# SST Profiling Points

*NOTE: Profiling points are still in development and syntax for enabling profiling tools is subject to change. Additional profiling points may also be added in the future.*

## Profiling point types

Profiling points are points in the code where a profiling tool can be instantiated. The profiling tool allows you to collect various data about code segments. There are currently three profiling points in SST core:

**clock**: profiles calls to user registered clock handlers

**event**: profiles calls to user registered event handlers set on Links

**sync**: profiles calls into the SyncManager (only valid for parallel simulations)

## Enabling profiling points

The format for enabling a profile point is a semicolon separated list where each item specifies details for a given profiling tool using the following format:

name:type(params)[point], where

name: name of tool to be shown in output

type: type of profiling tool in ELI format (lib.type)

params: optional parameters to pass to profiling tool, format is key=value,key=value...

point: profiling point to load the tool into (clock, event or sync)

Profiling tools can all be enabled in a single instance of --enable-profiling, or you can use multiple instances of --enable-profiling to enable more than one profiling tool. It is also possible to attach more than one profiling tool to a given profiling point.

**Examples:**

--enable-profiling="events:sst.profile.handler.event.time.high_resolution(level=component)[event]"

--enable-profiling="clocks:sst.profile.handler.clock.count(level=subcomponent)[clock]"

--enable-profiling=sync:sst.profile.sync.time.steady[sync]

# Full list of profiling tools and parameters

Currently, each profile point has its own set of valid profiling tools as listed below:

## clock:

profile.handler.clock.count

- Counts the number of times the clock handler was called

profile.handler.clock.time.high_resolution

- Times the clock handler using the high resolution clock

profile.handler.clock.time.steady

- Times the clock handler using the steady clock (note that for many systems, the high resoluation and steady clocks are the same)

All versions of the clock profile tools take the same parameter.

***level***: Level at which to track the profile data [default: type]
Available levels:

- global – all data will be tracked globally in a single value
- type – data will be tracked by Component/SubComponent type. All elements of the same type (lib.element) will be tracked in a single value
- component – data will be tracked as one value for all handlers in an instance of a Component and all if its SubComponents
- subcomponent – data will be tracked as one value for all handlers in an instance of a Component or SubComponent (i.e. data will not be aggregated for the entire component, the Component and each of its SubComponents will all be tracked independently)

## event:

profile.handler.event.count

- Counts the number of times event handlers were called

profile.handler.event.time.high_resolution

- Times the event handlers using the high resolution clock

profile.handler.event.time.steady

- Times the event handlers using the steady clock (note that for many systems, the high resoluation and steady clocks are the same)

All versions of the event profile tools take the same parameters.

***level***: Level at which to track the profile data [default: type]
Available levels:

- global – all data will be tracked globally in a single value
- type – data will be tracked by Component/SubComponent type. All elements of the same type (lib.element) will be tracked in a single value
- component – data will be tracked as one value for all handlers in an instance of a Component and all if its SubComponents
- subcomponent – data will be tracked as one value for all handlers in an instance of a Component or SubComponent (i.e. data will not be aggregated for the entire component, the Component and each of its SubComponents will all be tracked independently)

***track_ports***: Controls whether to track by individual ports [default: false]

***profile_sends***: Controls whether sends are profiled [default: false]

- Due to location of profiling point in the code, turning on send profiling will incur relatively high overhead
- Send timing really only times paths in the core and is likely not of much use to the end user

***profile_receives***: Controls whether receives are profiled [default: true]

## sync:

profile.sync.count

- Counts the number of times the SyncManager is called

profile.sync.time.high_resolution

- Times calls to the SyncManager using the high resolution clock

profile.sync.time.steady

- Times calls to the SyncManager using the steady clock (note that for many systems, the high resoluation and steady clocks are the same)

Sync profiling tools have no parameters.