# AMDTPowerProfileAPI

Generated by Doxygen 1.8.9.1

Mon Jan 30 2017 15:56:55

# Contents

# Chapter 1

# CodeXL Power Profiler API

The AMDTPwrProfileAPI is a powerful library to help analyze the energy efficiency of systems based on AMD CPUs, APUs and Discrete GPUs.

This API:

- Provides counters to read the power, thermal and frequency characteristics of APU/dGPU and their subcomponents.
- Supports AMD APUs (Kaveri, Temash, Mullins, Carrizo), Discrete GPUs (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.
- 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:

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# 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
- struct AMDTPwrProcessInfo
- struct AMDTPwrInstrumentedPowerData
- struct AMDTPwrCategoryInfo

### Enumerations

- enum AMDTPwrProfileMode { AMDT_PWR_PROFILE_MODE_ONLINE, AMDT_PWR_PROFILE_MODE_OFFLINE }
- enum AMDTDeviceType { AMDT_PWR_DEVICE_SYSTEM, AMDT_PWR_DEVICE_PACKAGE, AMDT_PWR_DEVICE_CPU_COMPUTE_UNIT, AMDT_PWR_DEVICE_CPU_CORE, AMDT_PWR_DEVICE_INTERNAL_GPU, AMDT_PWR_DEVICE_EXTERNAL_GPU, AMDT_PWR_DEVICE_SVI2, AMDT_PWR_DEVICE_CNT }
- enum AMDTPwrCategory { AMDT_PWR_CATEGORY_POWER, AMDT_PWR_CATEGORY_FREQUENCY, AMDT_PWR_CATEGORY_TEMPERATURE, AMDT_PWR_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_ENERGY, AMDT_PWR_CATEGORY_CORRELATED_POWER, 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 profileMode)
- AMDTResult AMDTPwrGetSystemTopology (AMDTPwrDevice **ppTopology)
- AMDTResult AMDTPwrGetDeviceCounters (AMDTPwrDeviceId deviceId, AMDTUInt32 *pNumCounters, AMDTPwrCounterDesc **ppCounterDescs)
- AMDTResult AMDTPwrGetCounterDesc (AMDTUInt32 counterId, AMDTPwrCounterDesc *pCounterDesc)
- AMDTResult AMDTPwrEnableCounter (AMDTUInt32 counterId)
- AMDTResult AMDTPwrDisableCounter (AMDTUInt32 counterId)
- AMDTResult AMDTPwrEnableAllCounters ()
- AMDTResult AMDTPwrGetMinimalTimerSamplingPeriod (AMDTUInt32 *pIntervalMilliSec)
- AMDTResult AMDTPwrSetTimerSamplingPeriod (AMDTUInt32 interval)
- AMDTResult AMDTPwrStartProfiling ()
- AMDTResult AMDTPwrStopProfiling ()
- AMDTResult AMDTPwrPauseProfiling ()
- AMDTResult AMDTPwrResumeProfiling ()
- AMDTResult AMDTPwrGetProfilingState (AMDTPwrProfileState *pState)
- AMDTResult AMDTPwrProfileClose ()
- AMDTResult AMDTPwrSetSampleValueOption (AMDTSampleValueOption opt)
- AMDTResult AMDTPwrGetSampleValueOption (AMDTSampleValueOption *pOpt)
- AMDTResult AMDTPwrReadAllEnabledCounters (AMDTUInt32 *pNumOfSamples, AMDTPwrSample **ppData)
- AMDTResult AMDTPwrReadCounterHistogram (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTPwrHistogram **ppData)
- AMDTResult AMDTPwrReadCumulativeCounter (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTFloat32 **ppData)
- AMDTResult AMDTPwrGetTimerSamplingPeriod (AMDTUInt32 *pIntervalMilliSec)
- AMDTResult AMDTPwrIsCounterEnabled (AMDTUInt32 counterId)
- AMDTResult AMDTPwrGetNumEnabledCounters (AMDTUInt32 *pCount)
- AMDTResult AMDTPwrGetApuPstateInfo (AMDTPwrApuPstateList *pList)
- AMDTResult AMDTPwrGetCounterHierarchy (AMDTUInt32 counterId, AMDTPwrCounterHierarchy *pInfo)
- AMDTResult AMDTPwrGetNodeTemperature (AMDTFloat32 *pNodeTemp)

- AMDTResult AMDTEnableProcessProfiling (void)
- AMDTResult AMDTGetProcessProfileData (AMDTUInt32 *pPIDCount, AMDTPwrProcessInfo **ppData, AMDTUInt32 pidVal, bool reset)
- AMDTResult AMDTPwrGetModuleProfileData (AMDTPwrModuleData **ppData, AMDTUInt32 *pModuleCount, AMDTFloat32 *pPower)
- AMDTResult AMDTPwrGetCategoryInfo (AMDTPwrCategory category, AMDTPwrCategoryInfo *pCategory)

### 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

***AMDT_PWR_PROFILE_MODE_OFFLINE*** Power profile mode is offline

Definition at line 62 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 72 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_ENERGY*** Energy consumed

***AMDT_PWR_CATEGORY_CORRELATED_POWER*** Energy consumed

***AMDT_PWR_CATEGORY_CNT*** Total category count

Definition at line 87 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 106 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 117 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 135 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 149 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_PSTA↩ TES_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 163 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**

| | | |
|---|---|---|
| in | *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_INVALID_ARG* | 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**

| | | |
|---|---|---|
| out | *ppTopology* | Device tree |

**Returns**

The status of system topology request

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | 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**

| | | |
|---|---|---|
| in | *deviceId* | The deviceId provided by AMDTPwrGetSystemTopology() function or AMDT_PWR_ALL_DEVICES to represent all the devices returned by AMDTPwrGetSystemTopology() |
| out | *pNumCounters* | Number of counters supported by the device |
| out | *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**

| | | |
|---|---|---|
| in | *counterId* | Counter index |
| out | *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_INVALID_ARG* | 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**

| | | |
|---|---|---|
| in | *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**

| | | |
|---|---|---|
| in | *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**

| | | |
|---|---|---|
| out | *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_INVALID_ARG* | 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**

| | | |
|---|---|---|
| in | *interval* | sampling period in millisecond |

**Returns**

The status of sampling time set request

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | 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_ACCESDENIED* | 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_PROFIL↩ E_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_PROFIL↩ E_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**

| | | |
|---|---|---|
| out | *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_INVALID↩ ARG* | 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 AMDTPwr↩ ProfileInitialize().

**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 AMDTPwrReadAllEnabledCounters() function.

**Parameters**

| | | |
|---|---|---|
| in | *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_INVALID↩ ARG* | An invalid opt was specified as parameter |
| *AMDT_ERROR_DRIVER↩ _UNINITIALIZED* | AMDTPwrProfileInitialize() function was neither called nor successful |
| *AMDT_ERROR_PROFIL↩ E_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**

| | | |
|---|---|---|
| out | *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_INVALID↩ ARG* | 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**

| | | |
|---|---|---|
| out | *ppData* | Processed profile data. No need to allocate or free the memory data is valid till we call this API next time |
| out | *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_INVALID_ARG* | 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**

| | | |
|---|---|---|
| in | *counterId* | Histogram type counter id. AMDT_PWR_ALL_COUNTERS to represent all supported histogram counters. |

| | | |
|---|---|---|
| out | *pNumEntries* | Number of entries in the histogram |
| out | *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_INVALID_ARG* | 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 AMDTPwrReadCumulativeCounter ( AMDTUInt32 *counterId*, AMDTUInt32 * *pNumEntries*, AMDTFloat32 ** *ppData* )

API to read one of the derived accumulated counters values from the raw counter values.

**Parameters**

| | | |
|---|---|---|
| in | *counterId* | Cumulative type counter id. AMDT_PWR_ALL_COUNTERS to represent all supported accumulated counters. |
| out | *pNumEntries* | Number of cumulative counters |
| out | *ppData* | Accumulated counter data for the given counter id |

**Returns**

The status of reading accumulated counter data

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | 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_PROFIL↩ E_DATA_NOT_AVAILAB↩ LE* | Profile data is not yet available |
| *AMDT_ERROR_OUTOF↩ MEMORY* | Memory not available |
| *AMDT_ERROR_FAIL* | An internal error occurred |

#### 5.1.3.21 AMDTResult AMDTPwrGetTimerSamplingPeriod ( AMDTUInt32 * *pIntervalMilliSec* )

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

**Parameters**

| | | |
|---|---|---|
| out | *pIntervalMilliSec* | sampling period in millisecond |

**Returns**

The status of the get sampling interval request

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID↩ ARG* | 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 |

#### 5.1.3.22 AMDTResult AMDTPwrIsCounterEnabled ( AMDTUInt32 *counterId* )

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

**Parameters**

| | | |
|---|---|---|
| in | *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_COUNT↩ ER_NOT_ENABLED* | Counter is not enabled already |
| *AMDT_ERROR_FAIL* | An internal error occurred |

#### 5.1.3.23 AMDTResult AMDTPwrGetNumEnabledCounters ( AMDTUInt32 * *pCount* )

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

**Parameters**

| | | |
|---|---|---|
| out | *pCount* | Number of enabled counters |

**Returns**

The status of query request

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success; Counter is enabled |
| *AMDT_ERROR_INVALID_ARG* | 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 |

#### 5.1.3.24 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**

| | | |
|---|---|---|
| out | *pList* | List of P-States |

**Returns**

The status reading the pstate list for the platform

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | 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 |

#### 5.1.3.25 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**

| | | |
|---|---|---|
| in | *counterId* | The counter id for which the dependent counters information is requested |
| out | *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_INVALID_ARG* | 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 hierarchical relationship |
| *AMDT_ERROR_FAIL* | An internal error occurred |

#### 5.1.3.26 AMDTResult AMDTPwrGetNodeTemperature ( AMDTFloat32 * *pNodeTemp* )

This API provides the node temperature in Tctl scale. This temperature is not absolute.

**Parameters**

| | | |
|---|---|---|
| out | *pNodeTemp* | Provides node temperature. |

**Returns**

The status retrieving hierarchical information for the given counters

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | NULL pointer was passed as argument |
| *AMDT_ERROR_DRIVER_UNINITIALIZED* | AMDTPwrProfileInitialize() function was neither called nor successful |
| *AMDT_ERROR_FAIL* | An internal error occurred |

#### 5.1.3.27 AMDTResult AMDTEnableProcessProfiling ( void )

This API enables process profiling. This API will enable backend and driver to collect running PIDs at lowest possible granularity and attribute them against the power values provided by the SMU.

**Returns**

The status of the process profiling enable request

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_DRIVER_UNINITIALIZED* | AMDTPwrProfileInitialize() function was neither called nor successful |
| *AMDT_ERROR_PROFILE_ALREADY_STARTED* | Process profiling can not be set when the profile is already started |
| *AMDT_WARN_PROCESS_PROFILE_ALREADY_ENABLED* | Process profiling already enabled |

| | |
|---|---|
| *AMDT_ERROR_OUTOF↩ MEMORY* | Failed to allocate required memory |
| *AMDT_ERROR_PROCE↩ SS_PROFILE_NOT_SUP↩ PORTED* | Platform not supported |

#### 5.1.3.28 AMDTResult AMDTGetProcessProfileData ( AMDTUInt32 * *pPIDCount*, AMDTPwrProcessInfo ** *ppData*, AMDTUInt32 *pidVal*, bool *reset* )

This API will provide the list of running PIDs so far from the time of profile start or bewteen two consecutive call of this function, their agregated power indicators. This API can be called at any point of time from start of the profile to the stop of the profile.

**Parameters**

| | | |
|---|---|---|
| in | *pidVal* | If AMD_PWR_ALL_PIDS is set will collect power for all the pids else for the given pid value. |
| in | *reset* | If set power data is collected from the time profile start else data bewtween two consecutive call of this fn. |
| out | *pPIDCount* | Total number of PIDs running during the profile session |
| out | *ppData* | List of PIDs with their power indicators |

**Returns**

The status reading process profiling data

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID↩ ARG* | NULL pointer was passed as pData parameters |
| *AMDT_ERROR_DRIVER↩ _UNINITIALIZED* | AMDTPwrProfileInitialize() function was neither called nor successful |
| *AMDT_ERROR_PROFIL↩ E_NOT_STARTED* | Profile is not started |
| *AMDT_ERROR_PROFIL↩ E_DATA_NOT_AVAILAB↩ LE* | Profile data is not yet available |
| *AMDT_ERROR_OUTOF↩ MEMORY* | Memory not available |
| *AMDT_ERROR_PROCE↩ SS_PROFILE_NOT_ENA↩ BLED* | Process profiling not enabled |
| *AMDT_ERROR_FAIL* | An internal error occurred |
| *AMDT_ERROR_PROCE↩ SS_PROFILE_NOT_SUP↩ PORTED* | Platform not supported |

#### 5.1.3.29 AMDTResult AMDTPwrGetModuleProfileData ( AMDTPwrModuleData ** *ppData*, AMDTUInt32 * *pModuleCount*, AMDTFloat32 * *pPower* )

This API will provide the list of running modules so far from the time of profile start of the profile and provides their agregated power indicators. This API can be called at any point of time from start of the profile to the stop of the profile.

**Parameters**

| | | |
|---|---|---|
| out | *pModuleCount* | Total number of modules running during the profile session |
| out | *ppData* | List of modules with their power indicators |
| out | *pPower* | Total power consumed by the profile session |

**Returns**

The status reading process profiling data

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | NULL pointer was passed as pData 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_PROCESS_PROFILE_NOT_ENABLED* | Process profiling not enabled |
| *AMDT_ERROR_FAIL* | An internal error occurred |
| *AMDT_ERROR_PROCESS_PROFILE_NOT_SUPPORTED* | Platform not supported |

#### 5.1.3.30 AMDTResult AMDTPwrGetCategoryInfo ( AMDTPwrCategory *category*, AMDTPwrCategoryInfo * *pCategory* )

This API will provide the category details for a given category id..

**Parameters**

| | | |
|---|---|---|
| in | *category* | Counter category |
| out | *pCategory* | Provides details of the category |

**Returns**

The status retrieving category information for the given category

**Return values**

| | |
|---|---|
| *AMDT_STATUS_OK* | On Success |
| *AMDT_ERROR_INVALID_ARG* | NULL pointer was passed as argument |
| *AMDT_ERROR_DRIVER_UNINITIALIZED* | AMDTPwrProfileInitialize() function was neither called nor successful |

# 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 252 of file AMDTPowerProfileDataTypes.h.

### 6.1.2 Field Documentation

#### 6.1.2.1 AMDTApuPStates m_state

P-State number

Definition at line 254 of file AMDTPowerProfileDataTypes.h.

#### 6.1.2.2 bool m_isBoosted

Boosted P-State flag

Definition at line 255 of file AMDTPowerProfileDataTypes.h.

#### 6.1.2.3 AMDTUInt32 m_frequency

P-State frequency

Definition at line 256 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 262 of file AMDTPowerProfileDataTypes.h.

### 6.2.2 Field Documentation

#### 6.2.2.1 AMDTUInt32 m_cnt

Number of P-States

Definition at line 264 of file AMDTPowerProfileDataTypes.h.

#### 6.2.2.2 AMDTPwrApuPstate m_stateInfo[AMDT_MAX_PSTATES]

P-States list

Definition at line 265 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.3 AMDTPwrCategoryInfo Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt8 m_name [AMDT_PWR_EXE_NAME_LENGTH]
- AMDTPwrCategory m_category

### 6.3.1 Detailed Description

Represents the counter category information.

Definition at line 351 of file AMDTPowerProfileDataTypes.h.

### 6.3.2 Field Documentation

#### 6.3.2.1 AMDTUInt8 m_name[AMDT_PWR_EXE_NAME_LENGTH]

Name of the category

Definition at line 353 of file AMDTPowerProfileDataTypes.h.

#### 6.3.2.2 AMDTPwrCategory m_category

Power/Freq/Temperature

Definition at line 354 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.4 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
- AMDTUInt32 m_parentCounterId

### 6.4.1 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 205 of file AMDTPowerProfileDataTypes.h.

### 6.4.2 Field Documentation

#### 6.4.2.1 AMDTUInt32 m_counterID

Counter index

Definition at line 207 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.2 AMDTUInt32 m_deviceId

Device Id

Definition at line 208 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.3 char* m_name

Name of the counter

**Examples:**

CollectAllCounters.cpp.

Definition at line 209 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.4 char* m_description

Description of the counter

Definition at line 210 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.5 AMDTPwrCategory m_category

Power/Freq/Temperature

Definition at line 211 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.6 AMDTPwrAggregation m_aggregation

Single/Histogram/Cumulative

Definition at line 212 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.7 AMDTFloat64 m_minValue

Minimum possible counter value

Definition at line 213 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.8 AMDTFloat64 m_maxValue

Maximum possible counter value

Definition at line 214 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.9 AMDTPwrUnit m_units

Seconds/MHz/Joules/Watts/Volt/Ampere

Definition at line 215 of file AMDTPowerProfileDataTypes.h.

#### 6.4.2.10 AMDTUInt32 m_parentCounterId

Counter id of the parent counter if applicable

Definition at line 216 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.5 AMDTPwrCounterHierarchy Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt32 m_counter
- AMDTUInt32 m_parent
- AMDTUInt32 m_childCnt
- AMDTUInt32 * m_pChildList

### 6.5.1 Detailed Description

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

Definition at line 272 of file AMDTPowerProfileDataTypes.h.

### 6.5.2 Field Documentation

#### 6.5.2.1 AMDTUInt32 m_counter

Counter Id

Definition at line 274 of file AMDTPowerProfileDataTypes.h.

#### 6.5.2.2 AMDTUInt32 m_parent

Parent counter Id

Definition at line 275 of file AMDTPowerProfileDataTypes.h.

#### 6.5.2.3 AMDTUInt32 m_childCnt

Number of child counters

Definition at line 276 of file AMDTPowerProfileDataTypes.h.

#### 6.5.2.4 AMDTUInt32* m_pChildList

List of child counters

Definition at line 277 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.6 AMDTPwrCounterValue Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt32 m_counterID
- AMDTFloat32 m_counterValue

### 6.6.1 Detailed Description

Structure represents a counter ID and its value

Definition at line 222 of file AMDTPowerProfileDataTypes.h.

### 6.6.2 Field Documentation

#### 6.6.2.1 AMDTUInt32 m_counterID

Counter index

Definition at line 224 of file AMDTPowerProfileDataTypes.h.

#### 6.6.2.2 AMDTFloat32 m_counterValue

Counter value

**Examples:**

CollectAllCounters.cpp.

Definition at line 225 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.7 AMDTPwrDevice Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTDeviceType m_type
- AMDTPwrDeviceId m_deviceID
- char * m_pName
- char * m_pDescription
- bool m_isAccessible
- AMDTPwrDevice * m_pFirstChild
- AMDTPwrDevice * m_pNextDevice

### 6.7.1 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 187 of file AMDTPowerProfileDataTypes.h.

### 6.7.2 Field Documentation

#### 6.7.2.1 AMDTDeviceType m_type

Device type- compute unit/Core/ package/ dGPU

Definition at line 189 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.2 AMDTPwrDeviceId m_deviceID

Device Id

Definition at line 190 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.3 char* m_pName

Name of the device

Definition at line 191 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.4 char* m_pDescription

Description about the device

Definition at line 192 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.5 bool m_isAccessible

If counters are accessible

Definition at line 193 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.6 AMDTPwrDevice* m_pFirstChild

Points to the sub-devices of this device

Definition at line 194 of file AMDTPowerProfileDataTypes.h.

#### 6.7.2.7 AMDTPwrDevice* m_pNextDevice

Points to the next device at the same hierarchy

Definition at line 195 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.8 AMDTPwrHistogram Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt32 m_counterId
- AMDTUInt32 m_numOfBins
- AMDTFloat32 m_range [AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT+1]
- AMDTFloat32 m_bins [AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT]

### 6.8.1 Detailed Description

Represents a generic histogram.

Definition at line 283 of file AMDTPowerProfileDataTypes.h.

### 6.8.2 Field Documentation

#### 6.8.2.1 AMDTUInt32 m_counterId

Counter being aggregated

Definition at line 285 of file AMDTPowerProfileDataTypes.h.

#### 6.8.2.2 AMDTUInt32 m_numOfBins

This is the number of histogram bins

Definition at line 286 of file AMDTPowerProfileDataTypes.h.

#### 6.8.2.3 AMDTFloat32 m_range[AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT+1]

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

Definition at line 287 of file AMDTPowerProfileDataTypes.h.

#### 6.8.2.4 AMDTFloat32 m_bins[AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT]

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

Definition at line 288 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.9 AMDTPwrInstrumentedPowerData Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt8 m_name [AMDT_PWR_MARKER_BUFFER_LENGTH]
- AMDTUInt8 m_userBuffer [AMDT_PWR_MARKER_BUFFER_LENGTH]
- AMDTPwrSystemTime m_systemStartTime
- AMDTUInt64 m_startTs
- AMDTUInt64 m_endTs
- AMDTPwrProcessInfo m_pidInfo

## 6.9.1 Detailed Description

Represents the instrumented power data.

Definition at line 337 of file AMDTPowerProfileDataTypes.h.

## 6.9.2 Field Documentation

### 6.9.2.1 AMDTUInt8 m_name[AMDT_PWR_MARKER_BUFFER_LENGTH]

Name of the user marker

Definition at line 339 of file AMDTPowerProfileDataTypes.h.

### 6.9.2.2 AMDTUInt8 m_userBuffer[AMDT_PWR_MARKER_BUFFER_LENGTH]

User supplied buffer

Definition at line 340 of file AMDTPowerProfileDataTypes.h.

### 6.9.2.3 AMDTPwrSystemTime m_systemStartTime

Profile start time

Definition at line 341 of file AMDTPowerProfileDataTypes.h.

### 6.9.2.4 AMDTUInt64 m_startTs

Marker start elapsed time

Definition at line 342 of file AMDTPowerProfileDataTypes.h.

### 6.9.2.5 AMDTUInt64 m_endTs

Marker end elapsed time

Definition at line 343 of file AMDTPowerProfileDataTypes.h.

### 6.9.2.6 AMDTPwrProcessInfo m_pidInfo

Process information

Definition at line 344 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.10 AMDTPwrModuleData Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt32 m_processId
- char m_processName [AMDT_PWR_EXE_NAME_LENGTH]
- char m_processPath [AMDT_PWR_EXE_PATH_LENGTH]
- AMDTFloat32 m_power
- AMDTFloat32 m_ipcLoad
- AMDTUInt64 m_sampleCnt
- bool m_isKernel
- char m_moduleName [AMDT_PWR_EXE_NAME_LENGTH]
- char m_modulePath [AMDT_PWR_EXE_PATH_LENGTH]
- AMDTUInt64 m_loadAddr
- AMDTUInt64 m_size

### 6.10.1 Detailed Description

Definition at line 318 of file AMDTPowerProfileDataTypes.h.

### 6.10.2 Field Documentation

#### 6.10.2.1 AMDTUInt32 m_processId

Process id

Definition at line 320 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.2 char m_processName[AMDT_PWR_EXE_NAME_LENGTH]

Executable name

Definition at line 321 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.3 char m_processPath[AMDT_PWR_EXE_PATH_LENGTH]

Path

Definition at line 322 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.4 AMDTFloat32 m_power

Power consumed

Definition at line 323 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.5 AMDTFloat32 m_ipcLoad

Agreegated IPC value

Definition at line 324 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.6 AMDTUInt64 m_sampleCnt

Number of PID samples

Definition at line 325 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.7 bool m_isKernel

Kernel/User module

Definition at line 326 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.8 char m_moduleName[AMDT_PWR_EXE_NAME_LENGTH]

Executable name

Definition at line 327 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.9 char m_modulePath[AMDT_PWR_EXE_PATH_LENGTH]

Path

Definition at line 328 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.10 AMDTUInt64 m_loadAddr

Module load address

Definition at line 329 of file AMDTPowerProfileDataTypes.h.

#### 6.10.2.11 AMDTUInt64 m_size

Module size

Definition at line 330 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.11 AMDTPwrProcessInfo Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt32 m_pid
- AMDTUInt64 m_sampleCnt
- AMDTFloat32 m_power
- AMDTFloat32 m_ipc
- char m_name [AMDT_PWR_EXE_NAME_LENGTH]
- char m_path [AMDT_PWR_EXE_PATH_LENGTH]

### 6.11.1 Detailed Description

Represents process power info.

Definition at line 294 of file AMDTPowerProfileDataTypes.h.

### 6.11.2 Field Documentation

#### 6.11.2.1 AMDTUInt32 m_pid

Process id

Definition at line 296 of file AMDTPowerProfileDataTypes.h.

#### 6.11.2.2 AMDTUInt64 m_sampleCnt

Number of PID samples

Definition at line 297 of file AMDTPowerProfileDataTypes.h.

#### 6.11.2.3 AMDTFloat32 m_power

PID power indicator

Definition at line 298 of file AMDTPowerProfileDataTypes.h.

#### 6.11.2.4 AMDTFloat32 m_ipc

Agreegated IPC value

Definition at line 299 of file AMDTPowerProfileDataTypes.h.

#### 6.11.2.5 char m_name[AMDT_PWR_EXE_NAME_LENGTH]

Executable name

Definition at line 300 of file AMDTPowerProfileDataTypes.h.

#### 6.11.2.6 char m_path[AMDT_PWR_EXE_PATH_LENGTH]

Path

Definition at line 301 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.12 AMDTPwrSample Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTPwrSystemTime m_systemTime

- AMDTUInt64 m_elapsedTimeMs
- AMDTUInt64 m_recordId
- AMDTUInt32 m_numOfValues
- AMDTPwrCounterValue * m_counterValues

### 6.12.1 Detailed Description

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

**Examples:**

CollectAllCounters.cpp.

Definition at line 240 of file AMDTPowerProfileDataTypes.h.

### 6.12.2 Field Documentation

#### 6.12.2.1 AMDTPwrSystemTime m_systemTime

Start time of Profiling

Definition at line 242 of file AMDTPowerProfileDataTypes.h.

#### 6.12.2.2 AMDTUInt64 m_elapsedTimeMs

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

**Examples:**

CollectAllCounters.cpp.

Definition at line 243 of file AMDTPowerProfileDataTypes.h.

#### 6.12.2.3 AMDTUInt64 m_recordId

Record id

Definition at line 244 of file AMDTPowerProfileDataTypes.h.

#### 6.12.2.4 AMDTUInt32 m_numOfValues

Number of counter values available

**Examples:**

CollectAllCounters.cpp.

Definition at line 245 of file AMDTPowerProfileDataTypes.h.

#### 6.12.2.5 AMDTPwrCounterValue* m_counterValues

list of counter values

**Examples:**

CollectAllCounters.cpp.

Definition at line 246 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.13 AMDTPwrSystemTime Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt64 m_second
- AMDTUInt64 m_microSecond

### 6.13.1 Detailed Description

This structure represents the system time in second and milliseconds

**Examples:**

CollectAllCounters.cpp.

Definition at line 231 of file AMDTPowerProfileDataTypes.h.

### 6.13.2 Field Documentation

#### 6.13.2.1 AMDTUInt64 m_second

Seconds

**Examples:**

CollectAllCounters.cpp.

Definition at line 233 of file AMDTPowerProfileDataTypes.h.

#### 6.13.2.2 AMDTUInt64 m_microSecond

Milliseconds

**Examples:**

CollectAllCounters.cpp.

Definition at line 234 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

## 6.14 ContextPowerData Struct Reference

`#include <AMDTPowerProfileDataTypes.h>`

### Data Fields

- AMDTUInt64 m_ip
- AMDTUInt32 m_processId
- AMDTUInt32 m_threadId
- AMDTUInt64 m_timeStamp
- AMDTUInt32 m_coreId
- AMDTFloat32 m_ipcLoad
- AMDTFloat32 m_power
- AMDTUInt64 m_sampleCnt

## 6.14.1 Detailed Description

Definition at line 305 of file AMDTPowerProfileDataTypes.h.

## 6.14.2 Field Documentation

### 6.14.2.1 AMDTUInt64 m_ip

Sample address

Definition at line 307 of file AMDTPowerProfileDataTypes.h.

### 6.14.2.2 AMDTUInt32 m_processId

Process id

Definition at line 308 of file AMDTPowerProfileDataTypes.h.

### 6.14.2.3 AMDTUInt32 m_threadId

Thread id

Definition at line 309 of file AMDTPowerProfileDataTypes.h.

### 6.14.2.4 AMDTUInt64 m_timeStamp

Sample time stamp

Definition at line 310 of file AMDTPowerProfileDataTypes.h.

### 6.14.2.5 AMDTUInt32 m_coreId

Cpu core id

Definition at line 311 of file AMDTPowerProfileDataTypes.h.

### 6.14.2.6 AMDTFloat32 m_ipcLoad

Agreegated IPC value

Definition at line 312 of file AMDTPowerProfileDataTypes.h.

#### 6.14.2.7 AMDTFloat32 m_power

Power consumed

Definition at line 313 of file AMDTPowerProfileDataTypes.h.

#### 6.14.2.8 AMDTUInt64 m_sampleCnt

Number of samples

Definition at line 314 of file AMDTPowerProfileDataTypes.h.

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

- AMDTPowerProfileDataTypes.h

# Chapter 7

# File Documentation

## 7.1 AMDTPowerProfileApi.h File Reference

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

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

### Functions

- AMDTResult AMDTPwrProfileInitialize (AMDTPwrProfileMode profileMode)
- AMDTResult AMDTPwrGetSystemTopology (AMDTPwrDevice **ppTopology)
- AMDTResult AMDTPwrGetDeviceCounters (AMDTPwrDeviceId deviceId, AMDTUInt32 *pNumCounters, AMDTPwrCounterDesc **ppCounterDescs)
- AMDTResult AMDTPwrGetCounterDesc (AMDTUInt32 counterId, AMDTPwrCounterDesc *pCounterDesc)
- AMDTResult AMDTPwrEnableCounter (AMDTUInt32 counterId)
- AMDTResult AMDTPwrDisableCounter (AMDTUInt32 counterId)
- AMDTResult AMDTPwrEnableAllCounters ()
- AMDTResult AMDTPwrGetMinimalTimerSamplingPeriod (AMDTUInt32 *pIntervalMilliSec)
- AMDTResult AMDTPwrSetTimerSamplingPeriod (AMDTUInt32 interval)
- AMDTResult AMDTPwrStartProfiling ()
- AMDTResult AMDTPwrStopProfiling ()
- AMDTResult AMDTPwrPauseProfiling ()
- AMDTResult AMDTPwrResumeProfiling ()
- AMDTResult AMDTPwrGetProfilingState (AMDTPwrProfileState *pState)
- AMDTResult AMDTPwrProfileClose ()
- AMDTResult AMDTPwrSetSampleValueOption (AMDTSampleValueOption opt)
- AMDTResult AMDTPwrGetSampleValueOption (AMDTSampleValueOption *pOpt)
- AMDTResult AMDTPwrReadAllEnabledCounters (AMDTUInt32 *pNumOfSamples, AMDTPwrSample **ppData)
- AMDTResult AMDTPwrReadCounterHistogram (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTPwrHistogram **ppData)
- AMDTResult AMDTPwrReadCumulativeCounter (AMDTUInt32 counterId, AMDTUInt32 *pNumEntries, AMDTFloat32 **ppData)
- AMDTResult AMDTPwrGetTimerSamplingPeriod (AMDTUInt32 *pIntervalMilliSec)
- AMDTResult AMDTPwrIsCounterEnabled (AMDTUInt32 counterId)
- AMDTResult AMDTPwrGetNumEnabledCounters (AMDTUInt32 *pCount)
- AMDTResult AMDTPwrGetApuPstateInfo (AMDTPwrApuPstateList *pList)
- AMDTResult AMDTPwrGetCounterHierarchy (AMDTUInt32 counterId, AMDTPwrCounterHierarchy *pInfo)

- AMDTResult AMDTPwrGetNodeTemperature (AMDTFloat32 *pNodeTemp)
- AMDTResult AMDTEnableProcessProfiling (void)
- AMDTResult AMDTGetProcessProfileData (AMDTUInt32 *pPIDCount, AMDTPwrProcessInfo **ppData, AMDTUInt32 pidVal, bool reset)
- AMDTResult AMDTPwrGetModuleProfileData (AMDTPwrModuleData **ppData, AMDTUInt32 *pModuleCount, AMDTFloat32 *pPower)
- AMDTResult AMDTPwrGetCategoryInfo (AMDTPwrCategory category, AMDTPwrCategoryInfo *pCategory)

### 7.1.1 Detailed Description

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

**Author**

AMD Developer Tools Team

## 7.2 AMDTPowerProfileDataTypes.h File Reference

Data types and structure definitions used by 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
- struct AMDTPwrProcessInfo
- struct ContextPowerData
- struct AMDTPwrModuleData
- struct AMDTPwrInstrumentedPowerData
- struct AMDTPwrCategoryInfo

### Macros

- #define AMDT_PWR_ALL_DEVICES 0xFFFFFFFFUL
- #define AMDT_PWR_ALL_COUNTERS 0xFFFFFFFFUL
- #define AMDT_PWR_EXE_NAME_LENGTH 64
- #define AMDT_PWR_EXE_PATH_LENGTH 256
- #define AMDT_MAX_PSTATES 8
- #define AMDT_PWR_MARKER_BUFFER_LENGTH 32
- #define AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT 32
- #define AMD_PWR_ALL_PIDS 0xFFFFFFFFU

### Typedefs

- typedef AMDTUInt32 AMDTPwrDeviceId

### Enumerations

- enum AMDTPwrProfileMode { AMDT_PWR_PROFILE_MODE_ONLINE, AMDT_PWR_PROFILE_MODE_OFFLINE }
- enum AMDTDeviceType { AMDT_PWR_DEVICE_SYSTEM, AMDT_PWR_DEVICE_PACKAGE, AMDT_PWR_DEVICE_CPU_COMPUTE_UNIT, AMDT_PWR_DEVICE_CPU_CORE, AMDT_PWR_DEVICE_INTERNAL_GPU, AMDT_PWR_DEVICE_EXTERNAL_GPU, AMDT_PWR_DEVICE_SVI2, AMDT_PWR_DEVICE_CNT }
- enum AMDTPwrCategory { AMDT_PWR_CATEGORY_POWER, AMDT_PWR_CATEGORY_FREQUENCY, AMDT_PWR_CATEGORY_TEMPERATURE, AMDT_PWR_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_ENERGY, AMDT_PWR_CATEGORY_CORRELATED_POWER, 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 }

## 7.2.1 Detailed Description

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

**Author**

AMD Developer Tools Team

## 7.2.2 Macro Definition Documentation

### 7.2.2.1 #define AMDT_PWR_ALL_DEVICES 0xFFFFFFFFUL

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

- AMD APUs 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 24 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.2 #define AMDT_PWR_ALL_COUNTERS 0xFFFFFFFFUL

This macro denotes all the counters that are relevant to power profiling.

Definition at line 29 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.3 #define AMDT_PWR_EXE_NAME_LENGTH 64

Process name length

Definition at line 33 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.4 #define AMDT_PWR_EXE_PATH_LENGTH 256

Process name length

Definition at line 37 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.5 #define AMDT_MAX_PSTATES 8

Maximum number of available APU P-States

Definition at line 41 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.6 #define AMDT_PWR_MARKER_BUFFER_LENGTH 32

Process marker buffer length

Definition at line 45 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.7 #define AMDT_PWR_HISTOGRAM_MAX_BIN_COUNT 32

Hisotgram maximum bin size

Definition at line 49 of file AMDTPowerProfileDataTypes.h.

#### 7.2.2.8 #define AMD_PWR_ALL_PIDS 0xFFFFFFFFU

All the PIDs are set

Definition at line 57 of file AMDTPowerProfileDataTypes.h.

### 7.2.3 Typedef Documentation

#### 7.2.3.1 typedef AMDTUInt32 AMDTPwrDeviceId

Device Id

Definition at line 53 of file AMDTPowerProfileDataTypes.h.

# Chapter 8

# Example Documentation

## 8.1 CollectAllCounters.cpp

Example program to collect all the available counters.

```
//==============================================================
// (c) 2017 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 <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

#include <AMDTPowerProfileApi.h>

void GetTimeStampString(AMDTPwrSystemTime& sampleTime, AMDTUInt64 elapsedMs, char*
      pTimeStr)
{
#define WINDOWS_TICK_PER_SEC     10000000
#define MICROSEC_IN_SECOND       1000000

#if defined ( WIN32 )
    ULARGE_INTEGER time;

    // Convert sample time to 100-nanosec
    time.QuadPart = (sampleTime.m_second * WINDOWS_TICK_PER_SEC) + (sampleTime.
      m_microSecond * 10);

    // adjust the absolute profile start TS with elapsed time (in ms)
    time.QuadPart += elapsedMs * 10000;

    FILETIME fileTime;
    fileTime.dwHighDateTime = (DWORD)(time.HighPart);
    fileTime.dwLowDateTime = (DWORD)(time.LowPart);

    SYSTEMTIME sysTime;

    if (FileTimeToSystemTime(&fileTime, &sysTime))
    {
        sprintf(pTimeStr, "%d:%d:%d:%03d", sysTime.wHour, sysTime.wMinute, sysTime.wSecond, sysTime.
      wMilliseconds);
    }

#else
    struct timeval ts;
    struct tm time;
    AMDTUInt64 tmp = 0;

    ts.tv_sec = sampleTime.m_second;
    ts.tv_usec = sampleTime.m_microSecond;
```

```
    tmp = ts.tv_usec + (elapsedMs * 1000);
    // when tmp > 1000000 usec add to seconds
    ts.tv_sec += tmp / MICROSEC_IN_SECOND;
    ts.tv_usec = tmp % MICROSEC_IN_SECOND;
    tzset();
    localtime_r(&(ts.tv_sec), &time);

    sprintf(pTimeStr, "%d:%d:%d:%03lu", time.tm_hour, time.tm_min, time.tm_sec, ts.tv_usec / (1000));
#endif
}

void CollectAllCounters()
{
    AMDTResult hResult = AMDT_STATUS_OK;

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

    // Configure the profile run
    //   1. Get the supported counters
    //   2. Enable all the counter
    //   3. Set the timer configuration

    // 1. Get the supported counter details
    AMDTUInt32 nbrCounters = 0;
    AMDTPwrCounterDesc* pCounters = NULL;
    AMDTPwrDeviceId deviceId = AMDT_PWR_ALL_DEVICES;

    hResult = AMDTPwrGetDeviceCounters(deviceId, &nbrCounters, &pCounters);
    assert(AMDT_STATUS_OK == hResult);

    // Enable all the counters
    hResult = AMDTPwrEnableAllCounters();
    assert(AMDT_STATUS_OK == hResult);

    // Set the timer configuration
    AMDTUInt32 samplingInterval = 100;      // in milliseconds
    AMDTUInt32 profilingDuration = 10;      // in seconds

    hResult = AMDTPwrSetTimerSamplingPeriod(samplingInterval);
    assert(AMDT_STATUS_OK == hResult);

    // Start the Profile Run
    hResult = AMDTPwrStartProfiling();
    assert(AMDT_STATUS_OK == hResult);

    // 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

    volatile bool isProfiling = true;
    bool stopProfiling = false;
    AMDTUInt32 nbrSamples = 0;

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

        // read all the counter values
        AMDTPwrSample* pSampleData = nullptr;

        hResult = AMDTPwrReadAllEnabledCounters(&nbrSamples, &pSampleData);

        if (AMDT_STATUS_OK != hResult)
        {
            continue;
        }

        if (nullptr != pSampleData)
        {
            // iterate over all the samples and report the sampled counter values
            for (AMDTUInt32 idx = 0; idx < nbrSamples; idx++)
            {
                // Iterate over the sampled counter values and print
                for (unsigned int i = 0; i < pSampleData[idx].m_numOfValues; i++)
                {
                    if (nullptr != pSampleData[idx].m_counterValues)
```

```
                {
                    // Get the counter descriptor to print the counter name
                    AMDTPwrCounterDesc counterDesc;
                    AMDTPwrGetCounterDesc(pSampleData[idx].m_counterValues->
    m_counterID, &counterDesc);

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

                    pSampleData[idx].m_counterValues++;
                }
            } // iterate over the sampled counters

            fprintf(stdout, "\n");
        } // iterate over all the samples collected

        // check if we exceeded the profile duration
        if ((profilingDuration > 0)
            && (pSampleData->m_elapsedTimeMs >= (profilingDuration * 1000)))
        {
            stopProfiling = true;
        }

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

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

int main()
{
    AMDTResult hResult = AMDT_STATUS_OK;
    CollectAllCounters();
    return hResult;
}
```

# Index