



SEP User Guide

Intel Corporation
www.intel.com
[Legal Information](#)

Contents

Notices and Disclaimers	4
Revision History	5
Chapter 1: About This Document	
Intended Audience.....	6
Related Information.....	6
Chapter 2: Event-Based Sampling	
Core Data Collection	7
Processor Event-Based Sampling (PEBS) Collection.....	7
-apc [[basic],[mem],[gpr],[lbr:[# of entries]],[xmm]]	7
Example Adaptive PEBS Commands.....	8
Last Branch Record (LBR) Collection	8
Collection on Hybrid Platforms	8
Collect Data with Intel® Resource Director Technology	13
Uncore Data Collection	15
Chapter 3: Using SEP	
Internals of Data Collection.....	16
Launch and Control SEP Collection.....	16
Launching SEP Collection.....	16
Launching with an Application	17
Launching a Delayed Collection	17
Launching Collection Indefinitely	17
Pausing Collection	17
Resuming Collection	17
Stopping Collection.....	18
Cancelling Collection.....	18
Chapter 4: SEP Commands	
Chapter 5: SEP Options	
Event Configuration Options.....	21
-atype <atype name1>, <atype name2>,	21
-atypelist [-config] [-details] [<atype1>, <atype2>,...]	21
-ec -event-config [-dc -data-config <optional-data1>,<optional- data2>...] "<event-name1>":modifier1=val:modifier2=val/ constraint1={:modifier3=val:modifier4=val}, "<event-name2>"... ..	22
-em -event-multiplexing [trigger=<fixed counter> factor=<value>]....	26
-experimental.....	26
Collection Options	27
-app <full-path-to-the-application>[-args <"list of application arguments">].....	27
-cm -cpu-mask <processor numbers>	27
-cp -continuous-profiling	27
-d -duration <in seconds>	27
-ebc -event-based-counts	27
-mr -multi-run	28
-nb -non-blocking	28

-osm -os-mode	29
-sam -sample-after-multiplier <value>	29
-sd -sampling-delay <delay in seconds>	29
-sp -start-paused	29
-uem -uncore-event-multiplexing [factor=<value>][timer=<value in ms>].....	29
-um -user-mode	30
-rdt-auto-rmid	30
Processor Event Based Sampling (PEBS) Options	30
-apc [[basic],[mem],[gpr],[lbr:[# of entries]],[xmm]]	30
Example Adaptive PEBS Commands.....	30
-fpc -full-pebs-capture.....	30
-multipebs <# of PEBS records>	31
-virt-phys-translation.....	31
Last Branch Records (LBRs) and Callstack Options	31
-apc lbr[:depth=# of entries] [-lbr <filter_name>] [-lbr-filter <filter1>:<filter2>...]	31
-callstacks.....	31
-lbr <capture_mode>	31
-lbr-filter <filter1>:<filter2>:<filter3>	32
P-STATE Options	32
Output Options	32
-of -options-from-file <file name>	33
-out -output-file <file name>	33
-verbose	33

Chapter 6: Use Intel® VTune™ Profiler with SEP

Hotspots Analysis in Hardware Event-based Sampling Mode	34
Microarchitecture Analysis	36
Custom Analysis Type	36
Viewing SEP Results in Intel® VTune™ Profiler.....	36

Notices and Disclaimers

Revision History

Revision Number	Description	Revision Date
1.0	Completed major documentation changes on previous version of the Sampling Enabling Product User's Guide and reset the document revision level from 3.12 to 1.0. Added commands for pausing, stopping, and resuming collection. Removed obsolete options. Added the <code>-atypes</code> and <code>-fpc</code> options.	November 2015
1.1	Removed obsolete option.	December 2015
1.2	Added Installation chapter.	February 2016
1.3	Upgraded version to SEP 4.0. Added Using Intel® VTune™ Amplifier with SEP chapter.	March 2016
1.4	Replaced obsolete <code>-atypes</code> option with new <code>-atypelist</code> option.	September 2016
1.5	Fixed location of drivers for FreeBSD*.	October 2016
1.6	Added new event modifiers for IA32/Intel® 64 architectures.	February 2017
1.7	Upgraded version to SEP 4.1. Updated with support for Intel® VTune™ Amplifier 2018 Beta.	April 2017
1.8	Removed obsolete examples. Fixed <code>-cpu-mask</code> and <code>-lbr-filter</code> examples.	June 2017
1.9	Updated <code>-pmu-types</code> description.	November 2017
2.0	Updated the guide with missing options and added description where required.	June 2018
2.1	Updated Using Intel® VTune™ Amplifier with SEP section to match changes made to analysis type names for the Intel VTune Amplifier 2019 product release: <ul style="list-style-type: none"> • General Exploration analysis is now known as Microarchitecture Analysis • Advanced Hotspots merged with Basic Hotspots, previous Advanced Hotspots are enabled by selecting Hardware Event-based Sampling mode 	September 2018
2.2	Minor updates to commands.	February 2019
2.3	Updates to reflect the product name change from the Intel® VTune™ Amplifier to Intel® VTune™ Profiler.	November, 2019

About This Document

SEP is a standalone command-line tool that performs hardware event-based sampling (EBS) on a given system. Hardware-based sampling is a low-overhead, system-wide profiling that identifies hotspots, giving a detailed look at the operating system and applications. SEP enables users to configure data collection, perform system-wide profiling, and store results in a binary (*.tb) file. You can import the output *.tb file into Intel® VTune™ Profiler and visualize the information graphically.

When installing SEP automatically with Intel® VTune™ Profiler, the drivers may need to be manually installed if the auto-installation fails. The following links provide information about manually installing sampling drivers for Windows*, Linux*, and Android* operating systems if auto-installation fails:

- [Windows* target](#)
- [Linux* target](#)
- [Android* target](#)

Intended Audience

This document is intended for all developers who use SEP to analyze performance data.

Related Information

For Intel® VTune™ Profiler information, go to <https://software.intel.com/en-us/intel-vtune-profiler-support/documentation>.

For Performance Monitoring Unit (PMU) counters information, go to Intel Software Developers Manual: <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>.

2

Event-Based Sampling

SEP is a performance data collector that uses event-based sampling, leveraging the counter overflow feature of the hardware Performance Monitoring Unit (PMU). The tool captures the processor's execution state (Instruction Pointer, Process ID, Thread ID, CPU number, etc.) each time a performance counter overflow raises an interrupt.

The number of events that can be monitored simultaneously in a single run is limited by the number of hardware performance counters available in the PMU of a processor. Some events can have limitations that allow them to be programmed only in certain counters. To overcome the limitation of available performance counters on the hardware, SEP splits events into multiple event groups. Each group consists of events that can be collected simultaneously.

Core Data Collection

SEP enables PMI as non-maskable (NMI) for Linux*, Android*, and FreeBSD* operating systems. All other operating systems use maskable PMI; in this case, the sampling mechanism cannot accurately capture the profiling data from masked regions (usually found in kernel or driver code).

In case of core data collection, the tool employs one of the following techniques to collect data for multiple event groups:

- Event multiplexing (default): SEP multiplexes the use of physical counters within a single sampling run and avoids the need for multiple runs of data collection at the cost of lower precision of the sampling data. SEP switches groups every few samples of a chosen trigger event. The default trigger event is `CPU_CLK_UNHALTED.REF_TSC`, and the default swap frequency is 50 samples. Refer to `-em` option to override the default options for core event multiplexing.
- Multiple runs: SEP runs a separate data collection on each event group. This mode can be enabled using `-mr` option of the tool.

Sample After Value (SAV) is the number of events after which the tool collects a sample. SEP internally defines an event-specific SAV that can be overridden by the user. For more information, see [:sa modifier](#).

Processor Event-Based Sampling (PEBS) Collection

Hardware PEBS enables capturing architectural state information of the processor upon counter overflow on applicable events. The Instruction Pointer and other hardware state captured in the PEBS mode are more accurate than regular sampling.

SEP provides the capability to program events to collect PEBS data in the output `*.tb` file. To program PEBS collection, see the `-fpc` option.

`-apc [[basic],[mem],[gpr],[lbr:[# of entries]],[xmm]]`

Adaptive PEBS capture on capable platforms. For PEBS-capable events provided by the user in the command line, SEP collects PEBS records in the sampling data. Users can select zero or more fields. Sampling data for each selection type contains related PEBS fields. For multiple selection types, sampling data will contain all relevant fields. For information on events that are PEBS capable for a given platform, see the Intel Software Developers Manual.

- `-apc` without any selection type, the sampling data contains basic PEBS record which includes record layout, applicable counters, PEBS eventing IP, PEBS TSC.
- `-apc mem` includes memory latency fields
- `-apc gpr` includes general purpose register fields
- `-apc lbr[:depth=<# of lbr entries>]` includes last branch record fields. By default, the tool collects maximum available `lbr` records for the platform. Optionally, when number of `lbr` entries is given, the tool programs PEBS control register to only fetch requested number of entries (min# - 1, max# - 32. # of entries is recommended to be a power of 2).

- `-apc xmm` includes XMM register fields.

If the user selects PEBS-capable events in the command line without using the `-apc` option, the tool still performs PEBS collection on those events and replaces non-PEBS eventing IP and TSC in sampling data with PEBS evening IP and TSC. No other fields added to the `*.tb` file.

Example Adaptive PEBS Commands

The following examples demonstrate a few combinations of the `-apc` command:

```
sep -start -apc -ec "INST_RETIREDD.ANY, BR_INST_RETIREDD.ALL_BRANCHES"
sep -start -apc basic,mem,gpr,xmm,lbr -ec "INST_RETIREDD.ANY, BR_INST_RETIREDD.ALL_BRANCHES"
```

Last Branch Record (LBR) Collection

The hardware LBR facility stores branch records in the LBR stack MSRs for the most recently taken branches, interrupts, and/or exceptions. A branch record consists of "from-instruction-pointer" and "to-instruction-pointer" addresses. The depth of the LBR stack can vary across processor families.

SEP can program branch events to collect LBR records and output them into the `*.tb` file. To program LBR collection, see the [-lbr option](#) and [-lbr-filter option](#).

Collection on Hybrid Platforms

Introduction to Hybrid Platforms

The recent Intel client architectures are based on a hybrid model with Performance Cores (P-Core) and Efficiency Cores (E-Core). Depending on the application, hybrid CPU architectures can distribute core usage more efficiently than non-hybrid architectures. P-Cores are designed to handle complex workloads while E-Cores are better suited for multi-threaded throughput and power-limited scenarios. At higher power envelopes, P-Cores can provide better performance than E-Cores. At lower power envelopes, E-Cores are more desirable. Each core type has different specifications and system configurations.

For these reasons, the P-cores are preferred for

- Priority tasks
- Limited threading applications

while E-Cores are better suited for:

- Power-limited scenarios
- Background applications that can meet their QOS (Quality of Service) requirements on that performance

Supported Core Types

To collect samples using SEP on hybrid platforms, you must first identify the core types that are supported on your system. To do this, run:

```
sep -pmu-types
```

For example, this output indicates that two core types supported by SEP tool on the system: `bigcore` and `smallcore`

```
$ sep -pmu-types
PMU Types supported on this platform:
bigcore
smallcore
imc
cbo
hac_cbo
ncu
hac_ncu
ufibridge
power
```

NOTE SEP notates P-Core as `bigcore` and E-Core as `smallcore`.

Available Core Types

Once you have identified the core types supported by your system, find out the core types that are available. Run:

```
sep -pmu-types available
```

In this example, there are two core types available on the system: `bigcore` and `smallcore`.

```
$ sep -pmu-types available
PMU Types available on this machine:
bigcore
smallcore
imc
cbo
hac_cbo
ncu
hac_ncu
ufibridge
power
```

Note that a core type supported by your system will not display in this output unless it is actually available on your system.

Core Type Specifications on Hybrid Platforms

Each core type has different specifications (such as cache, number of PerfMon counters etc) and system configurations on hybrid platforms.

To see the core type specification, run:

```
sep -platform-info
```

This command displays the following types of information about supported core types:

Number of Processors per Core Type

```
$ sep -platform-info
.....
total_number_of_processors ..... 22
number_of_online_processors ..... 22
number_of_processors (bigcore) ..... 12
number_of_online_processors (bigcore) ..... 12
number_of_processors (smallcore) ..... 10
number_of_online_processors (smallcore) ..... 10
.....
```

Cache Info per Core Type

```
$ sep -platform-info
.....
Cache Info (bigcore):
L1 Data Cache ..... 48KB, 12-way, 64-byte line size
    2 HW threads share this cache, No SW Init Required
L1 Code Cache ..... 64KB, 16-way, 64-byte line size
    2 HW threads share this cache, No SW Init Required
L2 Unified Cache ..... 2MB, 16-way, 64-byte lin size
    8 HW threads share this cache, No SW Init Required
64-byte Prefetching
```

```
Cache Info (smallcore):
L1 Data Cache ..... 32KB, 8-way, 64-byte line size
    No SW Init Required
L1 Code Cache ..... 64KB, 8-way, 64-byte line size
    No SW Init Required
L2 Unified Cache ..... 2MB, 16-way, 64-byte line size
    8 HW threads share this cache, No SW Init Required
64-byte Prefetching
.....
```

Specs and Configurations per Core Type

```
$ sep -platform-info
.....
Processor Features (bigcore):
number_of_selectors ..... 8
number_of_var_counters ..... 8
number_of_fixed_ctrs ..... 4
Fixed Counter Events:
counter 0 ..... INST_RETIRED.ANY
counter 1 ..... CPU_CLK_UNHALTED.THREAD
counter 2 ..... CPU_CLK_UNHALTED.REF_TSC
counter 3 ..... TOPDOWN.SLOTS
number of devices ..... 1
number_of_events ..... 595
    (Thermal Throttling) (Enabled)
    (Hyper-Threading) (Enabled)
    (DCU IP Prefetching) (Enabled)
    (DCU Streamer Prefetching) (Enabled)
    (MLC AMP Prefetching) (Enabled)
    (MLC Spatial Prefetching) (Enabled)
    (MLC Streamer Prefetching) (Enabled)
    (Cores Per Package: 6)
    (Threads Per Package: 12)
    (Threads Per COre: 2)

Processor Features (smallcore):
number_of_selectors ..... 8
number_of_var_counters ..... 8
number_of_fixed_ctrs ..... 3
Fixed Counter Events:
counter 0 ..... INST_RETIRED.ANY
counter 1 ..... CPU_CLK_UNHALTED.CORE
counter 2 ..... CPU_CLK_UNHALTED.REF_TSC
number of devices ..... 1
number_of_events ..... 422
    (Thermal Throttling) (Enabled)
    (DCU IP Prefetching) (Enabled)
    (DCU Streamer Prefetching) (Enabled)
    (DCU Next Page Prefetching) (Enabled)
    (MLC Streamer Prefetching) (Enabled)
    (Cores Per Package: 5)
    (Threads Per Package: 10)
    (Threads Per Core: 1)
.....
```

The information is displayed per core type with the each PMU name, such as `bigcore` and `smallcore`.

Mapping Core Type to Processors

SEP collects samples for perfmon events only from applicable core types. To understand the collection result, you must first understand the core type to which each processor is mapped.

```
$ sep -platform-info
.....
OS Processor <-> Physical/Logical Mapping
-----
OS Processor      Phys.Package   Core   Logical Processor   Core Type   Module
0                 0             0       0                   bigcore     2
1                 0             0       1                   bigcore     2
2                 0             0       0                   bigcore     3
3                 0             0       1                   bigcore     3
4                 0             0       0                   bigcore     4
5                 0             0       1                   bigcore     4
6                 0             0       0                   bigcore     5
7                 0             0       1                   bigcore     5
8                 0             0       0                   bigcore     6
9                 0             0       1                   bigcore     6
10                0             0       0                   bigcore     7
11                0             0       1                   bigcore     7
12                0             0       0                   smallcore   0
13                0             1       0                   smallcore   0
14                0             2       0                   smallcore   0
15                0             3       0                   smallcore   0
16                0             0       0                   smallcore   1
17                0             1       0                   smallcore   1
18                0             2       0                   smallcore   1
19                0             3       0                   smallcore   1
20                0             0       0                   smallcore   8
21                0             1       0                   smallcore   8
.....
```

The output indicates that processors 0-11 are `bigcore` and processors 12-21 are `smallcore`.

When the `tb7` file is generated from the collection, the samples from CPU 0-11 are for `bigcore` and the samples from CPU 12-21 are for `smallcore`.

The `-platform-info` command also provides Module ID mapping to the processor if the module exists on the system.

Event Specifications

Each core type has a different perfmon event list. These events can be defined as common events or core-type specific events depends on the number of core types that have these events.

Common Events

There are events supported in multiple core types. Those events are considered as common events and are collected on all applicable processors.

To check the list of supported events per core type for all core types, run this command:

```
sel -el [pmu name]
```

For example,

```
$ sep -el bigcore
INST_RETIRED.ANY
INST_RETIRED.PREC_DIST
BR_INST_RETIRED.ALL_BRANCHESLONGEST_LAT_CACHE.MISS
```

```
TLB_FLUSH.DTLB_THREAD
L2_RQSTS.HIT
.....
```

```
$ sep -el smallcore
INST_RETIRE.ANY
MACHINE_CLEARS.PAGE_FAULT
SERIALIZATION.NON_C01_MS_SCB
BR_INST_RETIRE.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
ICACHE.MISSES
ICACHE.ACCESSSES
.....
```

Events that are found in events lists for both core types are **common events** such as `INST_RETIRE`.ANY,`BR_INST_RETIRE`.ALL_BRANCHES, or `LONGEST_LAT_CACHE.MISS`.

Core-Type specific events

If the events are applicable only to certain core types, those events are considered as core-type specific events and are collected only on applicable core type processors.

To check the supported event list per core type for all core types, run this command:

```
sep -el [pmu name]
```

For example,

```
$ sep -el bigcore
INST_RETIRE.ANY
INST_RETIRE.PREC_DIST
BR_INST_RETIRE.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
TLB_FLUSH.DTLB_THREAD
L2_RQSTS.HIT
.....
```

```
$ sep -el smallcore
INST_RETIRE.ANY
MACHINE_CLEARS.PAGE_FAULT
SERIALIZATION.NON_C01_MS_SCB
BR_INST_RETIRE.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
ICACHE.MISSES
ICACHE.ACCESSSES
.....
```

Those events which are found only in the event list for a single core type are treated as core-type specific events.

For example, the following events are exclusively `bigcore` events:

- `INST_RETIRE`.PREC_DIST
- `TLB_FLUSH.DTLB_THREAD`
- `L2_RQSTS.HIT`

These events will be collected on `bigcore` processors only.

The following events are `smallcore` events:

- `MACHINE_CLEARS_PAGE_FAULT`
- `SERIALIZATION.NON_C01_MS_SCB`
- `ICACHE.MISSES`

- ICACHE.ACCESSSES

These events are collected on `smallcore` processors only.

Event Collection

This section describes how you collect common and core-type events.

Collect Common Events

To specify common events from the event list and collect these events using SEP, run:

```
$ sep -start -d <duration> -ec <events> -out common_event_collection.tb7

for example>
$ sep -start -d 5 -ec INST_RETIRED.ANY, LONGEST_LAT_CACHE.MISS
```

Collect Core-Type Specific Events

To specify `bigcore`-specific events from the event list and collect these events using SEP, run:

```
$ sep -start -d <duration> -ec <events> -out big_core_collection.tb7

for example>
$ sep -start -d 5 -ec TOPDOWN.SLOTS, CORE_POWER.LICENSE_1
```

To specify `smallcore`-specific events from the event list and collect these events using SEP, run:

```
$ sep -start -d <duration> -ec <events> -out small_core_collection.tb7

for example>
$ sep -start -d 5 -ec ICACHE.MISSES, ICACHE.ACCESSSES
```

Collect Combination of Common and Core-Type Specific Events

To collect a combination of common events, `bigcore`-specific events, and `smallcore`-specific events, run:

```
$ sep -start -d <duration> -ec <events> -out common_big_small_collection.tb7

for example>
$ sep -start -d 5 -ec
INST_RETIRED.ANY, LONGEST_LAT_CACHE.MISS, TOPDOWN.SLOTS, CORE_POWER.LICENSE_1, ICACHE.MISSES, ICACHE.A
CCESSES
```

Collect Data with Intel® Resource Director Technology

Introduction

Intel® Resource Director Technology (Intel® RDT) is a set of monitoring capabilities that you can use to measure shared resource metrics such as L3 cache occupancy in each logical processor.

The Resource Monitoring ID (RMID) is used to monitor the shared resources. The RMID provides a layer of abstraction between the software thread and logical processors. Each software thread is assigned to a unique RMID. The RMID can be assigned to a single logical processor or multiple logical processors (through `IA32_PQR_ASSOC_MSR`) for monitoring.

Operating Technologies

The operations of Intel® RDT are governed by two technologies:

- **Cache Monitoring Technology (CMT):** This allows an operating system, hypervisor, or similar system management agent to determine the usage of cache by applications running on the platform. The associated event in SEP is `UNC_CMT_L3_CACHE_OCCUPANCY`.

- **Memory Bandwidth Monitoring (MBM):** This is used to monitor the bandwidth from one level of the cache hierarchy to the next. The associated event in SEP is `UNC_MBM_TOTAL_EXTERNAL_BW`, `UNC_MBM_LOCAL_EXTERNAL_BW`.

You can find more information about these technologies in Chapter 17.16 of the Intel® Software Developer Manual.

Additionally, SEP provides the RMID association and RDT allocation through these events:

- `UNC_RDT_PQR_ASSOC` - bit 0:9 represents RMID and bit 32:63 represents CLOS
- `UNC_CAT_L2_MASK` - represents L2 cache allocation capacity associated with the COS on each logical processors.
- `UNC_CAT_L3_MASK` - represents L3 cache allocation capacity associated with the COS on each logical processors.

For more information, see these chapters in the Intel® Software Developer Manual:

- Monitoring Resource (RMID) Association - Chapter 17.16.6
- Cache Allocation Technology Architecture - Chapter 17.17.1

RDT Support Information

To see support information for Intel RDT on your system, run:

```
sep -platform-info
```

For example,

```
$ sep -platform-info
.....
RDT HW Support:
  L3 Cache Occupancy      : Yes
  Total Memory Bandwidth  : Yes
  Local Memory Bandwidth  : Yes
  L3 Cache Allocation     : Yes
  L2 Cache Allocation     : No
  Highest Available RMID  : 175
  Sample Multiplier       : 90112
.....
```

Supported RDT Events

SEP determines the support for each RDT event. To see a list of these events, run:

```
sep -el rdt
```

For example,

```
$ sep -el rdt
UNC_CMT_L3_CACHE_OCCUPANCY
UNC_MBM_TOTAL_EXTERNAL_BW
UNC_MBM_LOCAL_EXTERNAL_BW
UNC_RDT_PQR_ASSOC
UNC_CAT_L3_MASK
```

Collect RDT Events

To collect RDT events, run:

```
sep -start -d <duration> -ec <RDT Event List>
```

For example,

```
$ sep -start -d <duration> -ec <RDT Event List>

$ sep -start -d 5 -ec
UNC_CMT_L3_CACHE_OCCUPANCY,UNC_MBM_TOTAL_EXTERNAL_BW,UNC_MBM_LOCAL_EXTERNAL_BW,UNC_RDT_PQR_ASSOC,
UNC_CAT_L3_MASK
```

RDT Standalone Mode

To profile cache usage by hardware core, include the `-rdt-auto-rmid` option. The SEP tool assigns the core ID for each core as the RMID.

```
sep -start -d <duration> -ec <RDT Event List> -rdt-auto-rmid
```

Uncore Data Collection

The uncore subsystem is shared by more than one physical core in a processor package. Its components vary across platforms. Some of them include the Last Level Cache (LLC), Intel® QuickPath Interconnect (Intel® QPI) link logic, and Integrated Memory Controllers (IMC). SEP can leverage the performance monitoring capabilities that these uncore components sometimes provide. Certain uncore types have more than one unit present in the die. For example, there can be more than one IMC in a package. In such a case, SEP can collect performance data on each uncore unit.

To list all core and uncore PMU types supported and available on the current machine, use the following command:

```
sep -pmu-types available
```

To list events from a specific PMU type available in the system, provide the name of the PMU following `-el` command.

For example:

```
sep -el <pmu-type>
```

SEP polls uncore counters periodically (every 10 ms by default) to collect uncore performance data. The default time to poll can be changed using `-uem` option. In the following example, SEP polls counters every 20 ms:

```
sep -start -d 10 -uem timer=20 -ec
"UNC_CBO_CACHE_LOOKUP.READ_I,UNC_CBO_CACHE_LOOKUP.WRITE_I"
```

In case of uncore data collection, the tool uses time-trigger based multiplexing to collect data for multiple event groups. SEP alternates uncore groups every 1 s. Refer to the [-uem option](#) to override default options for uncore event multiplexing.

Using SEP

To analyze system or application performance with SEP, follow this usage model:

```
sep <SEP Commands> [SEP Options]
```

where:

- *<command>* - SEP command controlling data collection (start, stop, pause, cancel collection, and so on) and providing help information.
- *[options]* - collection options of the following types:
 - Generic collector options: to specify an application to analyze if any, configure collection-wide parameters as applicable, input/output options, etc.
 - Event-specific options: To configure event-specific parameters, event modifiers are used.

For example, in the SEP command below, `-lbr` option is applied to all events while `:OS=YES` is applied only to the `BR_INST_RETIRED.ALL_BRANCHES` event.

```
sep -start -lbr no_filter -ec
"BR_INST_RETIRED.ALL_BRANCHES:OS=YES, LONGEST_LAT_CACHE.REFERENCE"
```

Internals of Data Collection

When the data collection starts, SEP does the following:

- Executes the specified application, if any, and collects performance data
- Resolves symbol information for user and system modules
- Outputs performance data into a binary `*.tb` file

To view data collected with SEP, use the graphical interface of Intel® VTune™ Profiler. For more information, see [Use Intel® VTune™ Profiler with SEP](#).

SEP generates temporary files during data collection and deletes them at the end of collection. The temporary directory location varies between operating systems. They include the following:

- Linux*/FreeBSD*/Android*: By default, SEP uses the system environment variable `TMPDIR`. If this variable is unavailable, `/data` directory is used to create temporary files in Android* and `/tmp` in the rest.
- Windows*: By default, SEP uses the system environment variable `TEMP`.

To override the default temporary location used by SEP, configure the environment variable `SEP_TMP_DIR` with the desired location.

Launch and Control SEP Collection

This section describes how to launch and control SEP data collection.

Launching SEP Collection

To launch a data collection, use SEP in the following format:

```
sep -start <collection_options>
```

where:

- `-start` is the command to launch the sampling data collector.
- `<collection_options>` are sampling collector configuration options.

When a sampling collection is started, the default behavior of this call is blocking, which means that the control is returned back to the user only after the sampling collection finishes.

You can control the duration of a sampling data collection using one of the following methods, but not both:

- **Specify duration:** SEP runs for the duration specified with the `-d collector` option. You can start an asynchronous session by specifying 0 run duration.
- **Exit collection when application terminates:** SEP runs while the application is running. SEP session stops only when the application terminates. You can stop the sampling session explicitly using the `-stop` command.

The following subsections list the basic tool usage scenarios.

Launching with an Application

SEP enables launching an application using the `-app` option and an optional `-args` switch to specify application arguments.

For example:

```
sep -start -app <full-path-to-application> -args "<list of arguments>"
```

SEP starts the workload and collects data until the application finishes or SEP is stopped using the `-stop` command as follows:

```
sep -stop
```

Launching a Delayed Collection

Start delay is a separate time interval that is not a part of duration. For example, if you have an activity with duration of 60 s and a start delay of 10 s, SEP will start collecting samples after 10 s and run for 60 s, taking a total time of 70 s.

To have the sampling collection delayed, use the `-sd` collector option.

For example, to start a standard 20 s sampling session with a 10 s delay, enter:

```
sep -start -sd 10
```

Launching Collection Indefinitely

Set the duration to 0 s to run sampling indefinitely and optionally use `-nb` to run SEP in the background:

```
sep -start -d 0 -nb
```

This option cannot be used with `-mr` to schedule multiple sampling runs, but it is available in the default event multiplexing mode.

You may stop the sampling activity using the `-stop` command as follows:

```
sep -stop
```

Pausing Collection

This command pauses collection while a sampling run is in progress.

To pause a sampling collection, use the following command:

```
sep -pause
```

When a SEP run is paused, the duration of the run does not change. For example, if a SEP run is started for duration of 60 s and it is paused after approximately 20 s, then the sampling activity will still complete after 60 s, but the data is only collected during the first 20 s before it was paused.

Resuming Collection

This command resumes sampling collection that was previously paused.

To resume a sampling collection, use the following command:

```
sep -resume
```

When the SEP resume command is issued, the collection that was previously paused is resumed, and the sampling data is collected from that time.

Stopping Collection

This command stops the collection and generates the *.tb file.

To stop a sampling collection that was previously started, use the following command:

```
sep -stop
```

Cancelling Collection

This command cancels the collection and discards the data collected.

To cancel a sampling collection that was previously started, use the following command:

```
sep -cancel
```

SEP Commands



This chapter details the SEP collection, informative, and logging commands.

The following table describes the actions associated with collection commands:

Command	Action
<code>-start [SEP Options]</code>	Start collection with given options. For more information, see SEP Options .
<code>-pause</code>	Pause the current collection.
<code>-resume</code>	Resume the current collection that is paused.
<code>-stop</code>	Stop the current collection.
<code>-flush</code>	Flush out and generate an intermediary *.tb in the middle of a continuous profiling collection
<code>-cancel</code>	Cancel the current collection.
<code>-mark</code>	Insert a mark during sampling.
<code>-mark-off</code>	Insert an end marker during sampling.
<code>-reserve</code>	Reserve PMU resources and block usage by other processes.
<code>-release</code>	Release PMU resources previously reserved.

The following table describes the actions associated with informative commands:

Command	Action
<code>-atypelist [-config] [-details] [atype1, atype2, ...]</code>	Get a list of predefined analysis types (<i>atypes</i>). When specified without any other options, the <code>-atypelist</code> command lists all available <i>atypes</i> . Add the <code>-config</code> option to provide a path to the <i>atype</i> configuration file in the SEP installation directory. Add the <code>-details</code> option to also list related events. Specify a comma-separated list of <i>atypes</i> along with the sub-options <code>-config</code> or <code>-details</code> to get information on specific <i>atypes</i> .
<code>-pmu-types [available]</code>	Display the PMU types supported by the platform. Add the 'available' parameter to display PMU types available on the current system. Use the output from this command with <code>-el</code> to generate a list of events supported on the system for the given PMU type.
<code>-version [-display-features]</code>	Display version and build information. Add <code>-display-features</code> option to display supported capabilities in this version of driver.
<code>-platform-info</code>	Display version and build information of the tool along with details about the hardware platform.

Command	Action
-help [-list-event-modifiers] /?	Display help information. Add -list-event-modifiers to get a list of tool supported event modifiers.

The following table describes the actions associated with logging commands:

Command	Action
-dump-driver-log [file_name]	Dump the contents of the sampling driver's internal log to the given file in binary format. Default file name is driver_log.dump if none specified. Example command: sep-dump-driver-log [file_name]
-decode-driver-log [file_name]]	Decode the log buffer dump to text format. Default file to decode would be driver_log.dump if none specified. Example command: sep -decode-driver-log [file_name]
-extract-driver-log <core_dump_input> [out file]	Identifies and extracts the most recent instance of the driver log from the specified uncompressed core dump into the output file. Default output file is driver_log.dump if none specified. Example command: sep -extract-driver-log <core_dump_input> [out_file]

SEP Options

This chapter lists all SEP options and how to use them.

Event Configuration Options

This section lists all event configuration SEP options and how to use them.

-atype <atype name1>, <atype name2>, ...

Pre-defined set of events. *atypes* are tool-defined analysis types that group events related to a given type. Do not use `-ec` when using `-atype`. Only one *atype* can be specified per Performance Monitoring Unit (PMU) type, such as `core` or `imc`.

For example, to collect sampling data related to memory bandwidth use:

```
sep -start -atype memory_bandwidth
```

-atypelist [-config] [-details] [<atype1>, <atype2>,...]

Lists the available pre-defined analysis types (*atype*). This command is used to obtain the list of available *atypes*. The `-atype` option can then be used with one or many *atypes* to run a specific profiles.

- `-config`: provides the path to the *atype* configuration file located in the SEP installation directory.
- `-details`: lists all *atypes* with related events.
- `-details <atype1>, <atype2>`: Lists events for the specified comma-separated list of *atypes* provided with the command.

Example Analysis Type List Commands

Use the following examples to run `atypelist` commands.

List of Available Analysis Types

Provide a list of available analysis types with the following command:

```
sep -atypelist
```

Example output:

```
Atype: bandwidth
Atype: general_exploration
```

List of Available Analysis Types and the Location of the Analysis Type Configuration File

Provide a list of available analysis types and the location of the analysis type configuration file with the following command:

```
sep -atypelist -config
```

Example output:

```
Atype: bandwidth
Config_File:
/home/.../install/sep/release_posix/bin32/../../config/sampling/../../atypes/ivybridge_atype.txt
```

```
Atype: general_exploration
Config_File:
/home/.../install/sep/release_posix/bin32/.../config/sampling/.../atypes/ivybridge_atype.txt
```

List of Events for a Specific Analysis Type

Provide a list of events for a specific analysis type with the following command:

```
sep -atypelist -details general_exploration
```

Example output:

```
Atype: general_exploration
INST_RETIRED.PREC_DIST
BACLEARS.ANY
OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.HITM_OTHER_CORE_0
MACHINE_CLEARS.COUNT
LD_BLOCKS.NO_SR
```

-ec | -event-config [-dc | -data-config <optional-data1>,<optional-data2>...] "<event-name1>":modifier1=val:modifier2=val/constraint1={:modifier3=val:modifier4=val}, "<event-name2>"...

Configure events to sample. Event configuration options begin with `-ec` switch. Specify the events to monitor and embed the event names within double quotes ("). If no events are specified, the platform's fixed events are collected.

Both core and uncore events can be specified to be monitored. However, when user specifies only uncore events in the command line, SEP collects all the fixed core events along with the specified uncore events.

Individual core/uncore event behavior can be modified using event modifiers. The `[:modifier=val]` option enables you to specify individual event modifiers along with the respective values for a given platform. Each event specification is delimited by a comma (,).

For example:

```
sep -start -d 10 -out outfile -ec "CPU_CLK_UNHALTED.THREAD", "UNC_IMC_NORMAL_READS.ANY"
```

All event modifiers supported by SEP are listed in the following sections.

Event Modifiers

Event modifiers are attached to event names delimited by colon (:). They may or may not take values. Where applicable, values are of the following format: `<yes/no>`, `<0/1>`, `<dec/hex values>`. In some special cases explicitly mentioned they can take string values.

:sa | sample-after = <sample after value>

The Sample After Value (SAV) for an event indicates the number of events after which a sample is collected (lower the SAV for an event higher the sampling rate).

To calculate the sample after value for any event:

1. Calculate the targeted (or expected) number of samples:
Targeted Number of Samples = (Sampling Duration / Sampling Interval) * Number of processors
2. Calculate the average number of event counts for a single processor:
Avg. number of event counts = Total event counts across all CPUs / Number of CPUs
3. Finally, compute the SAV:
SAV = Average number of event counts (as in 2) / Targeted number of samples (as in 1).

The minimum value for SAV varies between events. If SAV specified by the user is lower than the default SAV computed by tool, the user specified value gets overridden with the tool computed SAV. The sample after value should not be zero or a negative value.

To specify the sample after value for your sampling collection, use the `:sa` event modifier option. For example, to collect samples after `1000000 CPU_CLK_UNHALTED.THREAD` events, enter:

```
sep -start -ec CPU_CLK_UNHALTED.THREAD:sa=1000000
```

Basic Event Modifiers

The following table lists the basic event modifiers and provides a short description of each modifier.

Modifier	Description
<code>:USR=<yes/no></code>	Specifies that events are counted only when the processor is operating at privilege levels 1, 2, or 3. This flag can be used in conjunction with the OS flag.
<code>:OS=<yes/no></code>	Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
<code>:PRECISE=<yes/no></code>	Enable PEBS capability for suitable events.
<code>:pdir</code>	Forces the collection of reduced skid PEBS on capable events. Append the <code>:pdir</code> modifier to the event name. For example, <pre>sep -start -ec "MEM_UOPS_RETIREED.ALL:pdir"</pre> Use the <code>sep -el -desc</code> command to show the pdir status of each event.
<code>:mg</code>	Available only for core events. It enables the specified event to be collected in all the event groups. For example, <pre>sep -start -ec "MEM_UOPS_RETIREED.ALL:mg, UOPS_ISSUED.ANY,UOPS_ISSUED.STALL_CYCLES,L2_RQSTS.REFERENCES, LONGEST_LAT_CACHE.MISS"</pre> In the example above, <code>MEM_UOPS_RETIREED.ALL</code> will be collected in all event groups. NOTE The modifier can be specified with a maximum of 2 events only. Rest of the events will discard this modifier.
<code>:sample</code>	Use this modifier to configure an event in sampling mode to override the default counting mode in <code>-ebc</code> collection.
<code>:perf_metrics</code>	Enable hardware based top-down metrics. This modifier is ignored on all events except for the fixed event <code>TOPDOWN.SLOTS</code> .
<code>:ocr_msr_val=<value></code>	Override the default offcore MSR programming with the user specified value for the event.

Event Modifiers for Intel® Transactional Synchronization Extensions (Intel® TSX)

The following table lists the event modifiers for supporting Intel® Transactional Synchronization Extensions (Intel® TSX) and provides a short description of each modifier. The Intel® TSX feature is supported on 4th Generation Intel processors and newer.

Modifier	Description
:tx	In Transaction - When this modifier is specified, the sampling data will only include samples that occurred inside an Intel® TSX region, regardless of whether that region was aborted or committed. For example, <pre>sep -start -d 10 -ec "INST_RETIRED.ANY":tx</pre>
:cp	In Check Point - When this modifier is specified, the sampling data will not include samples that occurred inside of an aborted Intel® TSX region. For example, <pre>sep -start -d 10 -ec "INST_RETIRED.ANY":cp</pre>

Advanced Event Modifiers

The following table lists the event modifiers for more advanced users having an understanding on hardware PMU.

Modifier	Description
:ANYTHR=<yes/no>	Sets (yes) or clears (no) the event's Any Thread control bit. A value of "no" causes the event to be counted on a per logical core basis when applicable. A value of "yes" causes the event to be counted on a per physical core basis. Please note that this feature is not supported on 10th generation Intel Core Processors and 3rd generation Intel Xeon Scalable Processors or newer.
:CMASK=<mask value>	Value that will be compared to the count of the specified event during a single cycle per core. If the event count is greater than or equal to this value, the counter is incremented by one. Otherwise the counter is not incremented. The value must be in the range of 0x0 to 0xff.
:e=<yes/no>	Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. For example, <pre>sep -start -ec "MACHINE_CLEARS.COUNT:cmask=1:e=yes"</pre>
:inv=<yes/no>	When the invert flag is set, inverts :c <cmask> comparison, so that both greater than or equal to and less than comparisons can be made (<0>: greater than equal to comparison, <1>: less than comparison).

Modifier	Description
	Invert flag is ignored when :c<cmask> is programmed to 0. A value of 0 disables invert, and 1 enables it.
:en=<yes/no>	Enable or disable performance counter using yes/no values, respectively.
:event_select=<value>	Input event code to program an event counter for collection.
:int=<yes/no>	Enable to disable interrupt flag using yes/no values, respectively.
:pc=<yes/no>	When set, enables toggling of PMi pin for each event occurrence rather than during counter overflow.
:request=<request name as string>	Programming request type in the off-core response facility for a transaction request to the uncore. The request type specification must be accompanied by a response type.
:response=<response name as string>	Programming response type in the off-core response facility for a transaction request to the uncore. The response type specification must be accompanied by a request type.
:freq=<value>	Applicable to PCU events. When given, the PCU filter register is programmed with this frequency value.
:link=<value>	Applicable to server uncore CBO/CHA units to program filter links.
:occ_inv=<0/1>	Applicable to uncore PCU events. Sets or unsets inversion of uncore PMON_CTLx register for occupancy events.
:occ_e=<0/1>	Applicable to uncore PCU events. When set counter registers transitions from no event to an incoming event for PCU's occupancy events each cycle.
:port=qpi<port number>	Applicable to uncore Intel QPI events to configure port number.
:t=<threshold value>	Threshold programming for uncore PMON_CTLx register. For events that increment more than 1 per cycle, if the threshold value is greater than 1, the data register will accumulate instances in which the event increment is >= threshold.
:rx_match=<value> :rx_mask=<value> :tx_match=<value> :tx_mask=<value>	Modifiers are all applicable to uncore Intel QPI for programming filter registers.
:state=<value>	Applicable to uncore CHA device to program state bit field of filter MSR_0.
:tid=<value>	Applicable to uncore CBO device to program tid bit field of filter MSR_0.
:filter0=<value>	Applicable to Uncore CBO/CHA devices to program filter MSR_0.

Modifier	Description
:filter1=<value>	Applicable to Uncore CBO/CHA devices to program filter MSR_1.
:nc=<value>	Applicable to uncore CBO/CHA devices to filter non-coherent requests by programming nc bit field of filter MSR_1.
:opc=<value>	Applicable to uncore CBO/CHA devices to filter events based on their OPCODE by programming opc bit field of filter MSR_1.
:nid=<value>	Applicable to uncore CBO/CHA devices to filter events by programming nid bit field of filter MSR_1.
:ccst_debug=<value>	Applicable to PCU for programming debug MSR.
:umask_ext=<value>	Enables setting extended umask bits in the counter control register when used with applicable uncore events.

-em | -event-multiplexing [trigger=<fixed counter> factor=<value>]

Enabling core event multiplexing. The `-em` option is used to enable event multiplexing on core. Event multiplexing is the ability to sample multiple groups of events within a single sampling run.

When the number of specified events in the command line cannot be programmed in a single run using available counters on the platform, SEP performs event multiplexing. The tool splices the time of event groups on counters for the duration of collection. The events are grouped such that all events in a group can be scheduled for collection simultaneously. SEP uses a trigger event to swap groups and the default trigger event is `CPU_CLK_UNHALTED.REF_TSC`.

The trigger event name can be changed using `trigger` option and must be an architectural fixed event. With `factor` option, user can control frequency of swapping groups. The default swap frequency is every 50 samples of the trigger event. Collection continues with each group swapped in a round-robin fashion until workload terminates or until end of specified sampling duration.

In the following example, SEP schedules two groups with events:

```
UOPS_ISSUED.ANY,UOPS_ISSUED.STALL_CYCLES,L2_RQSTS.REFERENCES, LONGEST_LAT_CACHE.MISS, LONGEST_LAT_CACHE.REFERENCE in group 1 and SW_PREFETCH_ACCESS.NTA in group 2.
```

The two groups are multiplexed every 50 samples of the default trigger event:

```
sep -start -d 20 -em -ec
"UOPS_ISSUED.ANY,UOPS_ISSUED.STALL_CYCLES,L2_RQSTS.REFERENCES, LONGEST_LAT_CACHE.MISS, LONGEST_LAT_CACHE.REFERENCE, SW_PREFETCH_ACCESS.NTA"
```

In the following example, SEP schedules same two groups, and they are multiplexed every 75 samples of the `INST_RETIRED.ANY`:

```
sep -start -d 20 -em trigger=INST_RETIRED.ANY factor=75 -ec
"UOPS_ISSUED.ANY,UOPS_ISSUED.STALL_CYCLES,L2_RQSTS.REFERENCES, LONGEST_LAT_CACHE.MISS, LONGEST_LAT_CACHE.REFERENCE, SW_PREFETCH_ACCESS.NTA"
```

Event multiplexing is enabled by default on all platforms.

-experimental

Collect experimental event data. Experimental events are those events that have not been validated in hardware. When used with `sep -e1`, all available experimental events are displayed along with regular events. Use `-experimental` with `sep -start` command to collect data for experimental events:

```
sep -start -d 10 -ec "LONGEST_LAT_CACHE.MISS" -experimental
```

Collection Options

This section lists all collection SEP event options and how to use them.

-app <full-path-to-the-application>[-args <"list of application arguments">]

Specify the application to be launched for data collection with SEP. You need to specify the full path to the application. If the application takes arguments, optionally the list of arguments can be specified using `-args` option.

For example, on Windows*:

```
sep -start -app C:\Users\test\sample.exe
sep -start -app C:\Users\test\sample.exe -args "1 10 5"
```

The `-d` | `-duration` option is not supported with this option. The SEP data collection continues indefinitely until the launched application terminates or SEP is stopped explicitly with the `sep -stop` command.

-cm | -cpu-mask <processor numbers>

Specify processors from which data is collected. Use the `-cm` option to specify a CPU mask that defines the processors from which you want to collect data. Enter the processor numbers or processor ranges separated by commas (,).

For example:

```
sep -start -cm 2-5,10,12-14
```

In this example the only following processors are sampled: 2, 3, 4, 5, 10, 12, 13, 14.

-cp | -continuous-profiling

Enable continuous profiling mode. SEP is run indefinitely, but the tool does not save all sampling data generated from the start of the collection. Data snapshot worth more recent few seconds (~10 s) can be obtained at any time using the `sep -flush` command, which generates an intermediary `*.tb` file. Use the `sep -stop` command to stop collection and generate a final `*.tb` file.

For example:

```
sep -start -cp -ec "LONGEST_LAT_CACHE.REFERENCE, LONGEST_LAT_CACHE.MISS" -nb
```

To flush out a `*.tb` file, use (can be invoked any number of times in an on-going continuous profiling session):

```
sep -flush
```

To stop collection use:

```
sep -stop
```

-d | -duration <in seconds>

Specify duration for the sampling collection. Use the `-d` option to specify duration for the sampling collection. The default is 20 s. Set duration to 0 s to run collection for an indefinite amount of time until it is stopped explicitly with the `sep -stop` command.

-ebc | -event-based-counts

This option is used to collect event-based sampling data and event counts data for the list of events specified in `-ebc` option. The event count information is collected and added at the end of each sample.

When the `-ebc` option is specified along with `-em -trigger <event_name>`:

- The trigger event specified in `-em trigger="<trigger-event>"` is used to trigger the interrupt.
- No other events interrupt or overflow. The counts for all events are recorded when the trigger event overflows along with other sampling profile data.

For example:

```
sep -start -d 10 -ec
"BR_INST_RETIRED.MISPRED","L1D_SPLIT.LOADS","L1D_SPLIT.STORES","MUL","DIV","L1D_ALL_REF","L1D_REP
L" -ebc -em trigger="INST_RETIRED.ANY" factor=1 -out data1
```

You can configure the `-ebc` option to generate sampling data for some events and counts for others. In this mixed mode:

- The trigger event is configured to overflow and interrupt. The counts for EBC events are recorded from the trigger event interrupt.
- Apart from the trigger event that is sampled by default, the core events that are specified with `:sample` modifier in the command line are configured to interrupt and generate sample records.

For example:

```
sep -start -d 5 -ebc -em trigger=CPU_CLK_UNHALTED.REF_TSC -ec
"BR_INST_RETIRED.ALL_BRANCHES:sample,DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK,L2_RQSTS.DEMAND_DATA_RD_
HIT -out data1
```

In the regular and mixed EBC modes:

- Event multiplexing (EM) groups are scheduled in round-robin fashion. The group swaps follow the general multiplexing rules specified in the introduction to event based sampling.
- Only fixed counter events can be used as trigger events in EM mode.
- The data is available in the `*.tb` file.
- Event group id is added as a separate column before the event counts column for each sample.

NOTE In the mixed EBC mode, only the trigger event sample record has valid counts for `-ebc` events. Other samples show zero counts for `-ebc` events.

-mr | -multi-run

When there are multiple event groups, `-mr` is used to schedule one sampling run for each group. This option overrides the default event multiplexing in case of multiple event groups and does a separate application run per event-group.

For example:

```
sep -start -d 20 -mr -ec "INST_RETIRED.ANY:sa=2000000",
"CPU_CLK_UNHALTED.CORE","CPU_CLK_UNHALTED.REF_TSC","INST_RETIRED.ANY_P","UOPS_RETIRED.ANY"
```

NOTE

- The `-em` and `-mr` options cannot be specified together.
 - `-mr` cannot be specified with indefinite SEP runs that use `-d0`.
-

-nb | -non-blocking

Enable non-blocking mode. SEP starts in the background. You regain control after data collection starts. When SEP is in background the default behavior is "blocking."

-osm | -os-mode

Enable sampling for operating system processes only. This option is used to configure sampling operating system processes only.

-sam | -sample-after-multiplier <value>

Specify sample after value multiplier. Use the `-sam` option to specify a value between 0.01 and 100.0 by which the Sample After Values (SAV) coming in from the event files of the platform are scaled/multiplied.

-sd | -sampling-delay <delay in seconds>

Specify delay of data collection. Use the `-sd` option to specify the number of seconds to delay sampling while your application executes. The default is 0 s.

The sampling delay is a separate time value that is not a part of collection. For example, if you have an activity with duration of 60 s and a start delay of 10 s, SEP starts collecting samples after 10 s and runs for 60 s, taking a total time of 70 s.

-sp | -start-paused

Start data collection in paused mode. Use the `-sp` option to start data collection in pause mode. To start collection, use the `-resume` command.

-uem | -uncore-event-multiplexing [factor=<value>][timer=<value in ms>]

Enable uncore event multiplexing. Use the `-uem` option to enable event multiplexing for uncore. Event multiplexing is the ability to sample multiple groups of events within a single sampling run.

The uncore event multiplexing is time-trigger based. SEP switches groups in each uncore unit every 1 s when needed. Multiplexing can be configured using `factor` and/or `timer` options. The group switches happen at the rate of `factor_value*timer_value`.

With `factor` option, user can specify the number of times each uncore group polls the performance counters before switching over to the next group. In this case, frequency of polling is the default value of 10 ms. When not specified, the default factor value used by SEP is 100.

With `timer` option, user can specify the frequency of polling and override the default value of 10 ms.

In the following examples, SEP schedules two groups for CBO and one group for IMC, assuming the total available counters are both for CBO and IMC. In the following example, SEP uses default factor and timer, switching groups for CBO every 1 s:

```
sep -start -d 10 -uem -ec
"UNC_IMC_DRAM_GT_REQUESTS,UNC_IMC_DRAM_IA_REQUESTS,UNC_CBO_CACHE_LOOKUP.READ_I,UNC_CBO_CACHE_LOOK
UP.WRITE_I,UNC_CBO_CLOCKTICKS"
```

In the following example, SEP uses a factor of 200 and default timer of 10 ms, switching groups for CBO every 2 s:

```
sep -start -d 10 -uem factor=200 -ec
"UNC_IMC_DRAM_GT_REQUESTS,UNC_IMC_DRAM_IA_REQUESTS,UNC_CBO_CACHE_LOOKUP.READ_I,UNC_CBO_CACHE_LOOK
UP.WRITE_I,UNC_CBO_CLOCKTICKS"
```

In the following example, SEP uses a timer value of 20 ms and default factor of 100, again switching groups for CBO every 2 s:

```
sep -start -d 10 -uem timer=20 -ec
"UNC_IMC_DRAM_GT_REQUESTS,UNC_IMC_DRAM_IA_REQUESTS,UNC_CBO_CACHE_LOOKUP.READ_I,UNC_CBO_CACHE_LOOK
UP.WRITE_I,UNC_CBO_CLOCKTICKS"
```

-um | -user-mode

Enable sampling data collection for user-mode processes only. This option is supported on Windows* and Linux* operating systems only.

-rdt-auto-rmid

Use the `-rdt-auto-rmid` option to run SEP in the RDT standalone mode. This action sets the RMID to all cores.

Specify an RDT monitoring event with this option.

Processor Event Based Sampling (PEBS) Options

This section lists all options supported by SEP for PEBS collection. The tool performs PEBS collection by default on PEBS-capable events and replaces regular sampling data with accurate Instruction Pointer (IP) and Time Stamp Counter (TSC) from PEBS fields. By default, the tool collects and reports data latency fields for applicable memory events.

-apc [[basic],[mem],[gpr],[lbr:[# of entries]],[xmm]]

Adaptive PEBS capture on capable platforms. For PEBS-capable events provided by the user in the command line, SEP collects PEBS records in the sampling data. Users can select zero or more fields. Sampling data for each selection type contains related PEBS fields. For multiple selection types, sampling data will contain all relevant fields. For information on events that are PEBS capable for a given platform, see the Intel Software Developers Manual.

- `-apc` without any selection type, the sampling data contains basic PEBS record which includes record layout, applicable counters, PEBS eventing IP, PEBS TSC.
- `-apc mem` includes memory latency fields
- `-apc gpr` includes general purpose register fields
- `-apc lbr[:depth=<# of lbr entries>]` includes last branch record fields. By default, the tool collects maximum available `lbr` records for the platform. Optionally, when number of `lbr` entries is given, the tool programs PEBS control register to only fetch requested number of entries (min# - 1, max# - 32. # of entries is recommended to be a power of 2).
- `-apc xmm` includes XMM register fields.

If the user selects PEBS-capable events in the command line without using the `-apc` option, the tool still performs PEBS collection on those events and replaces non-PEBS eventing IP and TSC in sampling data with PEBS eventing IP and TSC. No other fields added to the `*.tb` file.

Example Adaptive PEBS Commands

The following examples demonstrate a few combinations of the `-apc` command:

```
sep -start -apc -ec "INST_RETIREDD.ANY, BR_INST_RETIREDD.ALL_BRANCHES"
sep -start -apc basic,mem,gpr,xmm,lbr -ec "INST_RETIREDD.ANY, BR_INST_RETIREDD.ALL_BRANCHES"
```

-fpc | -full-pebs-capture

Capturing all applicable PEBS information. For PEBS-able events provided in command line, SEP collects full PEBS record in the sampling data. The PEBS data collected includes all the fields relevant to the hardware platform. The sampling fields may include eventing IP, Time Stamp Counter value, General Purpose Registers (GPRs), etc:

```
sep -start -d 10 -fpc -ec "BR_INST_RETIREDD.ALL_BRANCHES"
```

If this option is used in Adaptive PEBS capable platforms, SEP collects the basic PEBS record and general purpose register fields. Refer to the `-apc` section for more information on Adaptive PEBS collection.

-multipebs <# of PEBS records>

Multiple PEBS mode. Configure PEBS buffer threshold to cause an interrupt(PMi) after accumulating specified number of PEBS records as opposed to interrupting for each event overflow. When used with `-fpc` or `-apc` option, SEP generates all PEBS fields relevant to the given platform.

For example:

```
sep -start -d 10 -multipebs 10 -fpc -ec "BR_INST_RETIRED.ALL_BRANCHES"
```

-virt-phys-translation

Virtual to physical address translation of `DataLA` field of PEBS record. The data linear address field in PEBS record contains the source address of the load or the destination address of the store. With this option, the virtual address obtained in PEBS record is translated to physical address by SEP and made available in the `*.tb` file.

Last Branch Records (LBRs) and Callstack Options

This section lists all options supported by SEP for LBR collection.

-apc lbr[:depth=# of entries] [-lbr <filter_name>] [-lbr-filter <filter1>:<filter2>...]

Adaptive PEBS-based LBR capture. In adaptive PEBS capable platforms, branch records can be collected using PEBS flow. To configure collection of LBRs with an optional entry of number of records, use `-apc lbr[:depth=# of branch records]`. To filter `lbr` records use the `-lbr` or `-lbr-filter` options. Refer above for available filter modes.

For example:

```
sep -start -apc lbr:depth=16 -lbr no_filter -ec "BR_INST_RETIRED.ALL_BRANCHES"
```

-callstacks

Collect execution callstack information (FreeBSD* only). This option is used to capture callstack execution path in case of FreeBSD*. The tool programs the hardware LBR facility to collect callstacks. Use option `-lbr call_stack` to capture callstacks in all operating systems.

In the following example, SEP collects LBR callstacks along with sampling data:

```
sep -start -callstacks -ec "BR_INST_RETIRED.ALL_BRANCHES"
```

Another way to obtain callstacks would be to use:

```
sep -start -lbr call_stack -ec "BR_INST_RETIRED.ALL_BRANCHES"
```

-lbr <capture_mode>

Collect LBR information with given mode. SEP defines a set of predefined modes that capture specific set of branches.

The following LBR capture modes are supported in SEP:

- `no_filter` - Captures all branches
- `near_call` - Captures near relative and near indirect calls
- `near_call_ret` - Captures `near_call` branches along with near return calls (available only on Intel Atom® processors)
- `call_stack` - Captures call stack information

In addition to the capture mode, you can also filter by the following:

- `:usr` - Captures only user mode branches
- `:os` - Captures only operating system mode branches

For example:

`-lbr call_stack:os` captures only operating system mode call stack information.

The supported capture modes depend on the architecture. SEP will print a warning message if a user-specified capture mode is not supported on the platform that is running SEP. This option cannot be used with `-lbr-filter`

-lbr-filter <filter1>:<filter2>:<filter3>

Enable LBR Filtering in sampling. With this option, the user can control which set of branches are filtered out from the collection. The user can specify one or more filter names separated with a colon (:).

SEP supports the following filter modes:

- JCC - Filter conditional branches
- NEAR_REL_CALL - Filter near relative calls
- NEAR_IND_CALL - Filter near indirect calls
- NEAR_RET - Filter near returns
- NEAR_IND_JMP - Filter near unconditional indirect jumps except near indirect calls and near returns
- NEAR_REL_JMP - Filter near unconditional relative branches except near relative calls
- FAR_BRANCH - Filter far branches

For example:

```
sep -start -ec "CPU_CLK_UNHALTED.CORE" -lbr-filter JCC:FAR_BRANCH
```

This filters out conditional and far branches from the LBR information. SEP does not support collecting LBR information on Fixed Counter events. Be sure to specify at least one General Purpose event in the event configuration to trigger LBR collection.

P-STATE Options

-p-state

Collects MPERF, APERF, and all fixed register counts on PMI trigger. This option can be used independently of all other options.

The APERF/MPERF ratio provides actual CPU performance over marked (rated) performance, which is useful in performance and power measurements.

This option also counts fixed counter events `CPU_CLK_UNHALTED.THREAD/CORE` and `INST_RETIRED.ANY` on PMI trigger of `CPU_CLK_UNHALTED.REF_TSC` event.

This feature provides an accurate P-State/Turbo-State frequency Profile and CPI value.

Sample run:

```
sep -start -d 10 -ec <event_list> -p-state -out test
```

Sample output:

SampleID <...>	Module Name	Process Name <...>	Time (msec)	MPERF	APERF
INST_RETIRED.ANY	CPU_CLK_UNHALTED.THREAD				

Output Options

The following options are used to configure SEP output.

-of | -options-from-file <file name>

Read SEP options from a file. This option is used to specify a file from which the SEP options are read. SEP reads the options from the specified file and applies them. The options can be specified in the same line or multiple lines.

For example, this is the content of the `my_clocks.txt`:

```
-d 10
-ec CPU_CLK_UNHALTED.THREAD:sa=1000000
-out clock_out
```

You can get the same results using either of the following command lines:

```
sep -start -of my_clocks.txt
```

or

```
sep -start -d 10 -ec CPU_CLK_UNHALTED.THREAD:sa=1000000 -out clock_out
```

Command-line options will override options from a file.

-out | -output-file <file name>

Specify the file name for the output file. This option is used to specify the name of the output file where the data is written. For a sampling run, the extension is `*.tb`.

If the option is not specified, the base file name of the output file begins with `tbs` followed by a string of ten random digits.

In the case of multiple runs, an output file is generated for each run and the specified file name is appended with a unique identifier to generate distinct file names.

For example, the name `foo` is saved as `foo_001.tb7`, `foo_002.tb7`.

-verbose

Display information on actions performed during collection.

Use Intel® VTune™ Profiler with SEP

6

Intel® VTune™ Profiler includes hardware event-based sampling analysis types that provide a way to run a SEP collection using a graphical user interface (GUI). These Intel® VTune™ Profiler analysis types are relevant to SEP:

- **Hotspots** (hardware event-based sampling mode): Event-based sampling analysis that monitors all the software executing on your system, including the operating system modules.
- **Microarchitecture Exploration**: Event-based analysis that helps identify the most significant hardware issues affecting the performance of your application.
- **Memory Access**: Event-based analysis that measures a set of metrics to identify memory access related issues (for example, specific to NUMA architectures).
- **Custom Analysis**: User-created analysis types that can be based on the available collection types or based on the existing predefined analysis configurations.

See additional information about each analysis type in the Intel® VTune™ Profiler help.

The SEP drivers are automatically installed with Intel® VTune™ Profiler. If the driver installation was unsuccessful, installation steps are available in the Intel® VTune™ Profiler Installation Guide for your operating system.

You can import the *.tbb result files generated by a SEP command into Intel® VTune™ Profiler to view in a GUI.


Hotspots Analysis in Hardware Event-based Sampling Mode

To view the default events selected in Intel® VTune™ Profiler:

1. Click **Configure Analysis** from the VTune Profiler toolbar.
2. In the **HOW** pane, select the **Hotspots** analysis type.
3. Select **Hardware Event-based Sampling** mode.
4. Expand the **Details** section to view the default events selected for the configured analysis type.

For Hotspots with hardware event-based sampling mode enabled, the default events include CPU_CLK_UNHALTED.REF_TSC, CPU_CLK_UNHALTED.THREAD, and INST_RETIRED.ANY.

HOW



Hotspots

⋮
📄

Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. [Learn more](#)

User-Mode Sampling ?

Hardware Event-Based Sampling ?

CPU sampling interval, ms

Collect stacks

Show additional performance insights

▼ Details

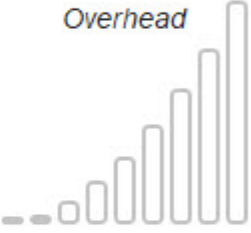
Analyze I/O waits

Collect I/O API data

Collect stacks

Stack size, in bytes

Overhead



Click the Start button to start the collection or the Start Paused button to restart the collection if collection has been paused. After collection begins, it can be paused, resumed, or stopped.

The Hotspots with hardware event-based sampling analysis type is equivalent to running a SEP collection in a non-blocking mode.

For example:

```
sep -start -out test -d 0 -nb
```

While the SEP collection is running, you can pause and resume the collection using the `-pause` and `-resume` commands.

For example:

```
sep -pause
sep -resume
```

The collection can be stopped using the `-stop` option.

For example:

```
sep -stop
```

Microarchitecture Analysis

The Microarchitecture Analysis category includes Microarchitecture Exploration and Memory Access, which are related to the `-atypelist` option in SEP. As with Hotspots, the Details section of each of these analysis types lists the events collected by the type. For example, the Microarchitecture Exploration analysis type lists all hardware events related to general performance issues.

The available analysis types for the current platform can be listed using the following command:

```
sep -atypelist
```

After determining the available analysis types, you can run a command to analyze the types. For example, the following command would be the same as the Microarchitecture Exploration analysis type in Intel® VTune™ Profiler:

```
sep -start -atype microarchitecture_exploration -out test
```

You can collect more than one type of microarchitecture analysis type at once by separating each type with a comma (,). For example, the following command would be the same as both Microarchitecture Exploration and Memory Access:

```
sep -start -atype microarchitecture_exploration, memory_bandwidth -out test
```

Custom Analysis Type

Users can create custom analysis types in Intel® VTune™ Profiler that use hardware event-based sampling. Additional information and instructions for creating a custom analysis type are available in the *Custom Analysis* topic in the [Intel® VTune™ Profiler help](#).

A custom analysis type is equivalent to customized events in SEP using the following option:


```
-ec "<event1>, <event2>, ..."
```

For example, the following command starts a collection using the default duration of 20 s with two custom events:

```
sep -start -ec "BR_INST_EXEC.ALL_BRANCHES, CYCLE_ACTIVITY.CYCLES_NO_EXECUTE" -out test
```

Viewing SEP Results in Intel® VTune™ Profiler

The *.tb result files generated by a SEP collection can be imported and viewed in Intel® VTune™ Profiler.

Click the **Import** button () on the toolbar or select the **Import Result** action from the action drop-down list to open the Import page. Click Browse, select the *.tb or *.tb file, and then click **Import**.

The result file opens in the Hardware Events viewpoint. Additional information about this viewpoint, as well as information about interpreting result data, is available in the [Intel® VTune™ Profiler help](#).

Hardware Events viewpoint (change) ?

Collection Log Analysis Target Analysis Type Summary Event Count Sample Count Caller/Callee Top-down Tree Platform

Elapsed Time [Ⓜ]: 4.995s

- CPU Time [Ⓜ]: 38.383s
- Total Thread Count: 42
- Paused Time [Ⓜ]: 0s

Hardware Events

Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
CPU_CLK_UNHALTED.REF_TSC	130,502,195,753	65,251	2000003
CPU_CLK_UNHALTED.THREAD	134,300,201,450	67,150	2000003
INST_RETIRED.ANY	49,774,074,661	24,887	2000003

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

- User Name: ██████████
- Operating System: Red Hat Enterprise Linux Server release 6.2 (Santiago)
- Computer Name: ██████████
- Result Size: 83 MB
- Collection start time: 04:35:48 08/07/2015 UTC
- Collection stop time: 04:35:53 08/07/2015 UTC

CPU

- Frequency: 3.4 GHz
- Logical CPU Count: 8