Esper Adapter Reference Documentation

Version: 1.6.0

Table of Contents

Preface

Esper Adapters contains all input and output adapters for the Esper Java event stream processor.

If you are new to Esper, the Esper reference manual should be your first stop. If you are looking for

If you are looking for information on a specific adapter, you are at the right spot.

Chapter 1. Adapter Overview

Input and output adapters to Esper provide the means of accepting events from various sources, and for making available events to destinations.

The following input adapters exist. There are currently no output adapters available.

Table 1.1. Input and Output Adapters

Adapter	Description
CSV Input Adapter	The CSV input adapter can read one or more CSV-formatted input sources, transform the textual values into events, and play the events into the engine. The adapter also makes it possible to run complete simulations of events arriving in time-order from different input streams.

1.1. Adapter Library Classes

1.1.1. Using AdapterInputSource

The `net.esper.adapter.AdapterInputSource` encapsulates information about an input source. Input adapters use the `AdapterInputSource` to determine how to read input. The class provides constructors for use with different input sources:

- `java.io.Reader` to read character streams
- `java.io.InputStream` to read byte streams
- `java.net.URL`
- Classpath resource by name
- `java.io.File`

Adapters resolve Classpath resources in the following order:

1. Current thread classloader via `Thread.currentThread().getContextClassLoader().getResourceAsStream`
2. If the resource is not found: `AdapterInputSource.class.getResourceAsStream`
3. If the resource is not found: `AdapterInputSource.class.getClassLoader().getResourceAsStream`

1.1.2. The InputAdapter Interface

The `InputAdapter` interface allows client applications to control the state of an input adapter. It provides state transition methods that each input adapter implements.

An input adapter is always in one of the following states:

- Opened - The begin state; The adapter is not generating events in this state
- Started - When the adapter is active and generating events

- Paused - When operation of the adapter is suspended
- Destroyed

The state transition table below outlines input adapter states and, for each state, the valid state transitions:

Table 1.2. Adapter State Transitions

Start State	Method	Next State
Opened	start()	Started
Opened	destroy()	Destroyed
Started	stop()	Opened
Started	pause()	Paused
Started	destroy()	Destroyed
Paused	resume()	Started
Paused	stop()	Opened
Paused	destroy()	Destroyed

Chapter 2. The CSV Input Adapter

This chapter discusses the CSV input adapter. CSV is an abbreviation for comma-separated values. CSV files are simple text files in which each line is a comma-separated list of values. CSV-formatted text can be read from many different input sources via `net.esper.adapter.AdapterInputSource`. Please consult the JavaDoc for additional information on `AdapterInputSource` and the CSV adapter.

2.1. Introduction

In summary the CSV input adapter API performs the following functions.

- Read events from an input source providing CSV-formatted text and send the events to an Esper engine instance
    - Read from different types of input sources
    - Use a timestamp column to schedule events being sent into the engine
    - Playback with options such as file looping, events per second and other options
    - Use the Esper engine timer thread to read the CSV file
- Read multiple CSV files using a timestamp column to simulate events coming from different streams

The following formatting rules and restrictions apply to CSV-formatted text:

- Comment lines are prefixed with a single hash or pound # character
- Strings are placed in double quotes, e.g. `"value"`
- Escape rules follow common spreadsheet conventions, i.e. double quotes can be escaped via double quote
- A column header is required unless a property order is defined explicitly
- The value of the timestamp column, if one is given, must be in ascending order

2.2. Playback of CSV-formatted Events

The adapter reads events from a CSV input source and sends events to an engine using the class `net.esper.adapter.csv.CSVInputAdapter`.

The below code snippet reads the CSV-formatted text file "simulation.csv" expecting the file in the classpath. The `AdapterInputSource` class can take other input sources.

```
AdapterInputSource source = new AdapterInputSource("simulation.csv");
(new CSVInputAdapter(epServiceProvider, source, "PriceEvent")).start();
```

To use the CSVInputAdapter without any options, the event type `PriceEvent` and it's property names and value types must be known to the engine. The next section elaborates on adapter options.

- Configure the engine instance for a Map-based event type
- Place a header record in your CSV file that names each column as specified in the event type

The sample application code below shows all the steps to configure, via API, a Map-based event type and play the CSV file without setting any of the available options.

```
Map<String, Class> eventProperties = new HashMap<String, Class>();
eventProperties.put("symbol", String.class);
eventProperties.put("price", double.class);
eventProperties.put("volume", Integer.class);
```

```
Configuration configuration = new Configuration();
configuration.addEventTypeAlias("PriceEvent", eventProperties);

epService = EPServiceProviderManager.getDefaultProvider(configuration);

EPStatement stmt = epService.getEPAdministrator().createEQL(
   "select symbol, price, volume from PriceEvent.win:length(100)");

(new CSVInputAdapter(epService, new AdapterInputSource(filename), "PriceEvent")).start();
```

The contents of a sample CSV file is shown next.

```
symbol,price,volume
IBM,55.5,1000
```

The next code snippet outlines using a `java.io.Reader` as an alternative input source :

```
String myCSV = "symbol, price, volume" + NEW_LINE + "IBM, 10.2, 10000";
StringReader reader = new StringReader(myCSV);
(new CSVInputAdapter(epService, new AdapterInputSource(reader), "PriceEvent")).start();
```

2.3. CSV Playback Options

Use the `CSVInputAdapterSpec` class to set playback options. The following options are available:

- Loop - Reads the CSV input source in a loop; When the end is reached, the input adapter rewinds to the beginning
- Events per second - Controls the number of events per second that the adapter sends to the engine
- Property order - Controls the order of event property values in the CSV input source, for use when the CSV input source does not have a header column
- Property types - Defines a new Map-based event type given a map of event property names and types. No engine configuration for the event type is required as long as the input adapter is created before statements against the event type are created.
- Engine thread - Instructs the adapter to use the engine timer thread to read the CSV input source and send events to the engine
- Timestamp column name - Defines the name of the timestamp column in the CSV input source; The timestamp column must carry long-typed timestamp values relative to the current time; Use zero for the current time

The next code snippet shows the use of `CSVInputAdapterSpec` to set playback options.

```
CSVInputAdapterSpec spec = new CSVInputAdapterSpec(new AdapterInputSource(myURL), "PriceEvent");
spec.setEventsPerSec(1000);
spec.setLooping(true);

InputAdapter inputAdapter = new CSVInputAdapter(epService, spec);
inputAdapter.start(); // method blocks unless engine thread option is set
```

2.4. Simulating Multiple Event Streams

The CSV input adapter can run simulations of events arriving in time-order from different input streams. Use the `AdapterCoordinator` as a specialized input adapter for coordinating multiple CSV input sources by timestamp.

The sample application code listed below simulates price and trade events arriving in timestamp order. Via the

adapter the application reads two CSV-formatted files from a URL that each contain a timestamp column as well as price or trade events. The `AdapterCoordinator` uses the timestamp column to send events to the engine in the exact ordering prescribed by the timestamp values.

```
AdapterInputSource sourceOne = new AdapterInputSource(new URL("FILE://prices.csv"));
CSVInputAdapterSpec inputOne = new CSVInputAdapterSpec(sourceOne, "PriceEvent");
inputOne.setTimestampColumn("timestamp");

AdapterInputSource sourceTwo = new AdapterInputSource(new URL("FILE://trades.csv"));
CSVInputAdapterSpec inputTwo = new CSVInputAdapterSpec(sourceTwo, "TradeEvent");
inputTwo.setTimestampColumn("timestamp");

AdapterCoordinator coordinator = new AdapterCoordinatorImpl(epService, true);
coordinator.coordinate(new CSVInputAdapter(inputOne));
coordinator.coordinate(new CSVInputAdapter(inputTwo));
coordinator.start();
```

The `AdapterCoordinatorImpl` is provided with two parameters: the engine instance, and a boolean value that instructs the adapter to use the engine timer thread if set to true, and the adapter can use the application thread if the flag passed is false.

2.5. Pausing and Resuming Operation

The CSV adapter can employ the engine timer thread of an Esper engine instance to read and send events. This can be controlled via the `setUsingEngineThread` method on `CSVInputAdapterSpec`. We use that feature in the sample code below to pause and resume a running CSV input adapter.

```
CSVInputAdapterSpec spec = new CSVInputAdapterSpec(new AdapterInputSource(myURL), "PriceEvent");
spec.setEventsPerSec(100);
spec.setUsingEngineThread(true);

InputAdapter inputAdapter = new CSVInputAdapter(epService, spec);
inputAdapter.start(); // method starts adapter and returns, non-blocking
Thread.sleep(5000); // sleep 5 seconds
inputAdapter.pause();
Thread.sleep(5000); // sleep 5 seconds
inputAdapter.resume();
Thread.sleep(5000); // sleep 5 seconds
inputAdapter.stop();
```