters\(\)](#).

Returns:

The status of starting the profile

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize](#) function was neither called nor successful

AMDT_ERROR_TIMER_NOT_SET,: Sampling timer was not set

AMDT_ERROR_COUNTERS_NOT_ENABLED,: No counters are enabled for collecting profile data

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Profile is already started

AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED,: Previous session was not closed

AMDT_ERROR_BIOS_VERSION_NOT_SUPPORTED,: BIOS needs to be upgraded

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_ACCESSDENIED,: Profiler is busy, currently not accessible

Examples:

[CollectAllCounters.cpp](#).

5.1.3.11 AMDTResult AMDTPwrStopProfiling ()

This APIs will stop the profiling run which was started by [AMDTPwrStartProfiling\(\)](#) function call.

Returns:

The status of stopping the profile

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_NOT_STARTED,: Profile is not started

AMDT_ERROR_FAIL,: An internal error occurred

Examples:

[CollectAllCounters.cpp](#).

5.1.3.12 AMDTResult AMDTPwrPauseProfiling ()

This API will pause the profiling. The driver and the backend will retain the profile configuration details provided by the client.

Returns:

The status of pausing the profile

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_NOT_STARTED,: Profile not started

5.1.3.13 AMDTResult AMDTPwrResumeProfiling ()

This API will resume the profiling which is in paused state.

Returns:

The status of resuming the profile

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_NOT_PAUSED,: Profile is not in paused state

5.1.3.14 AMDTResult AMDTPwrGetProfilingState (AMDTPwrProfileState * pState)

This API provides the current state of the profile.

Parameters:

→ ***pState*** Current profile state

Returns:

The status of getting the profile state

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as pState parameter

5.1.3.15 AMDTResult AMDTPwrProfileClose ()

This API will close the power profiler and unregister driver and cleanup all memory allocated during [AMDTPwrProfileInitialize\(\)](#).

Returns:

The status of closing the profiler

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

Examples:

[CollectAllCounters.cpp](#).

5.1.3.16 AMDTResult AMDTPwrSetSampleValueOption (AMDTSampleValueOption *opt*)

API to set the sample value options to be returned by the [AMDTPwrReadAllEnabled-Counters\(\)](#) function.

Parameters:

← *opt,:* One of the output value options defined in AMDTSampleValueOption

Returns:

The status of setting the output value option

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_INVALIDARG,: An invalid opt was specified as parameter

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_ALREADY_STARTED,: Cannot set the sample value option when the profile is running

5.1.3.17 AMDTResult AMDTPwrGetSampleValueOption (AMDTSampleValueOption **pOpt*)

API to get the sample value option set for the current profile session.

Parameters:

→ *pOpt,:* One of the output value options defined in AMDTSampleValueOption

Returns:

The status of setting the output value option

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_FAIL,: An internal error occurred

AMDT_ERROR_INVALIDARG,: An invalid opt was specified as parameter

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

5.1.3.18 AMDTResult AMDTPwrReadAllEnabledCounters (AMDTUInt32 * pNumOfSamples, AMDTPwrSample ** ppData)

API to read all the counters that are enabled. This will NOT read the histogram counters. This can return an array of { CounterID, Float-Value }. If there are no new samples, this API will return AMDTResult NO_NEW_DATA and pNumOfSamples will point to value of zero. If there are new samples, this API will return AMDT_STATUS_OK and pNumOfSamples will point to value greater than zero.

Parameters:

- **ppData**,: Processed profile data. No need to allocate or free the memory data is valid till we call this API next time
- **pNumOfSamples**,: Number of sample based on the [AMDTPwrSetSampleValueOption\(\)](#) set

Returns:

The status reading all enabled counters

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as pNumSamples or ppData parameters

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PROFILE_NOT_STARTED,: Profile is not started

AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE,: Profile data is not yet available

AMDT_ERROR_OUTOFMEMORY,: Memory not available

AMDT_ERROR_SMU_ACCESS_FAILED,: One of the configured SMU data access has problem it is advisable to stop the profiling session

AMDT_ERROR_FAIL,: An internal error occurred

Examples:

[CollectAllCounters.cpp](#).

5.1.3.19 AMDTResult AMDTPwrReadCounterHistogram (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTPwrHistogram ** ppData)

API to read one of the derived counters generate histograms from the raw counter values. Since the histogram may contain multiple entries and according to the counter values, a derived histogram counter type specific will be used to provide the output data.

Parameters:

- ← *counterId*,: Histogram type counter id
- *pNumEntries*,: Number of entries in the histogram
- *ppData*,: Compute histogram data for the given counter id

Returns:

The status of reading histogram data

Return values:

- AMDT_STATUS_OK*,: On Success
- AMDT_ERROR_INVALIDARG*,: NULL pointer was passed as pNumEntries or ppData parameters
- AMDT_ERROR_DRIVER_UNINITIALIZED*,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful
- AMDT_ERROR_INVALID_COUNTERID*,: An invalid counterId was passed
- AMDT_ERROR_PROFILE_NOT_STARTED*,: Profile is not started
- AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE*,: Profile data is not yet available
- AMDT_ERROR_OUTOFMEMORY*,: Memory not available
- AMDT_ERROR_FAIL*,: An internal error occurred

5.1.3.20 AMDTResult AMDTPwrGetTimerSamplingPeriod (AMDTUInt32 * pIntervalMilliSec)

This API will get the timer sampling period at which the samples are collected by the driver.

Parameters:

- *pIntervalMilliSec*,: sampling period in millisecond

Returns:

The status of the get sampling interval request

Return values:

AMDT_STATUS_OK,: On Success
AMDT_ERROR_INVALIDARG,: NULL pointer was passed as pIntervalMilliSec parameter
AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful
AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.21 AMDTResult AMDTPwrIsCounterEnabled (AMDTUInt32 *counterId*)

This query API is to check whether a counter is enabled for profiling or not.

Parameters:

← *counterId*,: Counter index

Returns:

The status of query request.

Return values:

AMDT_STATUS_OK,: On Success; Counter is enabled
AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful
AMDT_ERROR_INVALID_COUNTERID,: An invalid counterId was passed
AMDT_ERROR_COUNTER_NOT_ENABLED,: Counter is not enabled already
AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.22 AMDTResult AMDTPwrGetNumEnabledCounters (AMDTUInt32 * *pCount*)

This query API is to get the number of counters that are enabled for profiling.

Parameters:

→ *pCount*,: Number of enabled counters

Returns:

The status of query request

Return values:

AMDT_STATUS_OK,: On Success; Counter is enabled

AMDT_ERROR_INVALIDARG,: NULL pointer is passed as an argument

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.23 AMDTResult AMDTPwrGetApuPstateInfo (AMDTPwrApuPstateList * *pList*)

API to get the list of pstate supported by the target APU, where power profile is running. List contains both hardware and software P-States with their corresponding frequencies.

Parameters:

→ *pList*,: List of P-States

Returns:

The status reading the pstate list for the platform

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as argument

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#) function was neither called nor successful

AMDT_ERROR_PLATFORM_NOT_SUPPORTED,: Platform not supported

AMDT_ERROR_FAIL,: An internal error occurred

5.1.3.24 AMDTResult AMDTPwrGetCounterHierarchy (AMDTUInt32 *counterId*, AMDTPwrCounterHierarchy * *pInfo*)

This API provides the relationship with other counters for the given counter id. For the given counter id, this API provides the parent counter and as well the child counters list.

Parameters:

← *counterId*,: The counter id for which the dependent counters information is requested

→ *pInfo*,: Provides hierarchical relationship for the given counterId

Returns:

The status retrieving hierarchical information for the given counters

Return values:

AMDT_STATUS_OK,: On Success

AMDT_ERROR_INVALIDARG,: NULL pointer was passed as argument

AMDT_ERROR_DRIVER_UNINITIALIZED,: [AMDTPwrProfileInitialize\(\)](#)
function was neither called nor successful

AMDT_ERROR_INVALID_COUNTERID,: Invalid counterId parameter was
passed

AMDT_ERROR_COUNTER_NOHIERARCHY,: Counter does not have any hi-
erarchical relationship

AMDT_ERROR_FAIL,: An internal error occurred

Chapter 6

Data Structure Documentation

6.1 AMDTPwrApuPstate Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- [AMDTApuPStates m_state](#)
- bool [m_isBoosted](#)
- AMDTUInt32 [m_frequency](#)

6.1.1 Detailed Description

Provides various P-States and their corresponding frequencies.

Definition at line 227 of file AMDTPowerProfileDataTypes.h.

6.1.2 Field Documentation

6.1.2.1 AMDTApuPStates m_state

P-State number

Definition at line 229 of file AMDTPowerProfileDataTypes.h.

6.1.2.2 bool m_isBoosted

Boosted P-State flag

Definition at line 230 of file AMDTPowerProfileDataTypes.h.

6.1.2.3 AMDTUInt32 m_frequency

P-State frequency

Definition at line 231 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.2 AMDTPwrApuPstateList Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt32 [m_cnt](#)
- [AMDTPwrApuPstate m_stateInfo](#) [AMDT_MAX_PSTATES]

6.2.1 Detailed Description

List of the supported APU P-States details

Definition at line 237 of file AMDTPowerProfileDataTypes.h.

6.2.2 Field Documentation

6.2.2.1 AMDTUInt32 m_cnt

Number of P-States

Definition at line 239 of file AMDTPowerProfileDataTypes.h.

6.2.2.2 AMDTPwrApuPstate m_stateInfo[AMDT_MAX_PSTATES]

P-States list

Definition at line 240 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.3 AMDTPwrCounterDesc Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt32 [m_counterID](#)
- AMDTUInt32 [m_deviceId](#)
- char * [m_name](#)
- char * [m_description](#)
- [AMDTPwrCategory](#) [m_category](#)
- [AMDTPwrAggregation](#) [m_aggregation](#)
- AMDTFloat64 [m_minValue](#)
- AMDTFloat64 [m_maxValue](#)
- [AMDTPwrUnit](#) [m_units](#)

6.3.1 Detailed Description

Details of a supported power counter and its associated device. Following counter types are supported:

- Simple Counters has [m_aggregation](#) type as [AMDT_PWR_VALUE_SINGLE](#).
- Histogram Counters has [m_aggregation](#) type as [AMDT_PWR_VALUE_HISTOGRAM](#).
- Cumulative Counters has [m_aggregation](#) type as [AMDT_PWR_VALUE_CUMULATIVE](#).

Examples:

[CollectAllCounters.cpp](#).

Definition at line 181 of file [AMDTPowerProfileDataTypes.h](#).

6.3.2 Field Documentation

6.3.2.1 AMDTUInt32 m_counterID

Counter index

Definition at line 183 of file [AMDTPowerProfileDataTypes.h](#).

6.3.2.2 AMDTUInt32 m_deviceId

Device Id

Definition at line 184 of file [AMDTPowerProfileDataTypes.h](#).

6.3.2.3 char* m_name

Name of the counter

Examples:

[CollectAllCounters.cpp](#).

Definition at line 185 of file AMDTPowerProfileDataTypes.h.

6.3.2.4 char* m_description

Description of the counter

Definition at line 186 of file AMDTPowerProfileDataTypes.h.

6.3.2.5 AMDTPwrCategory m_category

Power/Freq/Temperature

Definition at line 187 of file AMDTPowerProfileDataTypes.h.

6.3.2.6 AMDTPwrAggregation m_aggregation

Single/Histogram/Cumulative

Definition at line 188 of file AMDTPowerProfileDataTypes.h.

6.3.2.7 AMDTFloat64 m_minValue

Minimum possible counter value

Definition at line 189 of file AMDTPowerProfileDataTypes.h.

6.3.2.8 AMDTFloat64 m_maxValue

Maximum possible counter value

Definition at line 190 of file AMDTPowerProfileDataTypes.h.

6.3.2.9 AMDTPwrUnit m_units

Seconds/MHz/Joules/Watts/Volt/Ampere

Definition at line 191 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.4 AMDTPwrCounterHierarchy Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt32 [m_counter](#)
- AMDTUInt32 [m_parent](#)
- AMDTUInt32 [m_childCnt](#)
- AMDTUInt32 * [m_pChildList](#)

6.4.1 Detailed Description

Provides hierarchical relationship details of a power counter. Both the parent and children counter details will be provided.

Definition at line 247 of file AMDTPowerProfileDataTypes.h.

6.4.2 Field Documentation

6.4.2.1 AMDTUInt32 m_counter

Counter Id

Definition at line 249 of file AMDTPowerProfileDataTypes.h.

6.4.2.2 AMDTUInt32 m_parent

Parent counter Id

Definition at line 250 of file AMDTPowerProfileDataTypes.h.

6.4.2.3 AMDTUInt32 m_childCnt

Number of child counters

Definition at line 251 of file AMDTPowerProfileDataTypes.h.

6.4.2.4 AMDTUInt32* m_pChildList

List of child counters

Definition at line 252 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.5 AMDTPwrCounterValue Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt32 [m_counterID](#)
- AMDTFloat32 [m_counterValue](#)

6.5.1 Detailed Description

Structure represents a counter ID and its value

Definition at line 197 of file AMDTPowerProfileDataTypes.h.

6.5.2 Field Documentation

6.5.2.1 AMDTUInt32 m_counterID

Counter index

Examples:

[CollectAllCounters.cpp](#).

Definition at line 199 of file AMDTPowerProfileDataTypes.h.

6.5.2.2 AMDTFloat32 m_counterValue

Counter value

Examples:

[CollectAllCounters.cpp](#).

Definition at line 200 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.6 AMDTPwrDevice Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- [AMDTPwrDeviceType m_type](#)
- [AMDTPwrDeviceId m_deviceID](#)
- [char * m_pName](#)
- [char * m_pDescription](#)
- [bool m_isAccessible](#)
- [AMDTPwrDevice * m_pFirstChild](#)
- [AMDTPwrDevice * m_pNextDevice](#)

6.6.1 Detailed Description

Following structure represents the device tree of the target system. Nodes will be available for components for which power counters are supported. Following are such components - AMD APUs and its subcomponents like CPU Compute-units, CPU Cores, integrated GPUs & AMD discrete GPUs.

Definition at line 163 of file AMDTPowerProfileDataTypes.h.

6.6.2 Field Documentation

6.6.2.1 AMDTPwrDeviceType m_type

Device type- compute unit/Core/ package/ dGPU

Definition at line 165 of file AMDTPowerProfileDataTypes.h.

6.6.2.2 AMDTPwrDeviceId m_deviceID

Device Id

Definition at line 166 of file AMDTPowerProfileDataTypes.h.

6.6.2.3 char* m_pName

Name of the device

Definition at line 167 of file AMDTPowerProfileDataTypes.h.

6.6.2.4 char* m_pDescription

Description about the device

Definition at line 168 of file AMDTPowerProfileDataTypes.h.

6.6.2.5 bool m_isAccessible

If counters are accessible

Definition at line 169 of file AMDTPowerProfileDataTypes.h.

6.6.2.6 AMDTPwrDevice* m_pFirstChild

Points to the sub-devices of this device

Definition at line 170 of file AMDTPowerProfileDataTypes.h.

6.6.2.7 AMDTPwrDevice* m_pNextDevice

Points to the next device at the same hierarchy

Definition at line 171 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.7 AMDTPwrHistogram Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt32 [m_counterId](#)
- AMDTUInt32 [m_numOfBins](#)
- AMDTFloat32 * [m_pRange](#)
- AMDTFloat32 * [m_pBins](#)

6.7.1 Detailed Description

Represents a generic histogram.

Definition at line 258 of file AMDTPowerProfileDataTypes.h.

6.7.2 Field Documentation

6.7.2.1 AMDTUInt32 m_counterId

Counter being aggregated

Definition at line 260 of file AMDTPowerProfileDataTypes.h.

6.7.2.2 AMDTUInt32 m_numOfBins

This is the number of histogram bins

Definition at line 261 of file AMDTPowerProfileDataTypes.h.

6.7.2.3 AMDTFloat32* m_pRange

The ranges of the bins are stored in an array of $n + 1$ elements pointed to by range

Definition at line 262 of file AMDTPowerProfileDataTypes.h.

6.7.2.4 AMDTFloat32* m_pBins

The counts for each bin are stored in an array of n elements pointed to by bin

Definition at line 263 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.8 AMDTPwrSample Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- [AMDTPwrSystemTime m_systemTime](#)
- [AMDTUInt64 m_elapsedTimeMs](#)
- [AMDTUInt64 m_recordId](#)
- [AMDTUInt32 m_numOfValues](#)
- [AMDTPwrCounterValue * m_counterValues](#)

6.8.1 Detailed Description

Output sample with timestamp and the counter values for all the enabled counters.

Examples:

[CollectAllCounters.cpp](#).

Definition at line 215 of file AMDTPowerProfileDataTypes.h.

6.8.2 Field Documentation

6.8.2.1 AMDTPwrSystemTime m_systemTime

Start time of Profiling

Examples:

[CollectAllCounters.cpp](#).

Definition at line 217 of file AMDTPowerProfileDataTypes.h.

6.8.2.2 AMDTUInt64 m_elapsedTimeMs

Elapsed time in milliseconds - relative to the start time of the profile

Definition at line 218 of file AMDTPowerProfileDataTypes.h.

6.8.2.3 AMDTUInt64 m_recordId

Record id

Definition at line 219 of file AMDTPowerProfileDataTypes.h.

6.8.2.4 AMDTUInt32 m_numOfValues

Number of counter values available

Examples:

[CollectAllCounters.cpp](#).

Definition at line 220 of file AMDTPowerProfileDataTypes.h.

6.8.2.5 AMDTPwrCounterValue* m_counterValues

list of counter values

Examples:

[CollectAllCounters.cpp](#).

Definition at line 221 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

6.9 AMDTPwrSystemTime Struct Reference

```
#include <AMDTPowerProfileDataTypes.h>
```

Data Fields

- AMDTUInt64 [m_second](#)
- AMDTUInt64 [m_microSecond](#)

6.9.1 Detailed Description

This structure represents the system time in second and milliseconds

Definition at line 206 of file AMDTPowerProfileDataTypes.h.

6.9.2 Field Documentation

6.9.2.1 AMDTUInt64 m_second

Seconds

Examples:

[CollectAllCounters.cpp](#).

Definition at line 208 of file AMDTPowerProfileDataTypes.h.

6.9.2.2 AMDTUInt64 m_microSecond

Milliseconds

Examples:

[CollectAllCounters.cpp](#).

Definition at line 209 of file AMDTPowerProfileDataTypes.h.

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

- [AMDTPowerProfileDataTypes.h](#)

Chapter 7

File Documentation

7.1 AMDTDefinitions.h File Reference

Basic data type definitions and error codes used by the AMD CodeXL Power Profiler APIs. `#include <limits.h>`

Defines

- `#define AMDT_STATUS_OK AMDTResult(0)`
- `#define AMDT_ERROR_FAIL AMDTResult(0x80004005)`
- `#define AMDT_ERROR_INVALIDARG AMDTResult(0x80070057)`
- `#define AMDT_ERROR_OUTOFMEMORY AMDTResult(0x8007000E)`
- `#define AMDT_ERROR_UNEXPECTED AMDTResult(0x8000FFFF)`
- `#define AMDT_ERROR_ACCESSDENIED AMDTResult(0x80070005)`
- `#define AMDT_ERROR_HANDLE AMDTResult(0x80070006)`
- `#define AMDT_ERROR_ABORT AMDTResult(0x80004004)`
- `#define AMDT_ERROR_NOTIMPL AMDTResult(0x80004001)`
- `#define AMDT_ERROR_NOFILE AMDTResult(0x80070002)`
- `#define AMDT_ERROR_INVALIDPATH AMDTResult(0x80070003)`
- `#define AMDT_ERROR_INVALIDDATA AMDTResult(0x8007000D)`
- `#define AMDT_ERROR_NOTAVAILABLE AMDTResult(0x80075006)`
- `#define AMDT_ERROR_NODATA AMDTResult(0x800700E8)`
- `#define AMDT_ERROR_LOCKED AMDTResult(0x80070021)`
- `#define AMDT_ERROR_TIMEOUT AMDTResult(0x800705B4)`
- `#define AMDT_STATUS_PENDING AMDTResult(0x8000000A)`
- `#define AMDT_ERROR_NOTSUPPORTED AMDTResult(0x8000FFFE)`
- `#define AMDT_ERROR_DRIVER_ALREADY_INITIALIZED AMDTResult(0x80080001)`
- `#define AMDT_ERROR_DRIVER_UNAVAILABLE AMDTResult(0x80080002)`
- `#define AMDT_WARN_SMU_DISABLED AMDTResult(0x80080003)`

- #define `AMDT_WARN_IGPU_DISABLED` `AMDTRResult(0x80080004)`
- #define `AMDT_ERROR_DRIVER_UNINITIALIZED` `AMDTRResult(0x80080005)`
- #define `AMDT_ERROR_INVALID_DEVICEID` `AMDTRResult(0x80080006)`
- #define `AMDT_ERROR_INVALID_COUNTERID` `AMDTRResult(0x80080007)`
- #define `AMDT_ERROR_COUNTER_ALREADY_ENABLED` `AMDTRResult(0x80080008)`
- #define `AMDT_ERROR_NO_WRITE_PERMISSION` `AMDTRResult(0x80080009)`
- #define `AMDT_ERROR_COUNTER_NOT_ENABLED` `AMDTRResult(0x8008000A)`
- #define `AMDT_ERROR_TIMER_NOT_SET` `AMDTRResult(0x8008000B)`
- #define `AMDT_ERROR_PROFILE_DATAFILE_NOT_SET` `AMDTRResult(0x8008000C)`
- #define `AMDT_ERROR_PROFILE_ALREADY_STARTED` `AMDTRResult(0x8008000D)`
- #define `AMDT_ERROR_PROFILE_NOT_STARTED` `AMDTRResult(0x8008000E)`
- #define `AMDT_ERROR_PROFILE_NOT_PAUSED` `AMDTRResult(0x8008000F)`
- #define `AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE` `AMDTRResult(0x80080010)`
- #define `AMDT_ERROR_PLATFORM_NOT_SUPPORTED` `AMDTRResult(0x80080011)`
- #define `AMDT_ERROR_INTERNAL` `AMDTRResult(0x80080012)`
- #define `AMDT_DRIVER_VERSION_MISMATCH` `AMDTRResult(0x80080013)`
- #define `AMDT_ERROR_BIOS_VERSION_NOT_SUPPORTED` `AMDTRResult(0x80080014)`
- #define `AMDT_ERROR_PROFILE_ALREADY_CONFIGURED` `AMDTRResult(0x80080015)`
- #define `AMDT_ERROR_PROFILE_NOT_CONFIGURED` `AMDTRResult(0x80080016)`
- #define `AMDT_ERROR_PROFILE_SESSION_EXISTS` `AMDTRResult(0x80080017)`
- #define `AMDT_ERROR_SMU_ACCESS_FAILED` `AMDTRResult(0x80080018)`
- #define `AMDT_ERROR_COUNTERS_NOT_ENABLED` `AMDTRResult(0x80080019)`
- #define `AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED` `AMDTRResult(0x80080020)`
- #define `AMDT_ERROR_COUNTER_NOHIERARCHY` `AMDTRResult(0x80080021)`
- #define `AMDT_ERROR_COUNTER_NOT_ACCESSIBLE` `AMDTRResult(0x80080022)`

Typedefs

- typedef unsigned int [AMDTResult](#)

7.1.1 Detailed Description

Basic data type definitions and error codes used by the AMD CodeXL Power Profiler APIs.

Definition in file [AMDTDefinitions.h](#).

7.1.2 Define Documentation

7.1.2.1 `#define AMDT_STATUS_OK AMDTResult(0)`

Returned on success

Definition at line 76 of file AMDTDefinitions.h.

7.1.2.2 `#define AMDT_ERROR_FAIL AMDTResult(0x80004005)`

An internal error occurred.

Definition at line 80 of file AMDTDefinitions.h.

7.1.2.3 `#define AMDT_ERROR_INVALIDARG AMDTResult(0x80070057)`

Invalid argument is passed.

Definition at line 84 of file AMDTDefinitions.h.

7.1.2.4 `#define AMDT_ERROR_OUTOFMEMORY AMDTResult(0x8007000E)`

Memory allocation failed.

Definition at line 88 of file AMDTDefinitions.h.

7.1.2.5 `#define AMDT_ERROR_UNEXPECTED AMDTResult(0x8000FFFF)`

An unexpected error occurred.

Definition at line 92 of file AMDTDefinitions.h.

7.1.2.6 `#define AMDT_ERROR_ACCESSDENIED AMDTResult(0x80070005)`

Profiler not available

Definition at line 96 of file AMDTDefinitions.h.

7.1.2.7 #define AMDT_ERROR_HANDLE AMDTResult(0x80070006)

Invalid handler is passed

Definition at line 100 of file AMDTDefinitions.h.

7.1.2.8 #define AMDT_ERROR_ABORT AMDTResult(0x80004004)

Profiler aborted due to an internal error

Definition at line 104 of file AMDTDefinitions.h.

7.1.2.9 #define AMDT_ERROR_NOTIMPL AMDTResult(0x80004001)

Requested profiler functionality is not yet implemented.

Definition at line 108 of file AMDTDefinitions.h.

7.1.2.10 #define AMDT_ERROR_NOFILE AMDTResult(0x80070002)

File not found.

Definition at line 112 of file AMDTDefinitions.h.

7.1.2.11 #define AMDT_ERROR_INVALIDPATH AMDTResult(0x80070003)

Invalid file path specified.

Definition at line 116 of file AMDTDefinitions.h.

7.1.2.12 #define AMDT_ERROR_INVALIDDATA AMDTResult(0x8007000D)

Invalid data is passed as a parameter.

Definition at line 120 of file AMDTDefinitions.h.

7.1.2.13 #define AMDT_ERROR_NOTAVAILABLE AMDTResult(0x80075006)

Requested functionality or data is not yet available.

Definition at line 124 of file AMDTDefinitions.h.

7.1.2.14 #define AMDT_ERROR_NODATA AMDTResult(0x800700E8)

No profile data is available.

Definition at line 128 of file AMDTDefinitions.h.

7.1.2.15 **#define AMDT_ERROR_LOCKED AMDTResult(0x80070021)**

Already locked.

Definition at line 132 of file AMDTDefinitions.h.

7.1.2.16 **#define AMDT_ERROR_TIMEOUT AMDTResult(0x800705B4)**

Timeout.

Definition at line 136 of file AMDTDefinitions.h.

7.1.2.17 **#define AMDT_STATUS_PENDING AMDTResult(0x8000000A)**

Profiler is currently active and the requested action is pending.

Definition at line 140 of file AMDTDefinitions.h.

7.1.2.18 **#define AMDT_ERROR_NOTSUPPORTED AMDTResult(0x8000FFFE)**

The requested functionality is not supported

Definition at line 144 of file AMDTDefinitions.h.

7.1.2.19 **#define AMDT_ERROR_DRIVER_ALREADY_INITIALIZED AMDTResult(0x80080001)**

Profiler is already initialized.

Definition at line 148 of file AMDTDefinitions.h.

7.1.2.20 **#define AMDT_ERROR_DRIVER_UNAVAILABLE AMDTResult(0x80080002)**

Profile driver is not available.

Definition at line 152 of file AMDTDefinitions.h.

7.1.2.21 **#define AMDT_WARN_SMU_DISABLED AMDTResult(0x80080003)**

SMU is disabled.

Definition at line 156 of file AMDTDefinitions.h.

7.1.2.22 #define AMDT_WARN_IGPU_DISABLED AMDTResult(0x80080004)

Internal GPU is disabled.

Definition at line 160 of file AMDTDefinitions.h.

7.1.2.23 #define AMDT_ERROR_DRIVER_UNINITIALIZED AMDTResult(0x80080005)

Driver is not yet initialized.

Definition at line 164 of file AMDTDefinitions.h.

7.1.2.24 #define AMDT_ERROR_INVALID_DEVICEID AMDTResult(0x80080006)

Invalid device ID is passed as a parameter.

Definition at line 168 of file AMDTDefinitions.h.

7.1.2.25 #define AMDT_ERROR_INVALID_COUNTERID AMDTResult(0x80080007)

Invalid profile counter id is passes as a parameter.

Definition at line 172 of file AMDTDefinitions.h.

7.1.2.26 #define AMDT_ERROR_COUNTER_ALREADY_ENABLED AMDTResult(0x80080008)

Specified counter ID is already enabled.

Definition at line 176 of file AMDTDefinitions.h.

7.1.2.27 #define AMDT_ERROR_NO_WRITE_PERMISSION AMDTResult(0x80080009)

No write permission to create the specified profile data file.

Definition at line 180 of file AMDTDefinitions.h.

7.1.2.28 #define AMDT_ERROR_COUNTER_NOT_ENABLED AMDTResult(0x8008000A)

Specified counter ID is not enabled.

Definition at line 184 of file AMDTDefinitions.h.

**7.1.2.29 #define AMDT_ERROR_TIMER_NOT_-
SET AMDTResult(0x8008000B)**

Sampling timer is not set.

Definition at line 188 of file AMDTDefinitions.h.

**7.1.2.30 #define AMDT_ERROR_PROFILE_DATAFILE_NOT_-
SET AMDTResult(0x8008000C)**

Profile data file is not set.

Definition at line 192 of file AMDTDefinitions.h.

**7.1.2.31 #define AMDT_ERROR_PROFILE_ALREADY_-
STARTED AMDTResult(0x8008000D)**

Profile was already started.

Definition at line 196 of file AMDTDefinitions.h.

**7.1.2.32 #define AMDT_ERROR_PROFILE_NOT_-
STARTED AMDTResult(0x8008000E)**

Profile was not started.

Definition at line 200 of file AMDTDefinitions.h.

**7.1.2.33 #define AMDT_ERROR_PROFILE_NOT_-
PAUSED AMDTResult(0x8008000F)**

Profile is not in paused state.

Definition at line 204 of file AMDTDefinitions.h.

**7.1.2.34 #define AMDT_ERROR_PROFILE_DATA_NOT_-
AVAILABLE AMDTResult(0x80080010)**

Profile data is not yet available.

Definition at line 208 of file AMDTDefinitions.h.

**7.1.2.35 #define AMDT_ERROR_PLATFORM_NOT_-
SUPPORTED AMDTResult(0x80080011)**

This HW platform is not supported.

Definition at line 212 of file AMDTDefinitions.h.

7.1.2.36 #define AMDT_ERROR_INTERNAL AMDTResult(0x80080012)

An Internal error occurred.

Definition at line 216 of file AMDTDefinitions.h.

7.1.2.37 #define AMDT_DRIVER_VERSION_MISMATCH AMDTResult(0x80080013)

Mismatch between the expected and installed driver versions.

Definition at line 220 of file AMDTDefinitions.h.

7.1.2.38 #define AMDT_ERROR_BIOS_VERSION_NOT_SUPPORTED AMDTResult(0x80080014)

Bios needs to be upgraded in the system.

Definition at line 224 of file AMDTDefinitions.h.

7.1.2.39 #define AMDT_ERROR_PROFILE_ALREADY_CONFIGURED AMDTResult(0x80080015)

Profile is already configured.

Definition at line 228 of file AMDTDefinitions.h.

7.1.2.40 #define AMDT_ERROR_PROFILE_NOT_CONFIGURED AMDTResult(0x80080016)

Profile is not yet configured.

Definition at line 232 of file AMDTDefinitions.h.

7.1.2.41 #define AMDT_ERROR_PROFILE_SESSION_EXISTS AMDTResult(0x80080017)

Profile session already exists.

Definition at line 236 of file AMDTDefinitions.h.

7.1.2.42 #define AMDT_ERROR_SMU_ACCESS_FAILED AMDTResult(0x80080018)

Could not access the configured profile counter due to access failure.

Definition at line 240 of file AMDTDefinitions.h.

7.1.2.43 #define AMDT_ERROR_COUNTERS_NOT_ENABLED AMDTResult(0x80080019)

Could not start the profile session as counters are not enabled.

Definition at line 244 of file AMDTDefinitions.h.

7.1.2.44 #define AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED AMDTResult(0x80080020)

Previous profile session was not closed.

Definition at line 248 of file AMDTDefinitions.h.

7.1.2.45 #define AMDT_ERROR_COUNTER_NOHIERARCHY AMDTResult(0x80080021)

Counter does not have any hierarchical relationship

Definition at line 252 of file AMDTDefinitions.h.

7.1.2.46 #define AMDT_ERROR_COUNTER_NOT_ACCESSIBLE AMDTResult(0x80080022)

Counter is not accessible

Definition at line 256 of file AMDTDefinitions.h.

7.1.3 Typedef Documentation**7.1.3.1 typedef unsigned int AMDTResult**

Examples:

[CollectAllCounters.cpp](#).

Definition at line 72 of file AMDTDefinitions.h.

7.2 AMDTPowerProfileApi.h File Reference

AMD Power Profiler APIs to configure, control and collect the power profile counters.

```
#include <AMDDefinitions.h>
```

```
#include <AMDTPowerProfileDataTypes.h>
```

Functions

- [AMDTRResult](#) [AMDTPwrProfileInitialize](#) ([AMDTPwrProfileMode](#) profile-Mode)
- [AMDTRResult](#) [AMDTPwrGetSystemTopology](#) ([AMDTPwrDevice](#) **ppTopology)
- [AMDTRResult](#) [AMDTPwrGetDeviceCounters](#) ([AMDTPwrDeviceId](#) deviceId, [AMDТУInt32](#) *pNumCounters, [AMDTPwrCounterDesc](#) **ppCounterDescs)
- [AMDTRResult](#) [AMDTPwrGetCounterDesc](#) ([AMDТУInt32](#) counterId, [AMDTPwrCounterDesc](#) *pCounterDesc)
- [AMDTRResult](#) [AMDTPwrEnableCounter](#) ([AMDТУInt32](#) counterId)
- [AMDTRResult](#) [AMDTPwrDisableCounter](#) ([AMDТУInt32](#) counterId)
- [AMDTRResult](#) [AMDTPwrEnableAllCounters](#) ()
- [AMDTRResult](#) [AMDTPwrGetMinimalTimerSamplingPeriod](#) ([AMDТУInt32](#) *pIntervalMilliSec)
- [AMDTRResult](#) [AMDTPwrSetTimerSamplingPeriod](#) ([AMDТУInt32](#) interval)
- [AMDTRResult](#) [AMDTPwrStartProfiling](#) ()
- [AMDTRResult](#) [AMDTPwrStopProfiling](#) ()
- [AMDTRResult](#) [AMDTPwrPauseProfiling](#) ()
- [AMDTRResult](#) [AMDTPwrResumeProfiling](#) ()
- [AMDTRResult](#) [AMDTPwrGetProfilingState](#) ([AMDTPwrProfileState](#) *pState)
- [AMDTRResult](#) [AMDTPwrProfileClose](#) ()
- [AMDTRResult](#) [AMDTPwrSetSampleValueOption](#) ([AMDTSampleValueOption](#) opt)
- [AMDTRResult](#) [AMDTPwrGetSampleValueOption](#) ([AMDTSampleValueOption](#) *pOpt)
- [AMDTRResult](#) [AMDTPwrReadAllEnabledCounters](#) ([AMDТУInt32](#) *pNumOfSamples, [AMDTPwrSample](#) **ppData)
- [AMDTRResult](#) [AMDTPwrReadCounterHistogram](#) ([AMDТУInt32](#) counterId, [AMDТУInt32](#) *pNumEntries, [AMDTPwrHistogram](#) **ppData)
- [AMDTRResult](#) [AMDTPwrGetTimerSamplingPeriod](#) ([AMDТУInt32](#) *pIntervalMilliSec)
- [AMDTRResult](#) [AMDTPwrIsCounterEnabled](#) ([AMDТУInt32](#) counterId)
- [AMDTRResult](#) [AMDTPwrGetNumEnabledCounters](#) ([AMDТУInt32](#) *pCount)
- [AMDTRResult](#) [AMDTPwrGetApuPstateInfo](#) ([AMDTPwrApuPstateList](#) *pList)
- [AMDTRResult](#) [AMDTPwrGetCounterHierarchy](#) ([AMDТУInt32](#) counterId, [AMDTPwrCounterHierarchy](#) *pInfo)

7.2.1 Detailed Description

AMD Power Profiler APIs to configure, control and collect the power profile counters.

Definition in file [AMDTPowerProfileApi.h](#).

7.3 AMDTPowerProfileDataTypes.h File Reference

Data types and structure definitions used by AMD CodeXL Power Profiler APIs.
`#include <AMDTDefinitions.h>`

Data Structures

- struct [AMDTPwrDevice](#)
- struct [AMDTPwrCounterDesc](#)
- struct [AMDTPwrCounterValue](#)
- struct [AMDTPwrSystemTime](#)
- struct [AMDTPwrSample](#)
- struct [AMDTPwrApuPstate](#)
- struct [AMDTPwrApuPstateList](#)
- struct [AMDTPwrCounterHierarchy](#)
- struct [AMDTPwrHistogram](#)

Defines

- `#define` [AMD_T_PWR_ALL_DEVICES](#) 0xFFFFFFFFFUL
- `#define` [AMD_T_MAX_PSTATES](#) 8

Typedefs

- `typedef` `AMDTUInt32` [AMDTPwrDeviceId](#)

Enumerations

- `enum` [AMDTPwrProfileMode](#) { [AMD_T_PWR_PROFILE_MODE_ONLINE](#) }
- `enum` [AMDTPwrDeviceType](#) {
[AMD_T_PWR_DEVICE_SYSTEM](#), [AMD_T_PWR_DEVICE_PACKAGE](#),
[AMD_T_PWR_DEVICE_CPU_COMPUTE_UNIT](#), [AMD_T_PWR_DEVICE_CPU_CORE](#),
[AMD_T_PWR_DEVICE_INTERNAL_GPU](#), [AMD_T_PWR_DEVICE_EXTERNAL_GPU](#), [AMD_T_PWR_DEVICE_SVI2](#), [AMD_T_PWR_DEVICE_CNT](#) }
- `enum` [AMDTPwrCategory](#) {
[AMD_T_PWR_CATEGORY_POWER](#), [AMD_T_PWR_CATEGORY_FREQUENCY](#), [AMD_T_PWR_CATEGORY_TEMPERATURE](#), [AMD_T_PWR_CATEGORY_VOLTAGE](#),
[AMD_T_PWR_CATEGORY_CURRENT](#), [AMD_T_PWR_CATEGORY_DVFS](#),
[AMD_T_PWR_CATEGORY_PROCESS](#), [AMD_T_PWR_CATEGORY_TIME](#),
[AMD_T_PWR_CATEGORY_COUNT](#), [AMD_T_PWR_CATEGORY_CNT](#) }

- enum `AMDTPwrAggregation` { `AMDT_PWR_VALUE_SINGLE`, `AMDT_PWR_VALUE_CUMULATIVE`, `AMDT_PWR_VALUE_HISTOGRAM`, `AMDT_PWR_VALUE_CNT` }
- enum `AMDTPwrUnit` {
`AMDT_PWR_UNIT_TYPE_COUNT`, `AMDT_PWR_UNIT_TYPE_PERCENT`, `AMDT_PWR_UNIT_TYPE_RATIO`, `AMDT_PWR_UNIT_TYPE_MILLI_SECOND`,
`AMDT_PWR_UNIT_TYPE_JOULE`, `AMDT_PWR_UNIT_TYPE_WATT`, `AMDT_PWR_UNIT_TYPE_VOLT`, `AMDT_PWR_UNIT_TYPE_MILLI_AMPERE`,
`AMDT_PWR_UNIT_TYPE_MEGA_HERTZ`, `AMDT_PWR_UNIT_TYPE_CENTIGRADE`, `AMDT_PWR_UNIT_TYPE_CNT` }
- enum `AMDTPwrProfileState` {
`AMDT_PWR_PROFILE_STATE_UNINITIALIZED`, `AMDT_PWR_PROFILE_STATE_IDLE`, `AMDT_PWR_PROFILE_STATE_RUNNING`, `AMDT_PWR_PROFILE_STATE_PAUSED`,
`AMDT_PWR_PROFILE_STATE_STOPPED`, `AMDT_PWR_PROFILE_STATE_ABORTED`, `AMDT_PWR_PROFILE_STATE_CNT` }
- enum `AMDTSampleValueOption` { `AMDT_PWR_SAMPLE_VALUE_INSTANTANEOUS`, `AMDT_PWR_SAMPLE_VALUE_LIST`, `AMDT_PWR_SAMPLE_VALUE_AVERAGE`, `AMDT_PWR_SAMPLE_VALUE_CNT` }
- enum `AMDTApuPStates` {
`AMDT_PWR_PSTATE_PB0`, `AMDT_PWR_PSTATE_PB1`, `AMDT_PWR_PSTATE_PB2`, `AMDT_PWR_PSTATE_PB3`,
`AMDT_PWR_PSTATE_PB4`, `AMDT_PWR_PSTATE_PB5`, `AMDT_PWR_PSTATE_PB6`, `AMDT_PWR_PSTATE_P0`,
`AMDT_PWR_PSTATE_P1`, `AMDT_PWR_PSTATE_P2`, `AMDT_PWR_PSTATE_P3`, `AMDT_PWR_PSTATE_P4`,
`AMDT_PWR_PSTATE_P5`, `AMDT_PWR_PSTATE_P6`, `AMDT_PWR_PSTATE_P7` }

7.3.1 Detailed Description

Data types and structure definitions used by AMD CodeXL Power Profiler APIs.

Definition in file [AMDTPowerProfileDataTypes.h](#).

7.3.2 Define Documentation

7.3.2.1 `#define AMDT_PWR_ALL_DEVICES 0xFFFFFFFFFUL`

HW Components for which power counters are supported are called devices. Following are such components:

- AMD APU's and its subcomponents like CPU Compute-units, CPU Cores, integrated GPUs
- AMD discrete GPUs This macro denotes all the devices that are relevant to power profiling.

Examples:

[CollectAllCounters.cpp](#).

Definition at line 28 of file AMDTPowerProfileDataTypes.h.

7.3.2.2 #define AMDT_MAX_PSTATES 8

Maximum number of available APU P-States

Definition at line 32 of file AMDTPowerProfileDataTypes.h.

7.3.3 Typedef Documentation**7.3.3.1 typedef AMDTUInt32 AMDTPwrDeviceId**

Device Id

Examples:

[CollectAllCounters.cpp](#).

Definition at line 36 of file AMDTPowerProfileDataTypes.h.

Chapter 8

Example Documentation

8.1 CollectAllCounters.cpp

Example program to collect all the available counters.

```
//=====
// (c) 2015 Advanced Micro Devices, Inc.
//
//
//=====

// This sample shows the code for:
// - Initializing the AMDTPwrProfile API in online mode
// - Get the number of available counters and enable all the counters
// - Start the profiling
// - Periodically read the counter values and report till the user has requested
  to stop

#include <AMDDefinitions.h>
#include <AMDTPwrProfileApi.h>
#include <AMDTPwrProfileDataTypes.h>

void CollectAllCounters()
{
    AMDTResult hResult;

    // Initialize online mode
    hResult = AMDTPwrProfileInitialize(AMDT_PWR_PROFILE_MODE_ONLINE);
    // --- Handle the error

    // Profile Configuration
    // 1. Get the supported counters
    // 2. Enable all the counter
    // 3. Set the timer configuration

    // Get the supported counter details
    AMDTUInt32 nbrCounters;
    AMDTPwrCounterDesc *pCounters;
    AMDTPwrDeviceId deviceId = AMDT_PWR_ALL_DEVICES;

    hResult = AMDTPwrGetDeviceCounters(deviceId, &nbrCounters, &pCounters);
```

```

// --- Handle the error

// Enable all the counters
hResult = AMDTPwrEnableAllCounters();
// --- Handle the error

// Set the timer configuration
AMDTUInt32 samplingInterval = 100;      // in milliseconds
AMDTUInt32 profilingDuration = 1000;    // in milliseconds
hResult = AMDTPwrSetTimerSamplingPeriod (samplingInterval);
// --- Handle the error

// Start the Profile Run

hResult = AMDTPwrStartProfiling();

// Collect and report the counter values periodically
//   1. Take the snapshot of the counter values
//   2. Read the counter values
//   3. Report the counter values

bool isProfiling = true;
bool stopProfiling = false; // set by monitor thread
AMDTUInt32 nbrSamples;

while (isProfiling)
{
    // sleep for refresh duration - at least equivalent to the sampling i
nterval specified
    #if defined ( WIN32 )           // for Windows
        Sleep(samplingInterval);
    #else                           // for Linux
        usleep(samplingInterval * 1000);
    #endif

    // read all the counter values
    AMDTPwrSample* pSampleData;
    hResult = AMDTPwrReadAllEnabledCounters (&nbrSamples, &pSampleData);

    // iterate over all the samples and report the sampled counter values

    for (AMDTUInt32 idx = 0; idx < nbrSamples; idx++)
    {
        // Timestamp
        fprintf(stdout, "Timestamp : %lu ", (pSampleData->m_systemTime.
m_second * 1000000 + pSampleData->m_systemTime.m_microSecond) / 1000);

        // Iterate over the sampled counter values and print
        for (unsigned int i = 0; i < pSampleData->m_numOfValues; i++)
        {
            // Get the counter descriptor to print the counter name
            AMDTPwrCounterDesc counterDesc;
            AMDTPwrGetCounterDesc (pSampleData->m_counterValues->
m_counterID, &counterDesc);

            fprintf(stdout, "%s : %f ", counterDesc.m_name, pSampleData->
m_counterValues->m_counterValue);

            pSampleData->m_counterValues++;
        } // iterate over the sampled counters

        fprintf(stdout, "\n");
    }
}

```

```
        pSampleData++;
    } // iterate over all the samples collected

    // check if we exceeded the profile duration
    if ((profilingDuration > 0)
        && ((pSampleData-1)->m_elapsedTimeMs >= profilingDuration))
    {
        stopProfiling = true;
    }

    if (stopProfiling)
    {
        // stop the profiling
        hResult = AMDTPwrStopProfiling();
        isProfiling = false;
    }
}

// Close the profiler
hResult = AMDTPwrProfileClose();
}

int main()
{
    CollectAllCounters();
}
```

Index

AMDT_PWR_CATEGORY_CNT profiling, 12	AMDT_PWR_PROFILE_MODE_- ONLINE profiling, 11
AMDT_PWR_CATEGORY_COUNT profiling, 12	AMDT_PWR_PROFILE_STATE_- ABORTED profiling, 14
AMDT_PWR_CATEGORY_CURRENT profiling, 12	AMDT_PWR_PROFILE_STATE_CNT profiling, 14
AMDT_PWR_CATEGORY_DVFS profiling, 12	AMDT_PWR_PROFILE_STATE_IDLE profiling, 13
AMDT_PWR_CATEGORY_- FREQUENCY profiling, 12	AMDT_PWR_PROFILE_STATE_- PAUSED profiling, 13
AMDT_PWR_CATEGORY_POWER profiling, 12	AMDT_PWR_PROFILE_STATE_- RUNNING profiling, 13
AMDT_PWR_CATEGORY_PROCESS profiling, 12	AMDT_PWR_PROFILE_STATE_- STOPPED profiling, 13
AMDT_PWR_CATEGORY_- TEMPERATURE profiling, 12	AMDT_PWR_PROFILE_STATE_- UNINITIALIZED profiling, 13
AMDT_PWR_CATEGORY_TIME profiling, 12	AMDT_PWR_PSTATE_P0 profiling, 14
AMDT_PWR_CATEGORY_VOLTAGE profiling, 12	AMDT_PWR_PSTATE_P1 profiling, 14
AMDT_PWR_DEVICE_CNT profiling, 12	AMDT_PWR_PSTATE_P2 profiling, 14
AMDT_PWR_DEVICE_CPU_- COMPUTE_UNIT profiling, 12	AMDT_PWR_PSTATE_P3 profiling, 14
AMDT_PWR_DEVICE_CPU_CORE profiling, 12	AMDT_PWR_PSTATE_P4 profiling, 14
AMDT_PWR_DEVICE_EXTERNAL_- GPU profiling, 12	AMDT_PWR_PSTATE_P5 profiling, 14
AMDT_PWR_DEVICE_INTERNAL_- GPU profiling, 12	AMDT_PWR_PSTATE_P6 profiling, 14
AMDT_PWR_DEVICE_PACKAGE profiling, 12	AMDT_PWR_PSTATE_P7 profiling, 14
AMDT_PWR_DEVICE_SVI2 profiling, 12	AMDT_PWR_PSTATE_PB0 profiling, 14
AMDT_PWR_DEVICE_SYSTEM profiling, 12	AMDT_PWR_PSTATE_PB1

- profiling, [14](#)
- AMDT_PWR_PSTATE_PB2
 - profiling, [14](#)
- AMDT_PWR_PSTATE_PB3
 - profiling, [14](#)
- AMDT_PWR_PSTATE_PB4
 - profiling, [14](#)
- AMDT_PWR_PSTATE_PB5
 - profiling, [14](#)
- AMDT_PWR_PSTATE_PB6
 - profiling, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
 - AVERAGE
 - profiling, [14](#)
- AMDT_PWR_SAMPLE_VALUE_CNT
 - profiling, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
 - INSTANTANEOUS
 - profiling, [14](#)
- AMDT_PWR_SAMPLE_VALUE_LIST
 - profiling, [14](#)
- AMDT_PWR_UNIT_TYPE_-
 - CENTIGRADE
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_CNT
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_COUNT
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_JOULE
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_MEGA_-
 - HERTZ
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_MILLI_-
 - AMPERE
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_MILLI_-
 - SECOND
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_PERCENT
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_RATIO
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_VOLT
 - profiling, [13](#)
- AMDT_PWR_UNIT_TYPE_WATT
 - profiling, [13](#)
- AMDT_PWR_VALUE_CNT
 - profiling, [13](#)
- AMDT_PWR_VALUE_CUMULATIVE
 - profiling, [13](#)
- AMDT_PWR_VALUE_HISTOGRAM
 - profiling, [13](#)
- AMDT_PWR_VALUE_SINGLE
 - profiling, [13](#)
- AMDT_DRIVER_VERSION_-
 - MISMATCH
 - AMDTDefinitions.h, [50](#)
- AMDT_ERROR_ABORT
 - AMDTDefinitions.h, [46](#)
- AMDT_ERROR_ACCESSDENIED
 - AMDTDefinitions.h, [45](#)
- AMDT_ERROR_BIOS_VERSION_-
 - NOT_SUPPORTED
 - AMDTDefinitions.h, [50](#)
- AMDT_ERROR_COUNTER_-
 - ALREADY_ENABLED
 - AMDTDefinitions.h, [48](#)
- AMDT_ERROR_COUNTER_-
 - NOHIERARCHY
 - AMDTDefinitions.h, [51](#)
- AMDT_ERROR_COUNTER_NOT_-
 - ACCESSIBLE
 - AMDTDefinitions.h, [51](#)
- AMDT_ERROR_COUNTER_NOT_-
 - ENABLED
 - AMDTDefinitions.h, [48](#)
- AMDT_ERROR_COUNTERS_NOT_-
 - ENABLED
 - AMDTDefinitions.h, [50](#)
- AMDT_ERROR_DRIVER_-
 - ALREADY_INITIALIZED
 - AMDTDefinitions.h, [47](#)
- AMDT_ERROR_DRIVER_-
 - UNAVAILABLE
 - AMDTDefinitions.h, [47](#)
- AMDT_ERROR_DRIVER_-
 - UNINITIALIZED
 - AMDTDefinitions.h, [48](#)
- AMDT_ERROR_FAIL
 - AMDTDefinitions.h, [45](#)
- AMDT_ERROR_HANDLE
 - AMDTDefinitions.h, [46](#)
- AMDT_ERROR_INTERNAL
 - AMDTDefinitions.h, [49](#)
- AMDT_ERROR_INVALID_-
 - COUNTERID
 - AMDTDefinitions.h, [48](#)
- AMDT_ERROR_INVALID_DEVICEID
 - AMDTDefinitions.h, [48](#)

- AMDT_ERROR_INVALIDARG
AMDTDefinitions.h, 45
- AMDT_ERROR_INVALIDDATA
AMDTDefinitions.h, 46
- AMDT_ERROR_INVALIDPATH
AMDTDefinitions.h, 46
- AMDT_ERROR_LOCKED
AMDTDefinitions.h, 47
- AMDT_ERROR_NO_WRITE_-
PERMISSION
AMDTDefinitions.h, 48
- AMDT_ERROR_NODATA
AMDTDefinitions.h, 46
- AMDT_ERROR_NOFILE
AMDTDefinitions.h, 46
- AMDT_ERROR_NOTAVAILABLE
AMDTDefinitions.h, 46
- AMDT_ERROR_NOTIMPL
AMDTDefinitions.h, 46
- AMDT_ERROR_NOTSUPPORTED
AMDTDefinitions.h, 47
- AMDT_ERROR_OUTOFMEMORY
AMDTDefinitions.h, 45
- AMDT_ERROR_PLATFORM_NOT_-
SUPPORTED
AMDTDefinitions.h, 49
- AMDT_ERROR_PREVIOUS_-
SESSION_NOT_CLOSED
AMDTDefinitions.h, 51
- AMDT_ERROR_PROFILE_-
ALREADY_CONFIGURED
AMDTDefinitions.h, 50
- AMDT_ERROR_PROFILE_-
ALREADY_STARTED
AMDTDefinitions.h, 49
- AMDT_ERROR_PROFILE_DATA_-
NOT_AVAILABLE
AMDTDefinitions.h, 49
- AMDT_ERROR_PROFILE_-
DATAFILE_NOT_SET
AMDTDefinitions.h, 49
- AMDT_ERROR_PROFILE_NOT_-
CONFIGURED
AMDTDefinitions.h, 50
- AMDT_ERROR_PROFILE_NOT_-
PAUSED
AMDTDefinitions.h, 49
- AMDT_ERROR_PROFILE_NOT_-
STARTED
AMDTDefinitions.h, 49
- AMDT_ERROR_PROFILE_SESSION_-
EXISTS
AMDTDefinitions.h, 50
- AMDT_ERROR_SMU_ACCESS_-
FAILED
AMDTDefinitions.h, 50
- AMDT_ERROR_TIMEOUT
AMDTDefinitions.h, 47
- AMDT_ERROR_TIMER_NOT_SET
AMDTDefinitions.h, 48
- AMDT_ERROR_UNEXPECTED
AMDTDefinitions.h, 45
- AMDT_MAX_PSTATES
AMDTPowerProfileDataTypes.h, 56
- AMDT_PWR_ALL_DEVICES
AMDTPowerProfileDataTypes.h, 55
- AMDT_STATUS_OK
AMDTDefinitions.h, 45
- AMDT_STATUS_PENDING
AMDTDefinitions.h, 47
- AMDT_WARN_IGPU_DISABLED
AMDTDefinitions.h, 47
- AMDT_WARN_SMU_DISABLED
AMDTDefinitions.h, 47
- AMDTApuPStates
profiling, 14
- AMDTDefinitions.h, 43
- AMDT_DRIVER_VERSION_-
MISMATCH, 50
- AMDT_ERROR_ABORT, 46
- AMDT_ERROR_-
ACCESSDENIED, 45
- AMDT_ERROR_BIOS_-
VERSION_NOT_-
SUPPORTED, 50
- AMDT_ERROR_COUNTER_-
ALREADY_ENABLED, 48
- AMDT_ERROR_COUNTER_-
NOHIERARCHY, 51
- AMDT_ERROR_COUNTER_-
NOT_ACCESSIBLE, 51
- AMDT_ERROR_COUNTER_-
NOT_ENABLED, 48
- AMDT_ERROR_COUNTERS_-
NOT_ENABLED, 50
- AMDT_ERROR_DRIVER_-
ALREADY_INITIALIZED,
47
- AMDT_ERROR_DRIVER_-
UNAVAILABLE, 47

- AMDT_ERROR_DRIVER_-
UNINITIALIZED, [48](#)
- AMDT_ERROR_FAIL, [45](#)
- AMDT_ERROR_HANDLE, [46](#)
- AMDT_ERROR_INTERNAL, [49](#)
- AMDT_ERROR_INVALID_-
COUNTERID, [48](#)
- AMDT_ERROR_INVALID_-
DEVICEID, [48](#)
- AMDT_ERROR_INVALIDARG, [45](#)
- AMDT_ERROR_INVALIDDATA, [46](#)
- AMDT_ERROR_INVALIDPATH, [46](#)
- AMDT_ERROR_LOCKED, [47](#)
- AMDT_ERROR_NO_WRITE_-
PERMISSION, [48](#)
- AMDT_ERROR_NODATA, [46](#)
- AMDT_ERROR_NOFILE, [46](#)
- AMDT_ERROR_-
NOTAVAILABLE, [46](#)
- AMDT_ERROR_NOTIMPL, [46](#)
- AMDT_ERROR_-
NOTSUPPORTED, [47](#)
- AMDT_ERROR_-
OUTOFMEMORY, [45](#)
- AMDT_ERROR_PLATFORM_-
NOT_SUPPORTED, [49](#)
- AMDT_ERROR_PREVIOUS_-
SESSION_NOT_CLOSED, [51](#)
- AMDT_ERROR_PROFILE_-
ALREADY_CONFIGURED, [50](#)
- AMDT_ERROR_PROFILE_-
ALREADY_STARTED, [49](#)
- AMDT_ERROR_PROFILE_-
DATA_NOT_AVAILABLE, [49](#)
- AMDT_ERROR_PROFILE_-
DATAFILE_NOT_SET, [49](#)
- AMDT_ERROR_PROFILE_NOT_-
CONFIGURED, [50](#)
- AMDT_ERROR_PROFILE_NOT_-
PAUSED, [49](#)
- AMDT_ERROR_PROFILE_NOT_-
STARTED, [49](#)
- AMDT_ERROR_PROFILE_-
SESSION_EXISTS, [50](#)
- AMDT_ERROR_SMU_ACCESS_-
FAILED, [50](#)
- AMDT_ERROR_TIMEOUT, [47](#)
- AMDT_ERROR_TIMER_NOT_-
SET, [48](#)
- AMDT_ERROR_UNEXPECTED, [45](#)
- AMDT_STATUS_OK, [45](#)
- AMDT_STATUS_PENDING, [47](#)
- AMDT_WARN_IGPU_-
DISABLED, [47](#)
- AMDT_WARN_SMU_DISABLED, [47](#)
- AMDTResult, [51](#)
- AMDDeviceType
profiling, [11](#)
- AMDTPowerProfileApi.h, [52](#)
- AMDTPowerProfileDataTypes.h, [54](#)
- AMDT_MAX_PSTATES, [56](#)
- AMDT_PWR_ALL_DEVICES, [55](#)
- AMDTPwrDeviceId, [56](#)
- AMDTPwrAggregation
profiling, [12](#)
- AMDTPwrApuPstate, [29](#)
- m_frequency, [29](#)
- m_isBoosted, [29](#)
- m_state, [29](#)
- AMDTPwrApuPstateList, [31](#)
- m_cnt, [31](#)
- m_stateInfo, [31](#)
- AMDTPwrCategory
profiling, [12](#)
- AMDTPwrCounterDesc, [32](#)
- m_aggregation, [33](#)
- m_category, [33](#)
- m_counterID, [32](#)
- m_description, [33](#)
- m_deviceId, [32](#)
- m_maxValue, [33](#)
- m_minValue, [33](#)
- m_name, [32](#)
- m_units, [33](#)
- AMDTPwrCounterHierarchy, [34](#)
- m_childCnt, [34](#)
- m_counter, [34](#)
- m_parent, [34](#)
- m_pChildList, [34](#)
- AMDTPwrCounterValue, [35](#)
- m_counterID, [35](#)
- m_counterValue, [35](#)

- AMDTpwrDevice, 36
 - m_deviceID, 36
 - m_isAccessible, 36
 - m_pDescription, 36
 - m_pFirstChild, 37
 - m_pName, 36
 - m_pNextDevice, 37
 - m_type, 36
- AMDTpwrDeviceId
 - AMDTPowerProfileDataTypes.h, 56
- AMDTpwrDisableCounter
 - profiling, 18
- AMDTpwrEnableAllCounters
 - profiling, 18
- AMDTpwrEnableCounter
 - profiling, 17
- AMDTpwrGetApuPstateInfo
 - profiling, 27
- AMDTpwrGetCounterDesc
 - profiling, 16
- AMDTpwrGetCounterHierarchy
 - profiling, 27
- AMDTpwrGetDeviceCounters
 - profiling, 16
- AMDTpwrGetMinimalTimerSamplingPeriod
 - profiling, 19
- AMDTpwrGetNumEnabledCounters
 - profiling, 26
- AMDTpwrGetProfilingState
 - profiling, 22
- AMDTpwrGetSampleValueOption
 - profiling, 23
- AMDTpwrGetSystemTopology
 - profiling, 15
- AMDTpwrGetTimerSamplingPeriod
 - profiling, 25
- AMDTpwrHistogram, 38
 - m_counterId, 38
 - m_numOfBins, 38
 - m_pBins, 38
 - m_pRange, 38
- AMDTpwrIsCounterEnabled
 - profiling, 26
- AMDTpwrPauseProfiling
 - profiling, 21
- AMDTpwrProfileClose
 - profiling, 22
- AMDTpwrProfileInitialize
 - profiling, 15
- AMDTpwrProfileMode
 - profiling, 11
- AMDTpwrProfileState
 - profiling, 13
- AMDTpwrReadAllEnabledCounters
 - profiling, 24
- AMDTpwrReadCounterHistogram
 - profiling, 24
- AMDTpwrResumeProfiling
 - profiling, 22
- AMDTpwrSample, 39
 - m_counterValues, 40
 - m_elapsedTimeMs, 39
 - m_numOfValues, 39
 - m_recordId, 39
 - m_systemTime, 39
- AMDTpwrSetSampleValueOption
 - profiling, 23
- AMDTpwrSetTimerSamplingPeriod
 - profiling, 19
- AMDTpwrStartProfiling
 - profiling, 20
- AMDTpwrStopProfiling
 - profiling, 21
- AMDTpwrSystemTime, 41
 - m_microSecond, 41
 - m_second, 41
- AMDTpwrUnit
 - profiling, 13
- AMDTResult
 - AMDTDefinitions.h, 51
- AMDTSampleValueOption
 - profiling, 14
- m_aggregation
 - AMDTpwrCounterDesc, 33
- m_category
 - AMDTpwrCounterDesc, 33
- m_childCnt
 - AMDTpwrCounterHierarchy, 34
- m_cnt
 - AMDTpwrApuPstateList, 31
- m_counter
 - AMDTpwrCounterHierarchy, 34
- m_counterID
 - AMDTpwrCounterDesc, 32
 - AMDTpwrCounterValue, 35
- m_counterId
 - AMDTpwrHistogram, 38
- m_counterValue
 - AMDTpwrCounterValue, 35

- m_counterValues
 - AMDT_PwrSample, 40
- m_description
 - AMDT_PwrCounterDesc, 33
- m_deviceID
 - AMDT_PwrDevice, 36
- m_deviceId
 - AMDT_PwrCounterDesc, 32
- m_elapsedTimeMs
 - AMDT_PwrSample, 39
- m_frequency
 - AMDT_PwrApuPstate, 29
- m_isAccessible
 - AMDT_PwrDevice, 36
- m_isBoosted
 - AMDT_PwrApuPstate, 29
- m_maxValue
 - AMDT_PwrCounterDesc, 33
- m_microSecond
 - AMDT_PwrSystemTime, 41
- m_minValue
 - AMDT_PwrCounterDesc, 33
- m_name
 - AMDT_PwrCounterDesc, 32
- m_numOfBins
 - AMDT_PwrHistogram, 38
- m_numOfValues
 - AMDT_PwrSample, 39
- m_parent
 - AMDT_PwrCounterHierarchy, 34
- m_pBins
 - AMDT_PwrHistogram, 38
- m_pChildList
 - AMDT_PwrCounterHierarchy, 34
- m_pDescription
 - AMDT_PwrDevice, 36
- m_pFirstChild
 - AMDT_PwrDevice, 37
- m_pName
 - AMDT_PwrDevice, 36
- m_pNextDevice
 - AMDT_PwrDevice, 37
- m_pRange
 - AMDT_PwrHistogram, 38
- m_recordId
 - AMDT_PwrSample, 39
- m_second
 - AMDT_PwrSystemTime, 41
- m_state
 - AMDT_PwrApuPstate, 29
- m_stateInfo
 - AMDT_PwrApuPstateList, 31
- m_systemTime
 - AMDT_PwrSample, 39
- m_type
 - AMDT_PwrDevice, 36
- m_units
 - AMDT_PwrCounterDesc, 33
- Power Profiling, 9
- profiling
 - AMDT_PWR_CATEGORY_CNT, 12
 - AMDT_PWR_CATEGORY_COUNT, 12
 - AMDT_PWR_CATEGORY_CURRENT, 12
 - AMDT_PWR_CATEGORY_DVFS, 12
 - AMDT_PWR_CATEGORY_FREQUENCY, 12
 - AMDT_PWR_CATEGORY_POWER, 12
 - AMDT_PWR_CATEGORY_PROCESS, 12
 - AMDT_PWR_CATEGORY_TEMPERATURE, 12
 - AMDT_PWR_CATEGORY_TIME, 12
 - AMDT_PWR_CATEGORY_VOLTAGE, 12
 - AMDT_PWR_DEVICE_CNT, 12
 - AMDT_PWR_DEVICE_CPU_COMPUTE_UNIT, 12
 - AMDT_PWR_DEVICE_CPU_CORE, 12
 - AMDT_PWR_DEVICE_EXTERNAL_GPU, 12
 - AMDT_PWR_DEVICE_INTERNAL_GPU, 12
 - AMDT_PWR_DEVICE_PACKAGE, 12
 - AMDT_PWR_DEVICE_SVI2, 12
 - AMDT_PWR_DEVICE_SYSTEM, 12
 - AMDT_PWR_PROFILE_MODE_ONLINE, 11
 - AMDT_PWR_PROFILE_STATE_ABORTED, 14

- AMDT_PWR_PROFILE_STATE_-
CNT, [14](#)
- AMDT_PWR_PROFILE_STATE_-
IDLE, [13](#)
- AMDT_PWR_PROFILE_STATE_-
PAUSED, [13](#)
- AMDT_PWR_PROFILE_STATE_-
RUNNING, [13](#)
- AMDT_PWR_PROFILE_STATE_-
STOPPED, [13](#)
- AMDT_PWR_PROFILE_STATE_-
UNINITIALIZED, [13](#)
- AMDT_PWR_PSTATE_P0, [14](#)
- AMDT_PWR_PSTATE_P1, [14](#)
- AMDT_PWR_PSTATE_P2, [14](#)
- AMDT_PWR_PSTATE_P3, [14](#)
- AMDT_PWR_PSTATE_P4, [14](#)
- AMDT_PWR_PSTATE_P5, [14](#)
- AMDT_PWR_PSTATE_P6, [14](#)
- AMDT_PWR_PSTATE_P7, [14](#)
- AMDT_PWR_PSTATE_PB0, [14](#)
- AMDT_PWR_PSTATE_PB1, [14](#)
- AMDT_PWR_PSTATE_PB2, [14](#)
- AMDT_PWR_PSTATE_PB3, [14](#)
- AMDT_PWR_PSTATE_PB4, [14](#)
- AMDT_PWR_PSTATE_PB5, [14](#)
- AMDT_PWR_PSTATE_PB6, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
AVERAGE, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
CNT, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
INSTANTANEOUS, [14](#)
- AMDT_PWR_SAMPLE_VALUE_-
LIST, [14](#)
- AMDT_PWR_UNIT_TYPE_-
CENTIGRADE, [13](#)
- AMDT_PWR_UNIT_TYPE_CNT,
[13](#)
- AMDT_PWR_UNIT_TYPE_-
COUNT, [13](#)
- AMDT_PWR_UNIT_TYPE_-
JOULE, [13](#)
- AMDT_PWR_UNIT_TYPE_-
MEGA_HERTZ, [13](#)
- AMDT_PWR_UNIT_TYPE_-
MILLI_AMPERE, [13](#)
- AMDT_PWR_UNIT_TYPE_-
MILLI_SECOND, [13](#)
- AMDT_PWR_UNIT_TYPE_-
PERCENT, [13](#)
- AMDT_PWR_UNIT_TYPE_-
RATIO, [13](#)
- AMDT_PWR_UNIT_TYPE_-
VOLT, [13](#)
- AMDT_PWR_UNIT_TYPE_-
WATT, [13](#)
- AMDT_PWR_VALUE_CNT, [13](#)
- AMDT_PWR_VALUE_-
CUMULATIVE, [13](#)
- AMDT_PWR_VALUE_-
HISTOGRAM, [13](#)
- AMDT_PWR_VALUE_SINGLE,
[13](#)
- AMDTApuPStates, [14](#)
- AMDTPwrDeviceType, [11](#)
- AMDTPwrAggregation, [12](#)
- AMDTPwrCategory, [12](#)
- AMDTPwrDisableCounter, [18](#)
- AMDTPwrEnableAllCounters, [18](#)
- AMDTPwrEnableCounter, [17](#)
- AMDTPwrGetApuPstateInfo, [27](#)
- AMDTPwrGetCounterDesc, [16](#)
- AMDTPwrGetCounterHierarchy, [27](#)
- AMDTPwrGetDeviceCounters, [16](#)
- AMDTPwrGetMinimalTimerSam-
plingPeriod, [19](#)
- AMDTPwrGetNumEnabledCoun-
ters, [26](#)
- AMDTPwrGetProfilingState, [22](#)
- AMDTPwrGetSampleValueOption,
[23](#)
- AMDTPwrGetSystemTopology, [15](#)
- AMDTPwrGetTimerSamplingPe-
riod, [25](#)
- AMDTPwrIsCounterEnabled, [26](#)
- AMDTPwrPauseProfiling, [21](#)
- AMDTPwrProfileClose, [22](#)
- AMDTPwrProfileInitialize, [15](#)
- AMDTPwrProfileMode, [11](#)
- AMDTPwrProfileState, [13](#)
- AMDTPwrReadAllEnabledCoun-
ters, [24](#)
- AMDTPwrReadCounterHistogram,
[24](#)
- AMDTPwrResumeProfiling, [22](#)
- AMDTPwrSetSampleValueOption,
[23](#)

AMDTPwrSetTimerSamplingPeriod, [19](#)
AMDTPwrStartProfiling, [20](#)
AMDTPwrStopProfiling, [21](#)
AMDTPwrUnit, [13](#)
AMDTSampleValueOption, [14](#)