About Esper and NEsper
News
About Esper for Java
About NEsper for .NET
License and Trademark Use

Tutorials and Case Studies
Tutorial
Quick Start
Articles and Presentations
Solution Patterns
Short Case Study
Longer Case Study
Additional Examples
Technology Links

Esper for Java
Download
Change History
Documentation
FAQ
Reporting Issues
Building
On Performance

NEsper for .NET
Download
Change History
Documentation
FAQ
Reporting Issues
Building

The Esper/NEsper Team
How To Contribute
Mailing Lists/Forums
Source Repository

Quick Start

This quick start guide provides step-by-step instructions for creating a first event-driven application using Esper.

Installation

Esper is easy to install and run: The first step is to download and unpack the distribution zip or tar file. Provided you have a Java VM installed, you may then run an example as described in the online documentation

Esper consists of a jar file named "esper-version.jar" that can be found in the root directory of the distribution (Download or Maven).

Dependent libraries to Esper are in the "lib" folder.

Esper includes several examples and a benchmark kit that are documented in the reference manual. These can be run from the command line. Esper does not include a GUI or a server middleware itself, other then the benchmark client and server components.

Creating a Java Event Class

Java classes are a good choice for representing events, however Map-based or XML event representations can also be good choices depending on your architectural requirements.

A sample Java class that represents an order event is shown below. A simple plain-old Java class that provides getter-methods for access to event properties works best:

```
package org.myapp.event;

public class OrderEvent {
    private String itemName;
    private double price;

    public OrderEvent(String itemName, double price) {
        this.itemName = itemName;
        this.price = price;
    }

    public String getItemName() {
        return itemName;
    }

    public double getPrice() {
        return price;
    }
}
```

Creating a Statement

A statement is a continuous query registered with an Esper engine instance that provides results to listeners as new data arrives, in real-time, or by demand via the iterator (pull) API.

The next code snippet obtains an engine instance and registers a continuous query. The query returns the average price over all OrderEvent events that arrived in the last 30 seconds:

```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String expression = "select avg(price) from org.myapp.event.OrderEvent.win:time(30 sec)";
EPStatement statement = epService.getEPAdministrator().createEPL(expression);
```

Adding a Listener

Listeners are invoked by the engine in response to one or more events that change a statement's result set. Listeners implement the *UpdateListener* interface and act on *EventBean* instances as the next code snippet outlines:

```
public class MyListener implements UpdateListener {
    public void update(EventBean[] newEvents, EventBean[] oldEvents) {
        EventBean event = newEvents[0];
        System.out.println("avg=" + event.get("avg(price)"));
    }
}
```

By attaching the listener to the statement the engine provides the statement's results to the listener:

```
MyListener listener = new MyListener();
statement.addListener(listener);
```

Sending events

The runtime API accepts events for processing. As a statement's results change, the engine indicates the new results to listeners right when the events are processed by the engine.

Sending events is straightforward as well:

```
OrderEvent event = new OrderEvent("shirt", 74.50);
epService.getEPRuntime().sendEvent(event);
```

Configuration

Esper runs out of the box and no configuration is required. However configuration can help make statements more readable and provides the opportunity to plug-in extensions and to configure relational database access.

One useful configuration item specifies Java package names from which to take event classes.

This snippet of using the configuration API makes the Java package of the OrderEvent class known to an engine instance:

```
Configuration config = new Configuration();
config.addEventTypeAutoName("org.myapp.event");
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider(config);
```

In order to query the OrderEvent events, we can now remove the package name from the statement:

```
String epl = "select avg(price) from OrderEvent.win:time(30 sec)";
EPStatement statement = epService.getEPAdministrator().createEPL(epl);
```