Sampling Enabling Product

User's Guide

Version: 1.9

Legal Information

Contents

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 `-atypes` and `-fpc` 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 `-atypes` option with new `-atypelist` 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 -cpu-mask and -lbr-filter examples.	June 2017
1.9	Updated -pmu-types description.	November 2017

1 About this Document

This document explains the various options that the Sampling Enabling Product (SEP) provides and how to use them to collect performance data on your application.

SEP is a standalone command-line tool that provides the event based sampling (EBS) and counting functionality on a local system. The hardware based sampling is a low-overhead, system-wide profiling that helps identify which modules and functions are consuming the most time, giving a detailed look at the operating system and application. SEP enables you to configure the data collection, perform the system-wide profiling and store the results in a .tb7 file.

You can import the .tb7 file into the Intel® VTune™ Amplifier and display the information graphically.

NOTE: SEP and Intel VTune Amplifier will continue to support the .tb6 format for results generated with the previous releases of SEP. Unless otherwise indicated, use the latest drivers.

1.1 Intended Audience

Read this document if you are a software engineer interested in monitoring the performance of your software on IA-32 or Intel 64 systems. For the full list of supported systems, see the SEP README.txt file.

1.2 Conventions and Symbols

The following conventions are used in this document.

Table 1 Conventions and Symbols used in this Document

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Indicates the exact characters you type as input. Also used to highlight the elements of a graphical user interface such as buttons and menu names.
`This type style`	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder.
`[ items ]`	Indicates that the items enclosed in brackets are optional.
`{ item | item }`	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.

1.3 Related Information

- For information on the Intel® VTune Amplifier, go to https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/documentation
- For information on Performance Monitoring Unit (PMU) counters, go to http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- For information on troubleshooting SEP in the Intel Embedded Tool Suite, go to https://software.intel.com/en-us/articles/troubleshooting-issues-with-sep-in-the-embedded-tool-suite-intel-system-studio

2 Installation

SEP can be run on Linux*, Windows*, FreeBSD*, or Android* systems. Follow the installation instructions for the system on which you want to run SEP.

2.1 Installing SEP with Intel VTune Amplifier

SEP is automatically installed with Intel VTune Amplifier. The drivers may need to be manually installed if the installation fails. For more information, see “Building and Managing the Sampling Drivers” in the Intel VTune Amplifier Installation Guide for your operating system or in the VTune Amplifier help.

2.2 Installing SEP without Intel VTune Amplifier

Follow the installation instructions for the system on which you want to run SEP.

2.2.1 Linux*

1. Create a directory where you want to install SEP. For example: `/opt/intel/sep`
2. Copy the SEP install package (`sep_4.*_linux.tar.bz2`) to this directory.
3. Unpack the SEP install Package. For example: `tar -xjzf sep_4.1_linux.tar.bz2`

   The following directories should appear in the specified location for Linux*:

   ```
   bin32
   bin64
   config
   docs
   include
   sepdk
     inc
   sep_vars.sh
   SEP_Users_Guide.pdf
   ```

   The following directories should appear in the specified location for a Linux system with Intel® Many Integrated Core Architecture (Intel® MIC Architecture):

   ```
   bin32
   bin64
     k1om
   config
   docs
   include
   mic_sepdk
     inc
     src
   ```

```
sep_vars.sh
SEP_Users_Guide.pdf
```

4. Install SEP and build the driver.

   — For Linux:

     1. Navigate to the sepdk directory: `cd /opt/intel/sep/sepdk`
     2. Build the SEP driver using the following command: `./build-driver -ni`
     3. Install the SEP sampling driver using the following command: `./insmod-sep -g <user_group>`

   — For Linux with Intel MIC Architecture:

     1. Navigate to the K1om directory: `cd /opt/intel/sep/bin64/k1om`
     2. Install SEP using the following command: `sudo ./sep_micboot_install.sh`
     3. Restart the service using the following command: `sudo service mpss restart`
     4. Run the following command: `micctrl -w`

5. Navigate to the directory from which you want to run SEP.
6. Create a bash shell: `sh`
7. Set up the SEP runtime environment by sourcing the sep_vars.sh file in the current bash shell: `source /opt/intel/sep/sep_vars.sh`
8. Start SEP using the following command: `sep -version`

2.2.2 Windows*

1. Create a directory where you want to install SEP. For example: `C:/Program Files (x86)/IntelSWTools/sep`
2. Extract the SEP install package (`sep_4.*_windows.zip` or `sep4_win_mic.zip`) into this directory.

   The following directories should appear in the specified location for Windows*:

```
bin32
bin64
config
include
docs
sep_vars.cmd
SEP_Users_Guide.pdf
```

   The following directories should appear in the specified location for a Windows system with Intel MIC Architecture:

```
bin32
  k1om
bin64
config
include
docs
```

```
sep_vars.cmd
SEP_Users_Guide.pdf
```

3. Install and register the sampling driver.
   - For Windows: Install and register the sampling driver from the bin32 or bin64 directory using the following command: `sepreg.exe -i`
   - For Windows with Intel MIC Architecture:
     1. Navigate to the bin32/k1om directory.
     2. Install SEP using the following command from within that directory: `sep_micboot_install.cmd`
     3. Run the following command: `net start mpss`
4. Run the following command to set the appropriate environment variables and adjust the path to run SEP in the current shell: `sep_vars.cmd`
5. Start SEP using the following command: `sep - version`

2.2.3 FreeBSD*

1. Create a directory where you want to install SEP. For example: `/opt/intel/sep`
2. Copy the SEP install package (`sep_4.*_freebsd_<version>.tar.bz2`) to this location.
3. Unpack the SEP install Package. For example: `tar -xjzf sep_4.1_freebsd_x86_64_10.tar.bz2`

   The following directories should appear in the specified location:

   ```
   bin
   config
   docs
   include
   sepdk
     fbsd_kernel
       inc
       dev
       modules
   sep_vars.sh
   SEP_Users_Guide.pdf
   ```
4. Navigate to the sepdk/fbsd_kernel/modules directory.
5. Build the sampling driver using the following command: `make`
6. Install the sampling driver using the following commands:

   ```
   make install
   kldload sep pax
   ```

To allow multiple users in the group to access the driver, use the following commands:

- `sudo kenv hw.sep.gid=`getent group <user_group> | awk -F: '{print $3;}'` ;`
- `sudo kenv hw.pax.gid=`getent group <user_group> | awk -F: '{print $3;}'` ;`
- `kldstat -q -m sep && sudo kldunload sep ;`
- `kldstat -q -m pax && sudo kldunload pax;`
- `kldload sep pax`

7. Navigate to the directory from which you want to run SEP.
8. Create a bash shell: `sh`
9. Set up the SEP runtime environment by sourcing the sep_vars.sh file in the current bash shell: `source /opt/intel/sep/sep_vars.sh`
10. Start SEP using the following command: `sep -version`

2.2.4 Android*

The OneAndroid userdebug release contains the sep4_* and sepint4_* drivers. Use the following steps to install the binaries on your Android* device from your host Windows or Linux system.

1. On your host system, create a directory where you want to install SEP. For example:
   - On a Windows host: `C:/Program Files (x86)/IntelSWTools/sep`
   - On a Linux host: `/opt/intel/sep`
2. Extract the SEP install package (Windows: `sep-4.*-for-OneAndroid.zip` Linux: `sep-4.*-for-OneAndroid.tar.bz2`) into this directory.

   The following directories should appear in the specified location:

   ```
   bin32
   bin64
   config
   tools
   sep_android_install.cmd
   sep_android_install.sh
   sepdk
   sep_vars.sh
   ```
3. Open a command prompt and navigate to the directory you just created.
4. Run the following command to install SEP:
   - Windows: `sep_android_install.cmd`
   - Linux: `sep_android_install.sh`
5. Run the following command to connect to your Android device: `adb shell`
6. Navigate to the data/sep directory. For example: `cd /data/sep`
7. Run the following command to set up the environment variables on your Android device: `. ./sep_vars.sh`

3 *Using SEP*

NOTE: SEP and Intel VTune Amplifier will continue to support the .tb6 format for results generated with the previous releases of SEP. Unless otherwise indicated, use the latest drivers.

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

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

where:

- `<commands>` are basic SEP commands controlling data collection (start, stop, pause, cancel collection, and so on) and providing help information
- `[options]` are collector options of the following types:
  - generic collector options used to specify an application to analyze, if any, and configure collection runs
  - event-specific options used to specify processor events to monitor and enable event counting
  - chipset-specific options used to enable chipset profiling and specify events to monitor
  - generic event modifiers used to configure event options

When the data collection is started, SEP does the following:

- Executes the specified application, if any, and collects performance data.
- Resolves symbol information for user and system modules.
- Collects performance data in a `*.tb7` file.

To view data collected with SEP, use the graphical interface of the Intel® VTune™ Amplifier. For more information, see Using Intel VTune Amplifier with SEP.

3.1 Usage Examples

Start a 20-second sampling session with default events and create a randomly named `tbsNNNNNNNNNN.tb7` file:

```
sep -start
```

Start a 10-second sampling session on events <event 1> with sample after value 4515 and <event 2> with the default sample after value, and create a tb6 file named `test.tb7`.

```
sep -start -d 10  -ec "<event 1>":sa=4515, "<event 2>" -out test
```

Start a 20-second counting session using default events, and create a counts data file called `test.txt`.

```
sep -start -out test -count
```

3.2 Starting Data Collection

To start a sampling collection, run:

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

where:

- `-start` is the command launching 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 zero 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.

NOTE: You can specify an application only if SEP is configured to have a single run. In the case of multiple runs (when selected events cannot be grouped in one run), the application launch option will not be valid.

3.2.1 Specifying an Application to Analyze

SEP enables launching an application using the `-app` switch. For example:

```
sep -start -app sample
```

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

```
 sep -stop
```

3.2.2 Running 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 seconds and a start delay of 10 seconds, then SEP will start collecting samples after 10 seconds and run for 60 seconds taking a total time of 70 seconds.

To have the sampling collection delayed, use the `-sd` collector option. For example, to start a standard 20-second sampling session with a 10-second delay, enter:

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

3.2.3 Running Collection Indefinitely

Set the duration to zero to run sampling indefinitely as follows:

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

This option is not available if you selected events that require multiple sampling runs.

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

```
sep -stop
```

3.3 Viewing Collection Status

A successful EBS run gives the following message:

```
SEP is collecting samples based on the following events <event names
separated by a comma >
```

An unsuccessful run gives the following message:

```
SEP failed to start sampling collection due to one of the following
reason(s): <error message>
```

3.4 Ending Data Collection

Typically data collection continues until the application terminates. When SEP is explicitly stopped, it will not terminate the application on the user's behalf. In general, SEP will only launch the application but never forcibly terminate it.

3.4.1 Pausing Sampling Collection

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 seconds and it is paused after approximately 20 seconds, then the sampling activity will still complete after 60 seconds, but the data is only collected during the first 20 seconds before it was paused.

3.4.2 Resuming Collection

To resume a sampling collection that was previously paused, 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.

3.4.3 Stopping Collection

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

```
sep -stop
```

This command stops the collection and generates the .tb6 file.

3.4.4 Canceling Collection

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

```
sep -cancel
```

This command command cancels the collection and discards the data collected.

3.5 Counting Collection

SEP supports two collection modes: sampling and counting. The default collection mode is the sampling mode.

Use the following command to collect data in counting mode:

```
sep -start -c <options>
```

When this option is specified, SEP collects counter data of the specified event for the duration of the collection. When –c option is not specified, SEP collects the data in sampling mode.

3.6 Configuring Data Collection

3.6.1 Specifying Sample After Value

The Sample after value (SAV) is the frequency or the number of events after which SEP interrupts the processor to collect a sample during data collection.

SEP initially computes a default sample after value (SAV) for the default events (Clockticks or CPU Cycles and retired instructions) as follows:

```
SAV for default events = CPU frequency * sample interval in microseconds.
```

The steps to calculate the sample after value for any event are the following:

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 sample after value (SAV) as
   Sample After Value (SAV) = Average number of event counts (as in 2) / Targeted number of samples (as in 1).

The minimum value for SAV is 1. 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
```

3.6.2 Counting Events

To count selected events, use the `-count` option. For example:

```
sep -start -count
```

As a result of this run, a file with the .txt extension (`XXX.txt`) is created and all of the counts for this sampling session (which may have multiple runs) are appended to the file. Even when an SEP activity generates multiple .tb6 files, it will always generate only one `XXX.txt` file, per SEP session. Duration of 0 is allowed for the `-count` and a single run as the `XXX.txt` output file can be written at the time of `sep -stop`.

The zero duration/multiple run restriction also applies to the `-count` option as it is for a regular sampling run.

3.6.2.1 Event Count File Format

When the `-count` option is specified, SEP generates a file that contains the count information for all the events, selected across all the runs, in a session. Only one file will be generated per SEP session. The count file contains two types of output: informational and data.

- Each informational output always starts with `#sep:` keyword as the first argument ($1 Perl notation). The next keyword in the informational output represents the type of the information.
- Currently the supported informational output types are
  - `version` -- version of SEP
  - `header` -- format of the event count data
  - `date` – date and time when the session was initiated
- The event count data follows after the informational output. The format of the data output is specified in the `header` keyword.

3.6.2.2 Fixed Counter Support

On Intel® Core™2 Duo, Intel® Core™ i7, Intel® Atom™ processors and processors based on the Intel® microarchitecture code named Sandy Bridge, three fixed counters are implemented to count "CPU_CLK_UNHALTED.CORE" (CPU_CLK_UNHALTED.THREAD on Intel Core i7 processors), "INST_RETIRED.ANY", and "CPU_CLK_UNHALTED.REF_TSC" events respectively. When one or more of these three events are specified in the event configuration, the corresponding fixed counters are used. On Intel Atom and Intel Core2 Duo processors, the two general counters are still available for counting other Performance Monitoring Unit (PMU) events. Therefore, you can count up to five events in a single run; three fixed events and two general events. On Intel Core i7 processors, there are four general counters available in addition to the three fixed counters. Thus, one can count/sample up to seven events in a single run on Intel Core i7 processors.

In the following example, SEP counts five events in a single run:

```
sep -start -c -ec "CPU_CLK_UNHALTED.CORE","INST_RETIRED.ANY",
"CPU_CLK_UNHALTED.REF_TSC","BR_INST_RETIRED.ANY","SIMD_INST_RETIRED.ANY"
```

4 SEP Commands

This chapter details the SEP commands.

Table 3-1 SEP Commands

Command	Action
`-start  [SEP Options]`	Start collection with given options.
` -atypelist [-config] [-details] [atype1, atype2,…]`	Get a list of Pre-defined analysis types (atypes). When specified without any other options, the –atypes command lists all available atypes. Add the `-config` option to provide a path to the atype configuration file in the SEP installation directory. Add the `-details` option to list all atypes with related events. Specify a comma-separated list of atypes to get events for the specific atypes listed.
`-pause`	Pause the current collection.
`-resume`	Resume the current collection that is paused.
`-stop`	Stop the current collection.
`-cancel`	Cancel the current collection.
`-mark`	Insert a mark during sampling.
`-mark-off`	Insert an end marker during sampling.
`-pmu-types [available]`	Display the PMU types supported by the platform. Add the 'available' parameter to display PMU types available on this system. Use the output from this command with `-el` to generate a list of events supported on the system for the given PMU type.
`-el | -event-list [pmu-type] [-desc]`	List events supported on the platform. Can be filtered by adding a PMU type (from `-pmu-types` command). Add the `-desc` option to print the description of each event.
`-help | /?`	Display help information.
`-version`	Display version or build information.

5 SEP Options

This chapter details the SEP options in alphabetical order.

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

Specify the application to be launched for data collection

Specify the application to be launched with SEP. You need to specify the full path to the application. For example, on Windows* OS:

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

If the application takes arguments, the list of arguments can be specified
using -args option.
```

NOTE: 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.

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

Pre-defined set of events

Pre-defined analysis types (atypes) corresponding to a certain set of events.

Use this instead of `-ec`. Do not use `-ec` and `-atype` together. Only one atype can be specified per Performance Monitoring Unit (PMU) type, such as `core`, or `imc`.

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

List available analysis types

Lists the available pre-defined analysis types (atypes).

Use this command without any options to get the list of available atypes and then use the `-atype` option to run a specific atype-based profile from this list.

- `-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

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

```
sep -atypelist
```

Example output:

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

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

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
```

-c | -count

Count selected events

Use the `-c` option to count the selected events. The event counting results are available in the output file with the .txt extension. This is the format of the `XXX.txt` file:

```
event, cpu #, count, duration
```

Each event and processor pair has a separate line.

-cec | -chipset-event-config

Specify events to monitor

The chipset support in SEP is limited to chipset event counts data.

Chipset event configuration options begin with the `-cec` or `-chipset-event-config` switch. Specify the events(s) to monitor and embed the event names within double quotes("). Each event should be delimited by a comma (,).

To list the supported chipset events, run `sep -el chipset`.

The chipset event counts data is available in a `.csv` file (the file name is the same as the `.tb7` file name except for the file extension). The `.csv` file contains multiple data points with comma separated list of chipset and CPU event counts along with CPU number, timestamp and IP in each line as shown below:

```
CPU number,Timestamp,Instruction Pointer,<event name1>,<event-name2>...,<cpu
event name1>,<cpu event name2>
```

Example Chipset Commands

List chipset event names along with CPU event names:

```
sep -el chipset
```

Run sampling collection for 10 seconds with CPU_CLK_UNHALTED.THREAD as the trigger event and count MCH data for "MCH D clock" event and save the results into the `chipsetdata.csv` file:

```
sep -start -d 10 -ec "CPU_CLK_UNHALTED.THREAD" -cec "MCH D clock" -out
chipsetdata
```

Use a specific sample after value (SAV) for the Clockticks trigger event. The collection includes two chipset events - MCH D Clock and ICH Link Clock:

```
sep -start -d 10 -ec "CPU_CLK_UNHALTED.THREAD":sa=400000 -cec "MCH D
Clock,ICH Link Clock" -out chipsetdata
```

Same as previous command but collect chipset counts only when the trigger is from CPU 0:

```
sep -start -d 10 -ec "CPU_CLK_UNHALTED.THREAD":sa=400000 -cec "MCH D
Clock,ICH Link Clock" -out chipsetdata -cm "0"
```

NOTE: The CPU event names can differ from processor to processor.

NOTE: The chipset support in SEP requires that the chipset counters (for example, CHAP) are enabled through BIOS or some other means. The SEP tool will not work if the chipset counters are disabled.

-cm | -cpu-mask "*processor numbers*"

Specify what processors to collect data from

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.

NOTE: The `-count` option is not supported with this option.

-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 seconds. Set duration to zero to run collection for an indefinite amount of time until it is stopped explicitly with the `sep -stop` command.

-ebc | -event-based-counts [tfactor=<value>]

Enable event-based counting

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

For sampling on Intel® Many Integrated Core Architecture (Intel® MIC architecture), `-ebc` must be used in conjunction with `-mic` flag.

`tfactor` is supported only on first generation Intel® Xeon Phi™ product family formerly code named Knights Corner. The `tfactor` value is used to specify timer interrupt interval. It can be a number in [0,..,15]. If a `tfactor` value is not specified, the default value is 0, indicating a 2 millisecond time interval. The larger the `tfactor` value, the larger the period between interrupts. The first time-out is non-deterministic due to implementation, but all subsequent time-outs are deterministic with the period defined.

NOTE: The `-ebc` option can be used in conjunction with the `-em` option.

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

Configure the events that are sampled

Event configuration options begin with `-ec` switch. Specify the event(s) to monitor and embed the event names within double quotes ("). If no events are specified, the platform's default clockticks and instructions retired events are used.

The `[:modifier=val]` option enables you to specify individual event modifiers along with the respective values for a given platform. The modifiers can be generic to an event as well as specific to a constraint (or an event qualifier). The constraint specific special modifiers appear after

`[/constraint=]`. The modifier values can be in decimal or hexadecimal format. Only specific modifiers accept the value as a string. Each event specification is delimited by a comma (,).

Uncore events can also be specified on supported platforms.

Example command line:

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

The output may look like this (showing one line as an example):

```
00000000 64--0010:0xFFFFFFFF8048D609-0 p-0x00000000 c-03 t-0x00000000 sgno-
0x00000000 ei-00 tsc-0x000103B115A1E4C8 gid-0001 extra_00-0x0000000000000C28
sample
```

In this example, `gid` is group_id=1 and `extra_00` is the uncore event (`UNC_IMC_NORMAL_READS.ANY`). In this interval, the uncore event's count equals 0xC28. The total count for the whole run is equal to the sum of all the `UNC_IMC_NORMAL_READS.ANY` fields in the sfudmp5 output.

Supported modifiers and constraints are listed in the following sections.

Event Modifiers

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

The Sample After Value (SAV) for the event indicates the number of events after which a sample is collected. See Specifying Sample After Value for information on how to compute SAV.

The values for the following attributes can be either in hex or decimal format. For example:

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

Event Modifiers for IA32/Intel® 64 Architectures

The following table lists the event modifiers for IA32/Intel® 64 architectures and provides a short description of each modifier.

Table 4-1 Event Modifiers for the P6 Processor Family

Modifier	Description
`:USR=<yes/no>`	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.
`:OS=<yes/no>`	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.
`:ANYTHR=<yes/no>`	Enables counting of the event (or condition) while in any processor thread on the core are in C0 power state. When not set, the event on the counter will count only while the logical processor (with that event / condition programmed) is in C0 (not halted).

`:PRECISE=<yes/no>`	Enable the PEBS feature for the PEBSable event.
`:CMASK=<mask value>`	Specifies the mask value that will be compared by the logical processor to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. The value must be in the range of 0 to 255. The default value is defined by the event list.
`:e=<yes/no>`	Enables edge detection of the selected microarchitectural condition when set. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by other fields. The default value is defined by the event list. ***NOTE:*** Edge detection is enabled only when CMASK and ANYTHR flags are set. For example, `sep -start -ec "MACHINE_CLEARS.COUNT:anythr=yes:cmask=1:e=yes"`
`:inv=<yes/no>`	When the invert flag is set, inverts :CMASK comparison, so that both greater than or equal to and less than comparisons can be made (<no>: greater than comparison, <yes>: less than comparison). If :CMASK is programmed to 0, INV flag is ignored. The default value is defined by the event list.

Event Modifiers for Transactional Synchronization Extension (TSX)

The following table lists the event modifiers for supporting Transactional Synchronization Extension (TSX) and provides a short description of each modifier. The TSX feature is supported on platforms with Intel Haswell processor family and later.

Table 4-2 Event Modifiers for Transactional Synchronization Extension (TSX) support

Modifier	Description
`:tx`	In Transaction – When this modifier is specified, the sampling data will only include samples that occurred inside a TSX region, regardless of whether that region was aborted or committed. For example, `sep -start -d 10 -ec ‘‘INST_RETIRED.ANY’’:tx`
`:cp`	In Check Point – When this modifier is specified, the sampling data will not include samples that occurred inside of an aborted TSX region. For example, `sep -start -d 10 -ec ‘‘INST_RETIRED.ANY’’:cp`

-em | -event-multiplexing [dts=<*time in milliseconds*>] | [trigger=<*fixed counter*> factor=<*value*>] | [tfactor=<*value*>]

Enable event multiplexing

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

When using this option, you specify a list of events to be counted and a counting interval, based on the sample after value (SAV) of a specified trigger event. The interval is specified in a PMU event unit such as Instructions Retired or Clockticks. The event that dictates the counting interval must be interruptible and is defined as a trigger event. The trigger should be a fixed counter. The default trigger is `CPU_CLK_UNHALTED.CORE` (or `CPU_CLK_UNHALTED.THREAD` for Intel Core i7 processors). The default factor is 50. Only one trigger event is allowed. Events are grouped for each counting interval. Each group of events is counted in a round-robin fashion and this is repeated until the workload terminates or until a specified limit (max samples or sampling duration) is reached.

For example:

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

This SEP run collects the counts of the following events every time the sample value for instructions retired reaches 2000000: CPU_CLK_UNHALTED.CORE, CPU_CLK_UNHALTED.REF_TSC, INST_RETIRED.ANY_P, and UOPS_RETIRED.ANY.

At the end of each interval, SEP collects the values of fixed counters and programmable counters along with other data such as TID, PID, and core ID.

Two trigger modes are available, depending on the platform:

- Time-based event multiplexing for Intel Core and Pentium processor families. For this type of multiplexing, the `dts` (default time slice) specifies the duration between cycles for each event group.
- Trigger-based event multiplexing for all other processors. For this type of multiplexing, the trigger should be a fixed counter:
- On Intel Atom and Intel Core2, the default trigger event is `CPU_CLK_UNHALTED.CORE`.
- On Intel Core i7 and Intel microarchitecture code named Sandy Bridge, the default trigger event is `CPU_CLK_UNHALTED.THREAD`.

The default factor is 50 milliseconds on all platforms.

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

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

- In case of multiple EM groups, groups are scheduled in round-robin fashion. The groups are swapped for every overflow of the trigger event (with factor=1). This can be controlled by changing factor value. The default value for factor is 50. The counters for all non-trigger events are reset to zero when the group is swapped. With factor=1, since the groups are swapped on every interrupt (per CPU), the event counts are the increments since the last interrupt for a specific CPU.
- The data is available in the .tb7 file.
- Only fixed counter events can be used as trigger events in EM mode.
- Event group id is added as a separate column for each sample with a prefix `gid-xxxx`, just before the extra columns.

For example:

```
sep -start -d 10 -ec
"BR_INST_RETIRED.MISPRED","L1D_SPLIT.LOADS","L1D_SPLIT.STORES","MUL","DIV","L
1D_ALL_REF","L1D_REPL" -ebc -em trigger="INST_RETIRED.ANY" factor=1 -out
data1
```

For event multiplexing on Intel MIC Architecture, `-em` must be used in conjunction with `-mic` flag.

`tfactor` is used only on Intel MIC Architecture to specify the interrupt period to switch between groups. `tfactor` can be a number in [0,..,15]. If `tfactor` is not specified, the default value is 4, indicating a 32ms time interval. The larger the `tfactor` value, the larger the period between interrupts.

The first time-out is non-deterministic due to implementation, but all subsequent time-outs will be deterministic with the period defined.

This option is not supported in Intel Xeon Phi Software Development Vehicle (formerly code named Knights Ferry).

Example for Intel Core or Intel® Pentium™ processor families (time-based trigger):

```
    sep -start -em dts=100 -ec "Instructions Retired",
        "Branch Instructions Executed", "Clockticks",
        "L2 Cache Request Misses"
```

Example for Intel Atom or Intel Core2 Duo processor families (event-based trigger)

```
    sep -start -em trigger="CPU_CLK_UNHALTED.CORE" factor=100
        -ec "<event 1>","<event 2>", ... "<event n>"
            "UOPS_RETIRED.ANY:sa=0:int=no"
```

Example for processors with Intel MIC Architecture (timer-based trigger):

```
    sep -start -mic -em tfactor=3
        -ec "<event 1>","<event 2>", ... "<event n>"
```

NOTE: Event multiplexing is enabled by default on all platforms.

fpc <full pebs capture>

Full pebs buffer information

Collects the full PEBS buffer information in the sampling data. The PEBS data collected includes all the fields relevant for the hardware platform.

-lbr <capture_mode>

Collect LBR (Last Branch Records) information

Enables capturing the running trace of most recent branches, interrupts, or exceptions on LBR MSR stack. This option is valid only for the Intel Core2 Duo processor family and for Intel Core i7 processors. 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_jmp` - Captures near_call branches along with near relative and indirect jumps (Available only on the Intel Core processor family)
- `near_call_ret` - Captures near_call branches along with near return calls
- `near_call_jmp_ret` - Captures near_call_jmp branches along with near return calls (available only on the Intel Core processor family)
- `call_stack` - Captures call stack information (available on 4th Generation Intel Core Processors and Intel Atom processors based on Intel microarchitecture code named Silvermont)

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.

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

Enable last branch records (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.

-mic [dev=<id_1> [,<id_2>,...<id_n>]]

Run sampling on Intel MIC Architecture device

If an Intel MIC aArchitecture device is present, all commands are sent to the sampling collector running on that platform.

One or more target Intel MIC Architecture devices can be specified. The default is device 0.

-mr | -multi-run

Enable multiple runs

Enable a separate run per event-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.

[-nb | -non-blocking]

Switch to non-blocking mode

Use this option to switch SEP to 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”.

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

Read SEP options from a file

Use this option to specify a file from which the SEP options are read. SEP reads the options from the specified file and applies them.

The options specified in the file use the command line options. 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 the following two command lines:

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

Or,

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

NOTE: Command-line options will override options from a file.

-osm | -os-mode

Enable sampling for OS processes only

Collect sampling data for operating system processes only.

-out | -output-file <*file name*>

Specify the file name for the output file

Specify the name of the output file where the data is collected. The file extension depends on the type of data collection.

- For a sampling run, the extension is .tb7.
- For a counting run, the extension is .txt.

If the option is not specified, the base file name of the output file starts with `ebs` followed by a string of 10 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`.

-p-state

Collects MPERF, APERF and all fixed register counts on PMI trigger

Use this option to collect the APERF/MPERF MSR data and fixed counter events. 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)  INST_RETIRED.ANY
    CPU_CLK_UNHALTED.THREAD MPERF  APERF
```

-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 are scaled.

-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 sec.

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 seconds and a start delay of 10 seconds, SEP starts collecting samples after 10 seconds and runs for 60 seconds, taking a total time of 70 seconds.

-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.

-um | -user-mode

Enable sampling for user-mode processes only

Enable sampling data collection for user-mode processes only.

This option is supported on Windows* and Linux* OS only.

-verbose

Display information on actions performed during collection

Display information on actions performed during collection.

6 Using Intel VTune Amplifier with SEP

Intel VTune Amplifier includes hardware event-based sampling analysis types that provide a way to run a SEP collection using a graphical user interface. The following VTune Amplifier analysis types are relevant to SEP:

- Advanced Hotspots: Event-based sampling analysis that monitors all the software executing on your system including the operating system modules.
- General 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).
- TSX Exploration: Event-based sampling analysis that is targeted for Intel processors supporting Intel Transactional Synchronization Extensions (Intel TSX).
- TSX Hotspots: Event-based sampling analysis that is targeted for Intel processors supporting Intel TSX.
- Custom Analysis: User-created analysis types that can be based on the available collection types or based on the existing predefined analysis configurations.

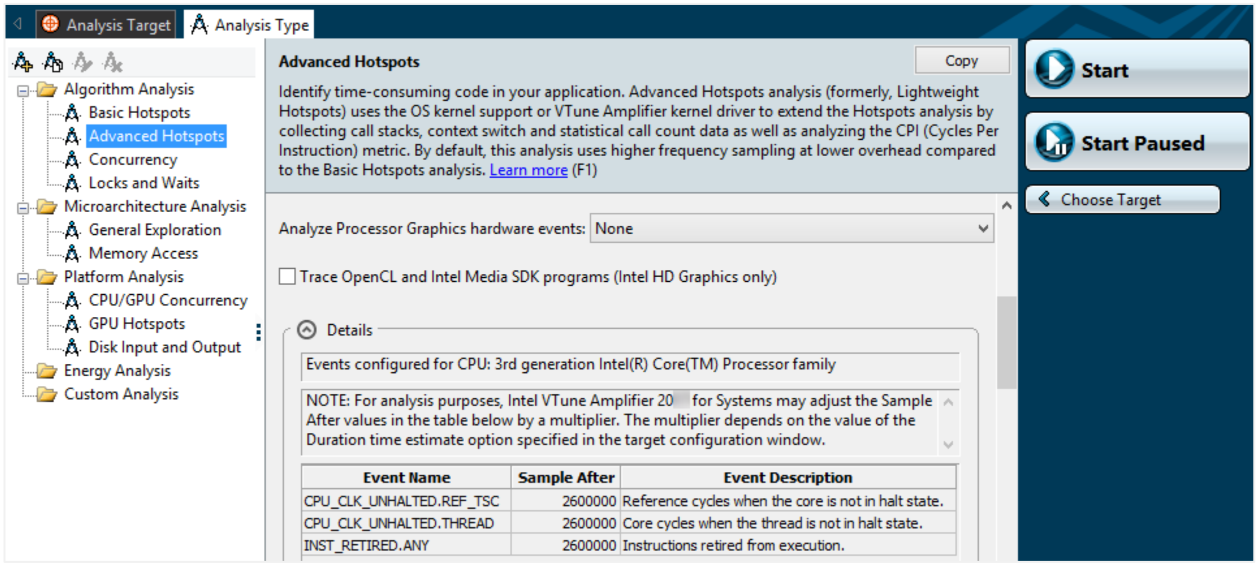
Additional information about each analysis type can be found in the Intel VTune Amplifier help.

The SEP drivers are automatically installed with VTune Amplifier. If the driver installation was unsuccessful, installation steps are available from the “Building and Managing the Sampling Drivers” section in the Intel VTune Amplifier Installation Guide for your operating system.

The .tb6 or .tb7 result files generated by a SEP command can be imported into VTune Amplifier to view in a graphical interface.

6.1 Advanced Hotspots Analysis

Select the **Advanced Hotspots** option from the **Algorithm Analysis** category on the **Analysis Type** tab in VTune Amplifier. The Details section of the Advanced Hotspots analysis shows the default events selected for the analysis type. For Advanced Hotspots, the default events include CPU_CLK_UNHALTED.REF_TSC, CPU_CLK_UNHALTED.THREAD and INST_RETIRED.ANY.



Advanced Hotspots

Identify time-consuming code in your application. Advanced Hotspots analysis (formerly, Lightweight Hotspots) uses the OS kernel support or VTune Amplifier kernel driver to extend the Hotspots analysis by collecting call stacks, context switch and statistical call count data as well as analyzing the CPI (Cycles Per Instruction) metric. By default, this analysis uses higher frequency sampling at lower overhead compared to the Basic Hotspots analysis. Learn more (F1)

Analyze Processor Graphics hardware events: None

Trace OpenCL and Intel Media SDK programs (Intel HD Graphics only)

Events configured for CPU: 3rd generation Intel(R) Core(TM) Processor family

NOTE: For analysis purposes, Intel VTune Amplifier 20 for Systems may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the target configuration window.

Event Name	Sample After	Event Description
CPU_CLK_UNHALTED.REF_TSC	2600000	Reference cycles when the core is not in halt state.
CPU_CLK_UNHALTED.THREAD	2600000	Core cycles when the thread is not in halt state.
INST_RETIRED.ANY	2600000	Instructions retired from execution.

Use the buttons to start the collection or start the collection paused. After collection begins, it can be paused, resumed, or stopped.

The Advanced Hotspots 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
```

6.2 Microarchitecture Analysis

The Microarchitecture Analysis category includes **General Exploration**, **Memory Access**, **TSX Exploration**, and **TSX Hotspots**, which are all related to the –atypelist option in SEP. As with Advanced Hotspots, the Details section of each of these analysis types lists the events collected by the type. For example, the **General 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 **General Exploration** analysis type in VTune Amplifier:

```
sep -start -atype general_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 **General Exploration** and **Memory Access**:

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

6.3 Custom Analysis Type

Users can create custom analysis types in VTune Amplifier 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 VTune Amplifier help.

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

```
-ec ‘‘<event1>, <event2>, …’’
```

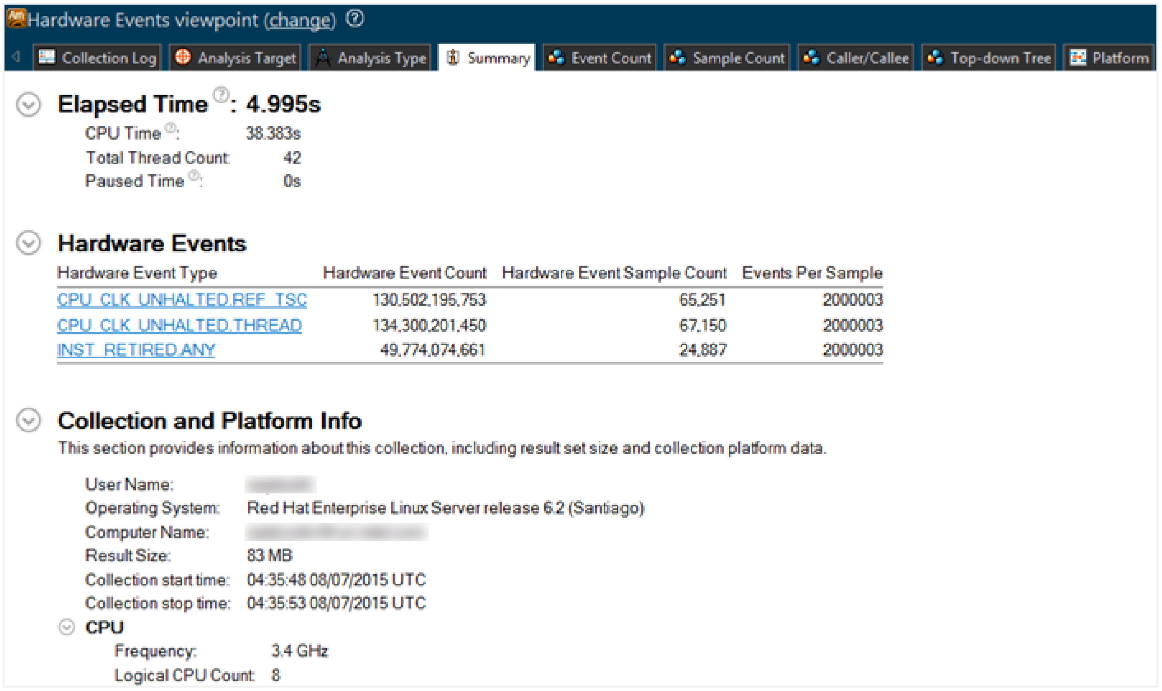
For example, the following command starts a collection using the default duration of 20 seconds with two custom events:

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

6.4 Viewing SEP Results in Intel VTune Amplifier

The .tb6 or .tb7 result files generated by a SEP collection can be imported and viewed in VTune Amplifier. 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 .tb6 or .tb7 file, and 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 VTune Amplifier help.



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:	[illegible]
Operating System:	Red Hat Enterprise Linux Server release 6.2 (Santiago)
Computer Name:	[illegible]
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