Esper Enterprise Edition - Debugger and Metrics

Version 5.4.0

by *EsperTech Inc.* [http://www.espertech.com]

Preface

This document describes the EPL debugger and the EPL detailed metrics features of Esper Enterprise Edition.

Both features can be used with applications that embed Esper and are not limited to Enterprise Edition server.

The *Chapter 1, EPL Debugger* chapter explains how to use the EPL debugger.

The *Chapter 2, EPL Detailed Metrics Including Memory Use* chapter documents the EPL detailed metrics feature.

Chapter 1. EPL Debugger

1.1. Introduction

The EPL debugger allows you to inspect and debug the impact of events and time passing in respect to EPL statements.

This section describes the EPL debugger features and provides the procedures to help you get started.

The EPL debugger is available for applications that embed Esper, as well as for use with Enterprise Edition server.

1. For using the EPL debugger in conjunction with your own Esper-embedded application, please review *Section 1.3, "Debugger API"*.
2. For a description of the EPL debugger for use with Enterprise Edition server, please see *Section 1.4, "Using the EPL Debugger with Enterprise Edition Server"*.

The EPL debugger user interface is part of the EsperHQ web client application. Its goals are:

1. Present an intuitive interface to allow you to quickly arrive at a diagnosis or solution.
2. Handle large amounts of events and reduce the time spent searching for relevant information.

The EPL debugger is a trace-file based debugging tool. Your application or the Enterprise Edition server records a trace of activity to a file that can be viewed using the EsperHQ web client application.

Please also review the important usage notes listed in *Section 1.5, "Usage Notes"*.

1.2. Understanding the EPL Debugger

EPL is a declarative language for event series analysis and EPL execution is event-driven. Therefore there are some fundamental differences between the EPL debugger and a control flow debugger (for example, a Java or Scala debugger).

Compared to a control-flow debugger, the EPL debugger involves:

1. Analyzing impact of the passage of time on state.
2. Analyzing the effects an event has on analysis constructs such as streams, data windows, patterns, subqueries etc..
3. Analyzing how new events are generated as a result of computations on events or time passing.

The EPL debugger allows event-driven debugging and highlights how an event interacts with analysis constructs, versus a control flow debugger that focuses on the execution aspect only. Thus the EPL debugger can help you understand the impact a given event and the passage of time has on other events and state in general.

1.3. Debugger API

In order to use the public client API of the debugger with your own application that embeds Esper, please follow the instructions herein.

1.3.1. Classpath Setup

The application that embeds Esper must have the following jar files in classpath:

Table 1.1. Debugger required jar files

Jar file	Description
esper-eeutil-*version*.jar	Debugger public API and domain classes
esper-instrumented-*version*.jar	Instrumented Esper jar (place before un-instrumented esper-*version*.jar) in classpath
kryo-*version*-all.jar	Debug object serialization

1.3.2. Using the Debugger API

The debugger API is provided by the class `com.espertech.esper.eeutil.client.debug.DebugClient`.

Please consult the Enterprise Edition Utility API docs in the folder `doc/esper-eeutil/api/index.html` for additional API information.

Use the `isDebugEnabled` static method of `DebugClient` to ascertain that the debug feature is available:

```
boolean canDebug = DebugClient.isDebugEnabled();
```

The `startSession` method of `DebugClient` starts a debug session. The method requires an Esper engine instance and a `File` object as parameters.

This code snippet invokes the debugger API and starts a debug session recording information to the trace file `appl-debug.espertrace`:

```
File file = new File("appl-debug.espertrace");
DebugClientFileSession session = DebugClient.startSession(engine, file);
```

The above example assumes that the `engine` variable is an object of type `EPServiceProvider`.

To end the debug session and close the trace file, please invoke the `endSession` method of the `DebugClientFileSession` returned by the `startSession` method.

A complete example is shown here:

```
File file = new File("appl-debug.espertrace");
DebugClientFileSession session = DebugClient.startSession(engine, file);
try {
  ... // wait or perform some activity
}
finally {
  session.endSession();
}
```

The trace file that is generated by the debug API can then be viewed using the EsperHQ web client. Please follow these steps:

1. Move the trace file to a directory visible to EsperHQ web client, which is by default *installation-root*`/data/hqsvc`. Use the extension `espertrace` so the file is visible in the EsperHQ web client.
2. Load or refresh the EsperHQ web client. The left side file explorer should now show the file. Click on the file and select `Open`.

1.4. Using the EPL Debugger with Enterprise Edition Server

You must start Enterprise Edition server in instrumented operation.

Use `instrumented` (instead of `run`) to start instrumented operation, for example:

```
esperee instrumented (Windows)
// or
startup instrumented (Windows)
// or
esperee.sh instrumented (Linux)
// or
startup.sh instrumented (Linux)
```

Passing the `instrumented` parameter makes sure the classpath uses the Esper instrumented jar file.

After starting the Enterprise Edition server in instrumented operation, you can use the Create Scenario dialog to debug a scenario.

Detailed information about EPL Debugger dialogs as well as Create Scenario dialogs can be found in the EsperHQ web client documentation. Alternatively, please utilize the information (i) dialog button to position to the respective client documentation.

1.5. Usage Notes

Enterprise Edition provides an instrumented build of Esper as part of the product and under commercial license only.

It is save to use the instrumented build for production applications, however performance profiles may differ between the instrumented Esper and the un-instrumented Esper engine.

Multiple simultaneous debug sessions are not supported for the same engine instance.

Chapter 2. EPL Detailed Metrics Including Memory Use

2.1. Introduction

The metrics service feature of Enterprise Edition reports detailed, fine-grained metrics as well as summarized metrics for EPL statements.

The metrics service feature is available for applications that embed Esper as well as for use with Enterprise Edition server.

1. For obtaining detailed EPL metrics using the metrics service API in conjunction with your own Esper-embedded application, please review *Section 2.2, "Metrics Service API"*.
2. For a description of the EPL metrics pages in EsperHQ web client, please see *Section 2.3, "Metrics EsperHQ Web Client"*.

The metrics service can provide detailed information about the following constructs:

- Data windows, including lists of data windows that are in a grouped, intersected or union relationship.
- Patterns and pattern subexpression breakdowns.
- Aggregations including per-group detail.
- Match-recognize states including per-partition detailed information.
- Named window data window detailed information.
- Implicit indexes as well as explicit indexes created by `create index`.
- Subqueries and a explanation of their related data windows, aggregation and indexes.
- Per-context-partition lists.

This list is a limited overview of the data points reported:

- Memory use.
- Counts such as event counts, pattern subexpression counts, match-recognize state counts and key counts generally for any unique, group and partition-able constructs.
- Statement and context-partition information.

- Amount of wall time it took to obtain the measurement.

The metrics services API and web client allow you to choose and customize the detail level of information reported.

Please also review the important usage notes listed in *Section 2.4, "Usage Notes"*.

2.2. Metrics Service API

In order to use the public client API of metrics service with your own application that embeds Esper, please follow the instructions herein.

2.2.1. Configure Java VM Agent

A Java VM agent must be configured in order to report memory use information. If the Java VM agent is not configured then memory use reporting is automatically disabled and the metric information omits any memory byte counts.

The following argument to the Java application to instrument has to be given:

```
-javaagent:/path/to/esper-eeutilagent-<version>.jar
```

The agent jar file can be found in the Enterprise Edition distribution in the `lib` folder.

The agent jar file does not need to be added to the classpath.

2.2.2. Add Enterprise Edition Utility Jar to Classpath

Please add `esper-eeutil-`*`version`*`.jar` to the application classpath.

2.2.3. Metric Service API

The metrics service API is provided by the class `com.espertech.esper.eeutil.client.metric.MetricClient`.

Please consult the Enterprise Edition Utility API docs in the folder `doc/esper-eeutil/api/index.html` for additional API information.

Use the `getMetrics` static method to obtain metrics for a given engine instance and EPL statement names. The method takes an optional `MetricClientOptions` parameter that allows passing additional options.

This code snippet invokes the metric service API and returns metrics for statement `MyStatement`:

```
EngineMetricDetail detail =
```

```
    MetricClient.getMetrics(engine, new String[] {"MyStatement"});
```

The above example assumes that a statement by name `MyStatement` exists and that the `engine` variable is an object of type `EPServiceProvider`.

The options object allows your application to control the metrics level of detail and the flavors of memory use reporting. It is described in detail as part of the API docs.

The next sample code sets the measure-memory-for-events option to false (true by default, see usage notes below) and returns metrics for all statements:

```
MetricClientOptions options = new MetricClientOptions();
options.setMeasureMemoryForEvents(false);
String[] allStatements = engine.getEPAdministrator().getStatementNames();
EngineMetricDetail detail = MetricClient.getMetrics(engine, allStatements,
 options);
```

2.3. Metrics EsperHQ Web Client

The EsperHQ web client offers a dialog to view detailed metric breakdowns. The dialog can be accessed from the `Monitoring` category on the `View Endpoint` page. The View Endpoint page has a `Statement Detailed Metrics` button that leads to the `Browse Statement Detailed Metrics` dialog.

Please utilize the information (i) button for the dialog to position to a help page for this screen, or navigate to the web client user documentation for client help.

2.4. Usage Notes

Caution

- Metrics reporting itself allocates memory in order to produce a metrics report. If a system is already nearly out-of-memory then metrics reporting may itself cause an out-of-memory condition.
- Metrics reporting requires locking of each a context partition for which metrics are reported. The metric service reports the time that each lock is held.

By default, the memory measurement for any context partition includes the memory allocated for any event underlying objects referenced by that context partition. Thus, by default, the memory use of each event underlying object is counted once towards each context partition. It is counted for the construct that references the event first (data window, pattern etc.). The metric options allow changing this default.

The default options for metrics are preset such that only summary information is reported and not very detailed breakdowns. You can change the detail reporting level using the `Show Options` button or the metric options API object.

Note

- The time a lock is held and the memory required for the metrics report is generally increased when the metric options are set to include all details.