EsperHQ Client Documentation

Version 5.4.0

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

Preface

This document introduces the EsperHQ web application.

Chapter 1. Overview

This section describes Enterprise Edition web browser technology, implemented using JavaScript and HTML 5. The EsperHQ client application provides:

- The EsperHQ GUI presents a multi-window user interface to design, deploy and manage CEP engine(s) as well as create and launch eventlets and dashboards.
- Continuously-updating displays of type chart, grid, gauge and timeline, configurable and manageable at runtime.
- EsperHQ includes a faceless launch URL that you can pass an eventlet filename or declaration and that launches the eventlet streaming realtime display.
- A jQuery plugin to integrate eventlets into an HTML page and activate eventlets.
- An extension API for building your own eventlets.

We use the term `Eventlet` for real-time displays of events. Eventlets provide a declarative way to hook up sources of data with displays that can be continuously-updating. The client provides the Eventlet Wizard dialog to create and edit eventlets.

When eventlets are launched the client manages the subscriptions as well as the on-demand (pull) queries, if any, for data flows from CEP engine(s) to displays. When eventlets are de-activated (closed or timed out) then the client closes those subscriptions.

Eventlets can be launched within the multi-window captive GUI and may also be launched standalone via faceless launcher or URL, and integrated into HTML pages via jQuery eventlet plugin.

In addition, the EsperHQ client application has a dashboard page builder that presents a drag-and-drop dialog to easily build a web page or application consisting of eventlet and HTML content.

Under *Chapter 3, Cookbook* we provide instructions to address common use cases.

The *Chapter 2, Connectivity* section describes connection to CEP engine(s).

In *Chapter 4, Eventlet Wizard* we outline the eventlet wizard and configuration options.

The section *Chapter 12, Build-Your-Own Eventlet* discusses how to develop a plug-in eventlet.

The *Chapter 6, Engine Management Dialogs* chapter is the client application user manual.

Chapter 2. Connectivity

The EsperHQ client application connects to the same web application server it was loaded from for invoking HQ REST services. The web application server in turn connects to one or more CEP Push Services endpoints. Please see the EsperHQ web application (WAR) documentation for connectivity configuration.

It is not necessary to write JavaScript client code to use the client application and eventlets.

2.1. Endpoints

The EsperHQ client application presents a drop-down dialog box in the upper right corner labeled `Endpoint` that lists all configured endpoints. The list of available endpoints can be configured as part of EsperHQ configuration.

In the next drop-down dialog box titled `Engine URI` the client lists, for the currently chosen endpoint, the available Esper engine URIs. Engine URIs are dynamically discovered, thereby if your application creates and destroys engines any changes are reflected in the client application.

The current selection of endpoint and engine URI applies across the client to all dialogs.

Chapter 3. Cookbook

3.1. Integrating Historical and Real-time Data

All eventlet continuous displays allow multiple resources to be included in the composition. Each resource can use either data push or data pull as well as both together as described in *Chapter 4, Eventlet Wizard*.

In Esper, an EPL statement may query historical or reference data via two means: First, by listing an SQL statement within the EPL statement. Second, by a method invocation join. Please consult the Esper documentation for details. All EPL statements can be non-streaming queries called via the `iterator` pull API.

In addition, Esper supports on-demand SQL-style fire-and-forget queries against named windows. On-demand queries can also be used as data sources along with real-time data in an eventlet composition. A refresh schedule can be attached via iterate instructions.

You may combine historical and real-time data as follows:

1. Create an EPL statement to retrieve the historical or reference data.
2. Create an EPL statement to represent your real-time stream.
3. Create an eventlet with two resources: the historical data EPL statement and the real-time stream EPL statement. For the historical data EPL statement, use the `configure` option to set iterate on.

3.2. Merging Streams of Multiple Endpoints

All eventlet continuous displays allow multiple resources to be included in the composition. Each resource may originate from a different CEP Push Services endpoint. Each resource may explicitly name its endpoint, or may leave the endpoint out to indicate to use the default endpoint configured in Esper HQ.

3.3. Map Entries as Events

If your event rows are within a map property of an event object, use the client un-map instruction under the `configure` option to un-map key-value pairs of a map property.

Continuous displays handle the un-map instruction at runtime. The chart display, for example, creates a series per key value when the pointsource is configured with a label and the series does not have a value for label.

3.4. On-Click and Drill-Down

The eventlet composition XML allows registering a click callback function that is an external function. The click callback function gets invoked on a mouse click of an item in the eventlet

display. The engine passes an Object with contextual information for the item clicked on. The object passed to the script function will contain the event properties or data points for the chart item, timeline, gauge or grid row clicked on.

A sample JavaScript callback that simply prints all object properties is shown below:

```
<script language="JavaScript" type="text/javascript">
  function clicked(obj) {
    var str=""; //variable which will hold property values
    str += "Value for object alone: " + obj + "\n\n";
    str += "Properties are:\n";
    for(prop in obj) {
      str += " " + prop + "="+ obj[prop]+"\n";
    }
    alert(str); //Show all properties and their value
  }
</script>
```

In response to a callback your JavaScript function may, for example, construct an eventlet composition XML or construct the name of a stored eventlet from the values passed.

3.5. Display of Long-type Timestamps

Eventlets receive property metadata as part of their subscription(s) and when new event types are available. When eventlets recognize a `long`-type property the property value if it represents a valid date is automatically converted to the default date-time format for output, based on seconds since 1970. Therefore it is sufficient to simply select the `long`-type timestamp value in an EPL query.

To perform custom formatting or timestamp conversion, consider defining a plug-in single-row function (see CEP engine documentation).

Chapter 4. Eventlet Wizard

The eventlet wizard is a 3-page dialog to create and edit an eventlet.

The first page of the wizard, termed `Visualization Type`, presents a list of visualization types to choose from. Any visualization type can be combined with any data source. To proceed, please click on a table row and press the `Next` button.

The second page termed `CEP Resources` presents the possible resources to obtain data from. Possible resources include EPL statements, dynamic EPL statement sets, on-demand queries, variables and engine metrics. To proceed, in the section `Select Resources` please click on `Get All` to get a list of statements, for example, click on a table row and press the `Next` button.

The third page termed `Configuration` allows configuring the visualization and resources. Some display types do not require any configuration. To proceed, please populate the required form fields and press the `Save` or `Launch` button.

The button bar on the upper left side shows your progress through the wizard. On the right of the button bar is a progress text. Click the `Next` button to continue to the next page and the `Back` button to go back a page.

The `Save` button allows to save the eventlet as a file for later launching or editing. The `Launch` button launches the eventlet.

4.1. Eventlet Visualization Types (Step 1)

The below table summarizes the eventlet display types available.

Table 4.1. Eventlet Display Types

Name	Description
Chart	For display of line, column, area or pie charts.
Grid	For display of a single table or table of tables.
Gauge	An instrument with an arrow pointing to the current value.
Plug-In	An eventlet implementation provided by your application, see *Chapter 12, Build-Your-Own Eventlet*.
Timeline	Horizontal time-based display.

4.2. CEP Resources (Step 2)

The next table summarizes the resources available for providing data to eventlets. As CEP engine resources are the data sources for eventlets, we use that term interchangeably.

Resources can be configured in terms of their push policy: Whether the push includes real-time updates, initial snapshots and/or repeated snapshots following a pattern or schedule.

4.2.1. CEP Resources: Single EPL Statement

You can subscribe to a single EPL statement's output events, where the EPL statement is identified by statement name.

You can optionally provide a filter expression. The push engine applies the filter expression to each output event of the statement and only passes events on that satisfy the filter expression, when provided. The filter expression must return a boolean true or false.

The statement must be registered and in started state at time of eventlet activation.

By default, the eventlet receives all current data available (aka. a snapshot, iterator) for the statement at activation time. Subsequently it receives any updated data from the statement in real-time based on push services SLA levels.

Use the configure link displayed for the statement to change the default configuration. You may disable initial snapshot and real-time updates as well as configure pull frequency in addition or as an alternative.

4.2.2. CEP Resources: Set of EPL Statements

You can also subscribe to a dynamically changing set of zero, one or many statements. You may define selection criteria for statements to include in the set based on statement name, expression, state or statement tags using equals or SQL-like semantics and conjunctions.

At the time of activation it is not required that any statement matching the selection criteria exists. Only at the time one or more statements are started that match selection criteria will the eventlet received metadata and data from the statements. When a statement is destroyed, the eventlet stops receiving data from that statement.

You can also form a statement set from all statements. In this case the eventlet receives data from any currently started statement as well as any newly started statement after eventlet activation.

When selecting a set of statements then any updates are always real-time updates and pull frequency cannot be configured.

4.2.3. CEP Resources: Engine Metrics

You can elect to subscribe to certain pre-defined engine metrics, as displayed in the respective tab.

4.2.4. CEP Resources: On-Demand Query

You can subscribe to a query against a named window that is executed upon eventlet activation and can be executed repeatedly thereafter.

By default the query executes once at the time of eventlet activation. You may also configure an interval or pattern to follow for repeated query execution.

4.2.5. CEP Resources: Variables

You can subscribe to changes to a variable, by variable name.

The variable must exist at time of activation of the eventlet.

By default, the eventlet receives all current data available for the variable at activation time. Subsequently it receives any updated data from the variable in real-time based on push services SLA levels.

Use the configure link displayed for the variable to change the default configuration. You may disable initial snapshot and real-time updates as well as configure pull frequency in addition or as an alternative.

4.2.6. CEP Resource Configurations

The `Configure` button available on the resources dialog for each selected resource can provide additional configuration for each resource.

4.2.6.1. Push Policy

For all CEP resources with the exception of statement sets, you may configure push policy.

In the default push policy, the push services push a snapshot of current data once at the start of the subscription. Therefore `Push Snapshot Once On Start` is checked. If the eventlet does not need to receive initial snapshot data, please uncheck this flag.

In the default push policy, the push services push new data when new data is available. Therefore `Real-Time Updates` is checked. If the eventlet does not need to receive real-time data, please uncheck this flag.

In the default push policy, the push services does not push periodic snapshots. Therefore `Push Snapshots Based on Expression` is unchecked. If the eventlet desires to receive periodic snapshots, please check this flag and enter a time-only pattern.

The next example pattern pushes a snapshot after 10 seconds every 10 seconds:

```
every timer:interval(10 seconds)
```

The next example pattern pushes a snapshot every 15 minutes between 8am and 5pm and also immediately (timer-interval zero):

```
every timer:at(*/15, 8:17, *, *, *) or timer:interval(0)
```

4.2.6.2. Rows in a Map

When your single event contains a Map that itself has key-value pairs for use as individual events, you can transform a property of type Map to individual events.

You must provide the property name of the map-type property, the key property name and type, and the value property name and type in the dialog.

4.3. Configuration (Step 3)

In the `Configuration` step of the dialog you can enter eventlet configuration information. For certain eventlet types you must enter required information that is marked with `(*)`.

4.3.1. Chart Configuration Dialog

In the chart dialog you must populate the `Origin` and `X` and `Y` required fields of each series. The origin identifies which of the selected resources provides the data points. The `X` field is the name of the property or the built-in field that provided values for the horizontal X-axis of the chart, and the `Y` field for values of the vertical `Y` axis.

Checking `Stack` produces a stacked line, area or column chart (not supported for pie). In a stacked chart each of the series represents part of the stack and should have the same origin for `X`.

You may add as many series to a chart as needed.

Populate the click callback function name to register an external function (JavaScript, VBScript) to invoke when a mouse click on a chart item occurs.

The `Opacity` field is optional and defines the opacity of the area or columns (only for area and column charts). Please provide a number between 0 and 1. For column charts, the opacity should be set on the chart-level and is the same for all series.

For category-based charts such as bar charts or pie charts set the horizontal axis type to `Category`. Set the vertical axis type to `Linear` to display linear values. In the `X` field set the property that provides the category value. In the `Y` field set the property that provides the linear value.

Colors are specified in the CSS hex format `#000000`.

4.3.2. Gauge Configuration Dialog

The gauge configuration dialog lets you set many design attributes of the gauge. You must populate the `Origin` and `X` required fields.

Populate the click callback function name to register an external function (JavaScript, VBScript) to invoke when a mouse click on the gauge occurs.

4.3.3. Timeline Configuration Dialog

The timeline configuration dialog lets you set a click callback function name, which is an external function (JavaScript, VBScript) to invoke when a mouse click on an item occurs. It also allows to set a level of display detail.

In the `Tiny` and `Small` detail levels, a mouse click on on event brings up a modal popup showing of event properties for the event. In the `Full` detail level, since event properties are displayed inline, please use double-clikc to bring up the modal popup.

Optionally, you may also select the field that provides the timestamp values that the eventlet uses to place events in the timeline view. You must select a field that returns either numeric number of milliseconds or date-type values. The values returned by the field must be sorted in ascending date order.

4.3.4. Grid Configuration Dialog

Grids provide in-place-updated, as well as appended-to grids and last-value row grids that can be presented as part of a nesting of tables.

Use the *Nesting* option to enable nested grids. A nested grid requires a key property field by which the eventlet can nest tables.

Use the *In-place Updates* option to enable in-place updated grids. You must specify one or more key property fields that the eventlet uses to identify same-row. Rather than appending rows, the eventlet updates the row that returns the same key values. The label and background color of the last updated row(s) change according to the style property values.

4.3.5. Plug-In Configuration Dialog

This dialog is for an eventlet that your application or another provider may supply in the form of JavaScript. The EsperHQ eventlet container can load such externally-provided functions (or classes) at runtime. Please see *Chapter 12, Build-Your-Own Eventlet* for additional information.

The dialog requires you to enter the classname.

4.4. Eventlet Settings

The `Settings...` button in menu of the wizard dialog bring up the eventlet general settings dialog.

This dialog allows you to specify the number of data points retained by the eventlet. By default, eventlets retain all data points for display. You can change the number of data points retained by entering a count and selecting a unit from the dropdown.

The dialog also allows setting menu to false, causing the eventlet menu to not be displayed.

Chapter 5. Dashboard Page Builder

The dashboard page builder is a means to compose a complete web page that combines eventlets with static content or other dynamic content.

The output of the dashboard page builder is a HTML page that fully adheres to the HTML 5 standard. You may edit the output file in a text or HTML editor and introduce additional content or formatting. The output HTML file contains a short script that initializes eventlets when a web browser loads the page.

The dashboard page builder is a drag & drop dialog. On the left side of the dialog is a list of components that may be dragged onto the green-colored drop area.

Drag and drop the eventlet icon to place eventlet content into the web page. Drag and drop the Text/HTML icon to place text or HTML content onto the web page. Drag and drop any of the cell or row icons to organize the web page into rows and cells.

The dashboard page is organized into a grid system that consists of rows wherein each row contains one or more cells. You may add and remove rows and organize the cells in each row.

The size of all cells in a row always adds up to 12. The width of a cell that is size 12 is 100% of available size and therefore takes up the space for a whole row. The width of a cell that is size 6 is 50% of available size. Therefore if a row has 3 cells that are size 6, 4 and 2 then the cells take roughly 50%, 32% and 16% of space. Remaining space is allocated to padding between cells.

Use the `Save` button to save the HTML file to the file system. Use the `Save and Open in New Window` button to save the HTML file to the file system and open the HTML file in a new window. The saved HTML file can also be opened from the file explorer.

5.1. Dashboard Eventlet Configuration

After dropping an eventlet icon onto a cell, the builder displays eventlet configuration input fields. The input fields consist of a drop-down from which you can choose the origin of the eventlet, and of an input field.

Select `Filename` from the drop down if you want to identify an eventlet file that exists in the file system. You can then choose an eventlet file that you have previously saved in the file system by pressing the `Select File` button.

Select `External URL` from the drop down if your eventlet file resides on another server. When loading an eventlet from an HTTP or other resource, specify the URL to load from in the format `protocol:filename`. Examples for URL are `http://myhost/myeventlet.xml`.

Select `XML Document` from the drop down if you want to enter eventlet XML directly. You can then enter or cut and paste the eventlet XML document into the input field.

5.2. Dashboard Text/HTML content

The Text/HTML component can be used to insert HTML or XHTML-formatted content. The dashboard builder outputs an HTML document and the text you enter becomes part of the body, including its HTML formatting.

For example, by entering `<h3>My Title</h3>` into the text area the output page formats the title as a header.

5.3. Customizing Generated HTML

You may add style information to the generated HTML.

By default, style `row-fluid` has zero height thereby all content expands row height. Add `style="height: 400px; overflow: auto;"` to the `div` containing the div to assign a row height. In addition, please configure eventlet display height or width as follows.

For setting the height and width of a chart eventlet, please refer to *Section 10.5.1, “Chart Eventlet Parameters”*.

For setting the height and width of a timeline eventlet, please refer to *Section 10.5.2, “Timeline Eventlet Parameters”*.

Chapter 6. Engine Management Dialogs

This section outlines client dialogs for engine management.

Use the `Endpoint` and `Engine URI` drop-down dialog boxes to select the endpoint and engine.

6.1. Browse Statements Dialog

This dialog shows EPL statements.

You may click on a table entry to view statement detail.

Use the `Get All` button to retrieve a list of all statements regardless of the selection expression.

Click the `New` button to create a new statement.

Use `Find` to retrieve only a subset of statements according to current statement selection expression. Use the `Selection Criteria` dialog to compose a selection expression that selects only those statements matching the selected criteria. The selection criteria can be an expression such as `name like '%mystatement%'` to select all statements that have a statement name that contains the text `mystatement`. If the selection criteria are not a valid expression, they are taken as free-form text to match against statement name or statement EPL.

Click the `Refresh` button to retrieve the statement list again to refresh the screen.

The `Search` entry allows to reduce the list of returned statement to those matching the entered text in statement name, EPL, state or any other displayed field.

6.2. Statement Selection Criteria Dialog

This modal dialog allows putting together a selection expression that the engine applies to each statement to determine whether to show the statement. This allows you to filter statements by statement name, state, expression, tag or user object using equals, SQL-like or SQL-regex matching.

An example expression is `name like '%mystatement%'`. This expression selects all statements that have a statement name that contains the text `mystatement`.

After closing the dialog, use the `Find` button to find the selected statements.

6.3. Statement Detail Dialog

The statement detail dialog shows statement detail including statement name, description, EPL expression and properties of the statement output. This is a read-only form.

Via Start (displayed if a statement is not already started), Stop (displayed if a statement is not already stopped) and Destroy buttons a statement can be started, stopped or destroyed.

Use Iterate to view current iteration results for a statement.

Use Instant Live Capture to display an appended grid with real-time results for the statement.

Use Save Live Capture to save an eventlet configured to show an appended grid with real-time results for the statement.

The Eventlet Wizard button brings up the eventlet wizard, with the statement already added to the selected resources.

6.4. Statement Iterate Dialog

This dialog can be accessed from the statement detail dialog. It allows to view iteration results for the selected statement.

By default the dialog retrieves the first 100 rows and the table displays rows in batches of 10. Remove the value for maximum number of rows to retrieve all rows.

The Submit button refreshes iteration results.

6.5. Create Statement Dialog

This quick-entry screen for creating a new, started statement allows entering a statement name, an optional description and an EPL expression.

The checkbox for pattern can be used to create an EPL pattern statement (a statement utilizing only the EPL pattern language).

6.6. Browse Deployments Dialog

This screen lists all deployments, including deployments in undeployed state. Select a deployment by clicking on the list item to display its detailed information. Use New to create a new deployment. Use Refesh to re-read the list of deployments.

A deployment is a set of EPL statements that are deployed and undeployed as a unit. A deployment may mention a module name or may also omit the module name. The deployment id is the unique identifier of each deployment. Deployments that are in DEPLOYED (EPL statements started) or UNDEPLOYED state (no EPL statements started).

6.7. Deployment Detail Dialog

This screen displays the deployment detail in a read-only form. Among the deployment information is the deployment id which is a system-generated id or the deployment id provided on another form. When the deployment EPL text uses the module keyword, the form display the module name.

The deployment state of DEPLOYED indicates that the module may have started EPL statements. For deployed modules the screen shows a list of statements associated to the deployment.

The deployment state of UNDEPLOYED means the deployment is known (added to the list of deployments) but does not have EPL statements that are started.

For deployments in DEPLOYED state, the Undeploy and Remove button undeploys and permanently removes a deployment. If the deployment is in deployed state, this operation destroys all statements associated to the deployment.

For deployments in DEPLOYED state, use Undeploy to undeploy a deployment. This operation destroys all statements associated to the deployment but does not remove the deployment, therefore the deployment can be deployed again later.

For deployments in UNDEPLOYED state, you can use Remove to permanently remove a deployment.

For deployments in UNDEPLOYED state, you can use Deploy to deploy a deployment and therefore start all statements associated to the deployment.

The Save button allows saving the EPL text to a file.

6.8. Create Deployment Dialog

This dialog allows entering, validating and deploying zero, one or more EPL statements as a unit. The editing rules follow EPL module text editing rules: EPL Statements are separated by a semicolon (;) character. The dialog also offers to look up existing deployments, populate EPL from a template, save to a file and load from a file.

The Templates button brings up a dialog that lists templates. Please select a template row to populate the EPL text area from the predefined template text.

The Existing Deployments button can be used to select an existing deployment for editing.

Use Load to load EPL text from a file. Use Save to save EPL text to a file. Files reside in a server-side directory.

Use Validate to validate the EPL statements. This action validates syntax as well as all dependencies of all EPL statements in the EPL text. Any statement errors are displayed in the messages list.

Use Add to add the deployment but no start any EPL statements. This operation assigns a deployment id but does not validate the EPL text. The deployment remains in undeployed state.

Use Deploy to deploy the deployment. When successful the deployment is in deployed state and a confirmation dialog displays the deployment id. All statements for the deployment are started. When validation errors occur then the messages are displayed in the messages list and none of the statements therein are in started state.

Upon Deploy, when the Undeploy And Remove Existing checkbox is checked (the default), any existing deployments that have either the same module name or the same deployment id are

undeployed and removed before deploying the EPL statements, without rollback if the deployment fails.

Upon `Deploy`, when the `Close On Deploy` checkbox is checked (the default), the dialog closes.

6.9. Browse Event Types Dialog

This dialog shows all registered event types and associated type detail. By selecting a type from the list you may bring up the event type detail dialog for the selected type.

The `Search` textbox and button is for quick finding of an event type.

Use the `Refresh` button to populate the list anew.

6.10. Event Type Detail Dialog

The event type detail dialog shows detail information about a selected event type. The form is read-only.

For the selected event type the form shows where the event type originates from (named window, application-configured or inserted-into stream), the properties available, the statements referring to that type and the event type's super-types, if any.

You can click on a statement referring to that type to navigate to the statement detail screen.

6.11. Named Windows / Tables Dialog

The named window and tables dialog lists all named windows as well as tables including their associated statement name and EPL for creation of the named window or table.

The `Search` textbox is for quick finding of a named window or table. Use `Refresh` to retrieve and display an updated list of named windows and tables.

For each named window, the `Event Type` button opens up the event type information for the named window.

For each named window or table, the `Statement` button opens up the statement detail of the create-window statement and the `Query` button opens the on-demand query dialog.

6.12. Browse Map of Streams Dialog

This dialog presents a graphical overview of the statements including event types and their dependency as depicted by arrows.

The `Map All` button loads and displays all statements. To select a subset of statements use `Selection Criteria` or enter a selection criteria expression and press `Read`. The `Refresh` button re-loads statements according to selection criteria.

The scale slider controls the scale at which the display operates. The `Show Abstract` checkbox controls whether the displayed information includes statement summary information, the `Show Name` checkbox controls whether statement name is part of the display, and the `Show Expression` checkbox controls whether the EPL expression is displayed as well.

Each statement and event type in the display has a pen icon that turns blue when hovering over the pen. By clicking on the pen you can call up the statement detail or event type detail dialog.

The display shows the selected event types and statements. Stream consumption relationships are indicated by an arrow. Update and insert relationships are also indicated by arrows.

6.13. Browse Dataflow Dialog

The dataflows dialog lists all declared dataflows including the dataflow name and the declaration text.

The `Search` textbox is for quick finding of a dataflow. Use `Refresh` to retrieve and display an updated list of dataflows.

Click on a dataflow row in the list to see the dataflow declaration detail dialog.

6.14. Dataflow Detail Dialog

The dataflow detail dialog displays the selected dataflow declaration including the dataflow name and the declaration text.

This dialog allows instantiating dataflows: Please enter an instance name and click `Instantiate` to instantiate the dataflow. The instance name must be unique among the currently known dataflow instances.

You can optionally enter additional operator parameters: Specify a URI of *operatorName/propertyName* for each parameter as well as the parameter value and value type. For example, if the operator is `BeaconSource` specify the `iterations` property using the URI `BeaconSource/iterations`.

The `Destroy` button removes the dataflow declaration by destroying the statement declaring the dataflow.

The browse dataflow instances dialog lists the known dataflow instances.

6.15. Browse Dataflow Instances Dialog

The dataflows instances dialog lists all dataflow instances that are running, cancelled or completed.

The `Search` textbox is for quick finding of a dataflow instance. Use `Refresh` to retrieve and display an updated list of dataflow instances.

Click on a dataflow instance row in the list to see the dataflow instance detail dialog.

6.16. Dataflow Instance Detail Dialog

The dataflow detail dialog displays the selected dataflow instance including the dataflow name and the dataflow instance state, as well as parameters provided at time of dataflow instantiation.

When the dataflow instance is in `RUNNING` state the dataflow instance is currently executing. Click on `Cancel` to cancel the execution which causes the dataflow instance to transition to `CANCELLED`. Click on `Cancel And Remove` to cancel execution and remove the dataflow instance. This causes the data flow instance to no longer appear in the list.

When the dataflow instance is in `CANCELLED` or `COMPLETED` state the dataflow instance has been cancelled or completed and is no longer executing. Click on `Remove` to remove the dataflow instance. This causes the data flow instance to no longer appear in the list.

6.17. Create Sequence Dialog

A *Sequence* is a list of instructions that can be executed. Instructions consists of events to send into the engine and of time passing. Use a Sequence to stimulate or simulate events and time passing.

A sample sequence is shown below:

```
StockTick={symbol='IBM', price='50'}
t = t.plus(5 seconds)
StockTick={symbol='GE', price='100'}
```

The sample sequence above starts by sending an event of the type `StockType` (define event types separately) that has a `symbol` value of `IBM` and a price value of `50`. The sequence then pauses for 5 seconds wall clock time. It then sends a `StockType` event that has a `symbol` value of `GE` and a price value of `100`.

The syntax for sequences is described in greater detail below.

You must enter a unique name for the sequence as well as a sequence to execute. Check the `Loop` checkbox to have the execution loop, which causes the sequence to start from the beginning when the sequence arrived at the end.

Click the `Load` button to bring up a file selection dialog for loading a sequence. Click the `Save` button to bring up a file selection dialog for saving a sequence.

Use the `Validate` button to validate the sequence. The system performs type-assignment checking and lists validation messages below the editor.

The `Add` button adds the sequence but does not execute it. The `Start` button starts execution of the sequence.

The `Add Interval` button brings up a form to enter an interval.

You can select an event type in the drop-down and click the `Add Event` button to bring up a form that you can use to populate event property expressions. The `Refresh Types` button re-reads the list of event types.

Sequences can be in `STARTED` and `STOPPED` state. Started sequences are currently executing. Stopped sequences are not currently executing.

6.17.1. Sequence Syntax

Each instruction of a sequence can either send an event or wait a specified amount of wall clock time.

To send an event into the engine, enter an assignment of event type name to an array of name-value pairs, as follows:

```
mytype = {intProperty = 0, stringProperty = 'abc'}
```

Within the `{}` array you can list individual event property names and assignments to a value. Right-hand side assignment expressions can be any constant.

To assign the value of an expression evaluation to an event property, use the `eval` function and specify the expression string.

The next example assigns the result of the expression `5*5` to `intProperty` and the result of the `myFunction` function (assumed to be registered with the engine instance) to `stringProperty`.

```
mytype = {intProperty = eval("5*5"), stringProperty = eval("myFunction()")}
```

To advance wall clock time, enter an assignment of time `t` to t plus a time period:

```
t = t.plus(10 minutes 5 seconds)
```

Please see the "plus" date-time enumeration method and the time period parameter documentation in Esper docs for more information.

Alternatively, you may also advance time in millisecond steps by adding milliseconds to t:

```
t = t + 1000
```

The above example adds 1 second (1000 milliseconds) to the current engine time.

You may also advance engine time to a specific point in time by specifying the date-time in the format `yyyy-MM-dd HH:mm:ss.SSS`.

The following assignment advances engine time to May 1st 2002 at 8am:

```
t = '2002-05-1 8:00:00.000'
```

6.18. Browse Sequences Dialog

This dialog lists sequences. More information on sequences can be found at *Section 6.17, "Create Sequence Dialog"*.

Click the `New` button to create a new Sequence. The `Refresh` button refreshes the list with the most current information.

Click on a list item to bring up the sequence detail dialog for the selected sequence.

6.19. Sequence Detail Dialog

This read-only dialog displays detail information about a Sequence. The past activity list displays the last 100 actions taken by the sequence, with the most recent action on the top of the list.

For a started, currently-executing sequence you can use `Refresh` to refresh the list of past activity log entries.

The `Stop` button is enabled when the sequence is in started state and allows stopping the sequence.

The `Start` button is enabled when the sequence is in stopped state and allows starting the sequence.

The `Edit And Remove` button stops the sequence if it is started and brings up an editor for the sequence.

The `Remove` button stops and removes the sequence.

6.20. Create Scenario Dialog

This dialog allows testing and debugging a scenario consisting of EPL statements and a sequence of events.

The `Debug` button allows bringing up the EPL debugger for the scenario. The button is greyed out when the server does not run in instrumented operation. Start the server with the instrumented option as described in the Esper Enterprise Edition - Debugger and Metrics documentation (see link from index.html). You must provide the file name of a trace file into the input text box next to the button. The debugger uses the value you entered to create a trace file of the same name and prompts to open the trace file.

The editor dialogs on this screen can be resized if the space is insufficient. You may drag the borders of each section to resize the contents.

You can enter EPL statements on the left side of the page. You can enter multiple EPL statements separated by the semicolon (;) character, in the same format as the Create Deployment dialog. The `Load EPL` and `Save EPL` buttons allow loading and saving EPL content.

Use the middle section of the page to enter a sequence of instructions that sends events into the engine or advances time, or both. Please see *Section 6.17.1, "Sequence Syntax"* for information on sequence syntax. This dialog also allows time assignment instructions in the format `t = '`*time*`'`, for example `t = '2001-01-01 08:00:00.000'`. The `Load Sequence` and `Save Sequence` buttons allow loading and saving sequences.

The `Run` button triggers validation and execution. Only the entered EPL and sequence are validated. The entered EPL must contain all event types, variables or named windows for the scenario. Any other event types, variables or named windows are not visible to the scenario.

The right side of the page displays results. The first tab `All Output Events` displays a breakdown of all events produced by all EPL statements. The `Output Per Statement` tab shows output per statement. The third tab `All Audit Text` displays the time and events that were sent in, the output of @Audit annotations (if any were declared) and a line per output event that shows the statement name of the statement producing the output. The last tab `Audit Text Per Statement` displays the same but per statement.

Timestamps follow the format `yyyy-MM-dd HH:mm:ss.SSS`. For example, the value `2013-07-01 08:00:00.000` is a valid timestamp.

The date-time value entered in the `Beginning Of Time` input is the start time. Before the application creates EPL statements, it sets the CEP engine time to the provided time.

After pressing `Run`, the application lists output events and audit information on the right side. Prefix your statements with `@Name('my name')` to assign a statement name for easier identification of statement output. Prefix your statements with `@Audit` to obtain audit (debug) output for that statement.

This is a stateless service in that each time you execute a scenario the service allocates and destroys an engine instance. The configuration of the engine instance matches the instance selected for the dialog, thereby event types, variables and plug-in functions, for example, are all available if they have been preconfigured for the selected engine.

6.21. On-Demand Query Dialog

This screen allows entering a fire-and-forget, one-time executed query against a named window into the query dialog box and executing the query. The client displays query results in a table.

You can enter a value for the maximum number of rows to return in the query into the `Max Rows` input.

The `Submit` button evaluates the query anew and displays updated query results.

6.22. Variables Dialog

The variable list displays all variables and their current values. Select a variable, enter a new value and use `Apply` to set a new variable value.

The filter textbox and button is for quick finding of a variable.

Chapter 7. EPL Debugger Dialogs

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

The EPL debugger dialog operates on a trace file previously acquired either by means of the debug API or in the Create Scenario dialog.

For documentation on EPL debugger prerequisites, the debugger client API, comparison and overview please consult the Esper Enterprise Edition - Debugger and Metrics documentation (see link from index.html).

We use the term *entry point* to mean a specific event or a specific time. An entry point may in addition relate to a specific EPL statement context partition.

The EPL Debugger consists of two dialogs:

- The EPL Debugger main dialog for filtering and finding entry points.
- The EPL Debugger Entry Point dialog for detailed viewing of the processing for a given entry point.

7.1. EPL Debugger Main Dialog

In summary, the left hand side read-only editor presents all EPL statements. The right hand side read-only editor presents the sequence of events and time passing interleaved with event output. The bottom section presents activity filtered according to current selections and organized in various ways.

7.1.1. EPL Statements

All EPL statements that exist at the time the debug session started, as well as all EPL statements that are created, are displayed within the left hand side editor.

Click on the gutter between the line number and each EPL statement to select an EPL statement. The editor will display a filter and indicate statement selection. Click again to remove the statement selection.

If, during a debug session, your application creates EPL statements assigning the same statement name multiple times, the EPL statement also lists the system-generated statement id that uniquely identifies each statement.

7.1.2. Time And Event Sequence

The right hand side editor displays events sent into the engine as well as rows indicating time passing. The editor interleaves output events as comments.

For each event sent into the engine, the editor shows the event in a summary format:

```
eventtype={properties}
```

For time passing, the editor shows time passing in the summary format:

```
t='time'
```

Click on the gutter between the line number and each event or time to select an input event or time. The editor will display a filter and indicate event or time selection. Click again to remove the selection.

7.1.3. Activity

The bottom part of the debugger presents activity, filtered according to the current filter criteria, if any.

Each tab organizes the same overall set of activity in different ways:

- The `Sequential` tab shows a time-ordered list of activity as a nested list of entry points.
- The `By Statement And Context Partition` tab organizes activity by statement and context partition(s), if any.
- The `By Input Event` tab organizes activity by input event only, showing only input events.
- The `By Time Advancing` tab organizes activity by time passing only, showing only time passing.

You may select an item in any of the tabs to navigate to the EPL Debugger Entry Point dialog.

7.1.3.1. Sequential Tab - Additional Information

In the sequential tab the outermost items that are highlighted by a blue arrow icon represent the input events and time passing.

Nested inside and highlighted by a gear icon are to-be-processed events including generated (insert-into or route) events and including named window insert and remove stream events, to be processed as a result of an input event or time passing.

Innermost and highlighted by a green arrow icon are statement(s) that require processing.

7.2. EPL Debugger Entry Point Dialog

Selection of an entry point in the EPL debugger brings up detailed information about how the event applies to the different EPL constructs.

This dialog presents event and time information at the top of the screen, for reference.

The left side of the screen shows the specific EPL statement currently under consideration, as applicable.

The right side of the dialog presents a navigable trace of activity for the event or time passing. Click on the + icon to expand a nested trace entry and click on the nested trace entry itself to obtain the trace variable information.

The bottom left side of the screen presents frame information. When navigating to a nested trace, this information gets updated with enclosing trace information.

The bottom right side of the screen presents variable detail information in an expandable tree. When navigating to a nested trace, this information gets updated with information relevant to the selected entry.

Chapter 8. Push Monitoring Dialogs

This section outlines client dialogs for push monitoring.

8.1. Browser Dialog

This dialog shows the client id associated to the browser window or tab. The client id is unique to a browser window. The browser window is also called `client`. The web layer associates the client id to activations.

In addition, the dialog shows the web socket URL, the web socket state and a total message count.

Under `Activations` the dialog shows activations currently active for the client.

Under `Messages` the dialog shows messages for the client with the newest message listed first and the oldest message last.

The `Refresh` button fetches updated information.

8.2. Web Layer Dialog

The Web Layer Dialog is a view into operation of the web application server for the web client. Use the `Refresh` button to update the screen with the most current information.

In the Clients tab is an entry for each unique web-browser client. The client id serves as the unique identifier assigned to a client. The Client History tab lists closed or disconnected clients.

The Client Activations tab displays a list of activations for each client. The term activations stands for active eventlets. Under CA History (Client Activations History) are past activations.

The Client Resources tab displays per client and activation the endpoint(s) and resource(s) that make up the activation.

The Server Resources tab displays subscription information per server resource. We use the term server resource to mean CEP engine(s) statements, variables etc..

The SR History (Server Resources History) tab displays past server resources.

The Audit tab displays messages about state transitions for clients, activations and server resources.

The Log Messages tab displays log messages such as exceptions or other important information.

The Stats Total tab displays message count totals overall.

The Stats Per Client tab displays message count totals per client.

8.3. View Endpoint Dialog

This dialog view presents a view into a given endpoint's push service operation. A web client may connect and subscribe to information from multiple endpoints. Use the `Refresh` button to retrieve updated information.

This dialog is also the launch point for CEP engine monitoring. You may select an engine URI using the drop down dialog box, and select a display for CEP engine metrics, then use the `Open` button to display the engine metrics.

The `Statement Detailed Metrics` button opens the EPL detailed metric and memory reporting dialog.

The Sessions tab displays a list of sessions. A session is a logical connection between a push services endpoint and a server in the web layer. A session may have zero to many subscriptions to server resources managed by push services. When a session ends, its associated subscriptions also end. The session id is the unique identifier of a session. The Session History tab shows past sessions.

The Resources tab displays a list of server resources managed by push services. A server resource can be a statement, statement set, variable, on-demand query, engine metric or other publishable data. Push services assigns a resource id to each managed server resource. The Resources History tab displays a list of past server resources.

The Subscriptions tab displays a line for each session indicating an interest in a server resource. Push services assigns a subscription id to each such association. The Subscriptions History tab displays past subscriptions.

The Audit tab displays a history of all state transitions of sessions, resources and subscriptions.

The Log Messages tab displays relevant push services messages.

The Stats tab displays message statistics per destination. For each destination the dialog displays the since-start totals as well as last N combined-message totals.

8.4. Statement Detailed Metrics Dialog

Use this 2-step dialog to inspect statement detail metrics including memory use.

The first step in the dialog allows selecting statements for reporting. After you have selected one or more statements use `Report Metrics` to obtain resource reports. Make sure to check the checkbox `This action requires locking of each context partition`. When the engine obtains memory use, it must do so while having the context partition write locked disallowing any concurrent activity.

Use the `Show Options` button to configure metric reporting options. By default, the dialog reports high-level metrics for each statement and does not report detailed breakdown per grouping or criteria. Please use the checkboxes to control the level of detail of metrics reporting.

In step 1, the statement selection, the `Read All` reads all statements and the `Selection Criteria` button brings up a statement selection expression dialog. Use the `Find` button to read statements according to the selection expression and the `Refresh` button to refresh. The `Search` text box is a quick search among the displayed statements.

After pressing `Report Metrics`, the dialog displays metrics and summary information per statement.

Additional information can be found in the Esper Enterprise Edition - Debugger and Metrics documentation.

Chapter 9. Eventlet Menu Dialogs

This section outlines the menu options and dialogs available when launching eventlets from the menu dropdown.

The menu option `Clear` clears all data held by the eventlet.

The menu option `Suspend-Discarding` causes the eventlet to discard incoming data. When you click `Resume-Stop Discarding` the eventlet starts applying incoming data again.

The menu option `Suspend-Buffering` instructs the eventlet to buffer incoming data and not update the page. When you click `Resume-From Buffered Data` the eventlet applies the buffered data to the page.

9.1. Eventlet About Dialog

The About dialog for eventlets is a read-only dialog and shows the client id associated to the browser window or tab and additional information such as activation name, eventlet activation xml, message count, live messages and last update time. Live messages include eventlet initialization messages, configuration messages and any eventlet exceptions or other information an eventlet may report.

This screen gets updated automatically and there is no refresh option (the refresh is automatic).

9.2. Eventlet Settings Dialog

The Settings dialog for eventlets allows controlling the number of events retained and shown by an eventlet. This setting can also be changed for the eventlet in the eventlet wizard under `Settings` or directly in the eventlet XML definition.

The default is generally that an eventlet does not discard any events and thereby retains all events. You can uncheck the Retain All Events checkbox to change the amount of events retained. Please enter a numeric value into Retain Last and select a unit from the dropdown. This setting does not affect server operation and any memory associated with retaining data points is client web browser memory.

Use the `Apply` button to apply the new settings without closing the dialog, and the `Ok` button to apply and close the dialog.

9.3. Timeline Eventlet - Settings Dialog

The Settings dialog for Timeline eventlets can be used to change the level of detailed displayed for each event and to change the color associated to each event type.

The dialog presents a list of event type names and a color picker for each.

Use the `Apply` button to apply the new settings without closing the dialog, and the `Ok` button to apply and close the dialog.

Chapter 10. jQuery Plugin for Eventlets

The jQuery plugin for eventlets lets you create eventlets using jQuery.

10.1. Introduction and Usage

The jQuery plugin operates on a placeholder DOM element. Based on the initialization object provided, it enhances the DOM structure under the DOM placeholder element to load and display the eventlet.

The plugin operates in conjunction with Enterprise Edition push services to manage the flow of data to eventlet(s). Multiple eventlets on the same page share a websocket connection.

10.2. Examples

The web page can declare a `div` element with an id:

```
<div id="example"></div>
```

The jQuery plugin enhances the `div` element and activates the eventlet:

```
$(document).ready(function() {
  var startEventlets = function() {
    $("#example").eventlet({eventlet_filename: "SampleEventlet.eventlet"});
  };
  $.eventletLoad({success:startEventlets});
} );
```

The `eventletLoad` function must be called to set up the eventlet client. Pass a success callback function that activates eventlets by calling the `eventlet` function.

The example above passes the filename `SampleEventlet.eventlet` to the jQuery eventlet plugin, thereby the plugin loads the eventlet from the file system.

An example file that incorporates multiple eventlets into a single page can be found in `example/Example Dashboard JQuery Eventlet for Onlineshop.html` in the `data/hqsvc` folder.

You can also use the GUI Dashboard Page Builder to build an HTML page that incorporates eventlets and that uses the jQuery eventlet plugin.

10.3. API

10.3.1. Plugin API

The following functions are available from the `jQuery` eventlet plugin:

Table 10.1. jQuery Eventlet Plugin API

Function	Description
`eventletLoad(`*load object*`)`	Prepare for loading and activating eventlets. Must be called before `eventlet()` calls. Pass a callback function to wait for successful load.
`eventlet(`*initialization object*`)`	Activate an eventlet.
`eventletUnload()`	Deactivate all eventlets for this web page.

The `eventletLoad` function must be called at least once before activating any eventlets, to allocate client resources. In the `success` property of the object passed to `eventletLoad`, provide a callback function that the eventlet runtime invokes when completed:

The typical sequence of function calls is:

```
var startEventlets = function() {
  $("#example").eventlet({eventlet_filename: "SampleEventlet.eventlet"});
};
$.eventletLoad({success:startEventlets});
```

The `eventletUnload` function should be called before a web page unloads, to release the server and client-side resources:

```
$.eventletUnload();
```

The following example assigns a function to `window.onbeforeunload` that unloads all eventlets:

```
var unloadPageEventlets = function() {
 $.eventletUnload();
};
window.onbeforeunload=unloadPageEventlets;
```

Note

We recommend adding code to your web page to unload all eventlets or destroy each eventlet, at the time the web page unloads. This allows for a clean disconnect of the eventlet container from the web layer push service.

10.3.2. Initialization Object

You must specify either the `eventlet_filename`, or the `eventlet_xml` or the `eventlet_url` property of the initialization object. The next table outlines the initialization object properties.

Table 10.2. jQuery Eventlet Plugin Initialization Parameters

URL Parameter Name	Description
`eventlet_filename` : *filename*	Assign `eventlet_filename` to the file name of an eventlet file in the file system.
`eventlet_xml` : *xml document*	Assign `eventlet_xml` to the XML document that defines the eventlet.
`eventlet_url` : *URL*	Assign `eventlet_url` to the URL of the web resource that originates the eventlet.

Pass an eventlet XML document to provide the eventlet definition without reading from a file:

```
$("#example").eventlet({eventlet_xml:
  '<eventlet xmlns="http://www.espertech.com/esper-eventletxml">\
    <composition>\
      <timeline detail="full"/>\
    </composition>\
    <sources>\
      <source sourceId="0">\
        <statement name="MyStatement"
            engineUri="default" endpoint="defaultendpoint"/>\
      </source>\
    </sources>\
  </eventlet>'});
```

Or pass a URL that points to a web resource that returns the eventlet definition:

```
$("#example").eventlet(
  {eventlet_url: "http://myserver:port/path/to/file/MySampleEventlet.xml"}
);
```

10.3.3. Eventlet Control API

Although most of the time your Javascript interaction with the jQuery eventlet plugin will be done using the initialization object as described earlier, you might find it useful to have some external control of the eventlet.

The following functions are available from the `jQuery.eventlet` object:

Table 10.3. jQuery Eventlet Instance API

Function	Description
`destroy`	Destroys the eventlet.

Pass the literal `destroy` to the `eventlet` function to destroy a specific eventlet:

```
$("#example").eventlet('destroy'));
```

10.4. Required Script and CSS Files

Please include the following script files:

```
<script src="scripts/release/vendor.min.js"></script>
<script src="scripts/release/common.min.js"></script>
<script src="scripts/release/eventlet.min.js"></script>
```

Please include the following CSS files:

```
<link rel="stylesheet" href="styles/release/vendor.css">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/eventlet.css">
```

10.5. Additional Eventlet Parameters

10.5.1. Chart Eventlet Parameters

The chart eventlet can be parameterized by height and width.

Table 10.4. Chart Eventlet Parameters

Function	Description
`width`	The width in pixels as an int value or the CSS width as a string-typed value.

Function	Description
`height`	The height in pixels as an int value or the CSS height as a string-typed value.

For example:

```
$("#eventlet").eventlet(  {eventlet_filename:  "mychart.eventlet",  height:400,
 width:400});
```

10.5.2. Timeline Eventlet Parameters

The timeline eventlet can be parameterized by height.

Table 10.5. Timeline Eventlet Parameters

Function	Description
`height`	The height in pixels as an int value.

For example:

```
$("#eventlet").eventlet(       {eventlet_filename:       "mytimeline.eventlet",
 height:400});
```

Chapter 11. Eventlet Launcher

The eventlet faceless launcher allows your application to launch eventlets via URL. By passing URL parameters the launcher web page obtains all necessary information to provide the execution environment for the eventlet. You may start saved eventlets by name or by providing a URL or activation XML document.

The eventlet faceless launch is useful for web applications that require launching an eventlet in an HTML iframe or frame, for example. Use the jQuery eventlet plugin instead for those web applications that need finer-grained control via scripting.

The URI for the launcher is:

```
http://host:port/esperhqapp/launcher.html?launch_parameters
```

You must specify either the `eventlet_filename`, or the `eventlet_xml` or the `eventlet_url` parameter. The next table outlines the launch parameters.

Table 11.1. Eventlet Faceless URL Launch Parameters

URL Parameter Name	Description
`eventlet_filename`=*filename*	Specify `eventlet_filename` followed by the file name of an eventlet file in the file system to activate a saved eventlet.
`eventlet_xml`=*xml document*	Specify `eventlet_xml` followed by the XML document (escaped) that defines the eventlet.
`eventlet_url`=*URL*	Specify `eventlet_url` followed by the encoded URL of the web resource that defines the eventlet.

The following are examples of faceless launch URLs.

The next example launches an eventlet by providing the eventlet file name as it has been saved to the file system. It assumes that you have an eventlet by name `MySavedEventlet.eventlet` stored in the `myeventlets` subfolder of the root folder. The runtime loads the eventlet from the file system:

```
http://host:port/esperhqapp/launcher.html?eventlet_filename=myeventlets/
MySavedEventlet.eventlet
```

This example launch URL provides the eventlet XML as part of the URL parameters. Fill in the eventlet activation XML where 'MYXMLHERE' appears in the URL below, making sure the XML document is escaped.

```
http://host:port/esperhqapp/launcher.html?eventlet_xml=MYXMLHERE
```

This example launches an eventlet by providing a URL from which the eventlet itself can be loaded. This example loads the file `MySavedEventlet.eventlet` from the server `myserver` at port 8000 and at a given path:

```
http://host:port/esperhqapp/launcher.html?eventlet_url=http://myserver:8000/
path/to/file/MySavedEventlet.eventlet
```

Chapter 12. Build-Your-Own Eventlet

It is easy to develop a JavaScript eventlet and have that eventlet be managed by the eventlet runtime, jQuery plugin, EsperHQ GUI and faceless launcher. Thereby your application can provide only the minimal code needed for display logic of continuously-updating data from event streams. This chapter develops a simple eventlet that receives and displays event data.

The description herein requires working knowledge of JavaScript. An eventlet is a JavaScript prototype or object that provides a set of functions for the eventlet runtime to call.

It is not necessary to develop event classes in JavaScript: The eventlet runtime and push service map event object properties of CEP engine events of any underlying type to the corresponding structure in JSON. Nested, indexed and mapped properties are also supported.

If you would rather customize an existing eventlet other than the examples we provide as described below, please contact us.

The easiest approach to developing an eventlet can be to copy the eventlet example provided with the distribution. The example code can be found in the `examples-eventlet-javascript` folder.

12.1. Functions

The table below overviews the functions that must be provided by an eventlet object:

Table 12.1. Eventlet Object Functions

Function	Description
`eventletConfigure(`*configuration*`)`	Called once by the eventlet runtime after eventlet object construction and before loading HTML content, passing configuration information such as the composition and sources.
`eventletInitialize(`*context*`)`	Called once by the eventlet runtime after loading eventlet HTML content and before the eventlet receives any event data or manifests, passing the root DOM element and the eventlet container event target.
`eventletUpdate(`*update*`)`	Called by the eventlet runtime passing event data.
`eventletManifest(`*manifest*`)`	Called by the eventlet runtime passing manifests. Manifests contain source and type metadata.
`eventletExpire(`*expire*`)`	Called by the eventlet runtime passing expiry information.

The below code provides a complete JavaScript prototype that implements all required eventlet functions, for illustration purposes.

```
function EventletExampleMinimal() {
}

EventletExampleMinimal.prototype.eventletConfigure = function(configuration) {
    console.log("eventletConfigure called");
    console.log("  sources: " + JSON.stringify(configuration.sources));
    console.log("  composition: " + JSON.stringify(configuration.composition));
    console.log("  activation name: " + configuration.activationName);
    console.log("  activation xml: " + configuration.activationXML);
};

EventletExampleMinimal.prototype.eventletInitialize = function(context) {
    console.log("eventletInitialize called");
    // context.containerEventTarget to interact with the menu and output text
    // context.rootDOM for addressing DOM elements in HTML content
};

EventletExampleMinimal.prototype.eventletUpdate = function(update) {
    console.log("eventletUpdate called");
    console.log("  update: " + JSON.stringify(update));
};

EventletExampleMinimal.prototype.eventletManifest = function(manifest) {
    console.log("eventletManifest called");
    console.log("  manifest: " + JSON.stringify(manifest));
};

EventletExampleMinimal.prototype.eventletExpire = function(expire) {
    console.log("eventletExpire called");
    console.log("  expire: " + JSON.stringify(expire));
};
```

12.1.1. Function `eventletConfigure`

The eventlet runtime invokes the `eventletConfigure` method as the first call to the object after object instantiation. Here the code should validate the configuration, if any, and throw an error if the configuration is invalid.

The eventlet runtime passes a configuration object that contains useful properties that your eventlet may inspect to obtain information about the composition of the eventlet, the event type(s) and CEP server resource information.

Properties of the configuration object passed to the function are:

Table 12.2. Configuration Object Properties

Name	Description
`composition`	Contains the eventlet composition according to the eventlet XML document.
`sources`	Information about the sources contributing data.
`activationXML`	Contains the XML definition of the eventlet.
`activationName`	The activation name is the file name if activated from a file, or URL if activated from a URL.

12.1.1.1. Property `composition`

The `composition` property of the configuration object is a direct mapping of the eventlet XML document composition element to JSON and JavaScript objects and contains all of the eventlet definition.

An example structure for `composition` is shown below:

```
{
  "classname":"EventletExampleMinimal"
}
```

12.1.1.2. Property `sources`

The `sources` property of the configuration object is an array of sources and contains:

- The `sourceId` as per eventlet XML document identifying the server resource by number.
- The `resourceId` is generated by the server and is provided for every event.
- The `manifest` contains a `typeId` which is generated by the server and is unique per event type. The `properties` field is a list of event properties.
- The `target` is a description of the CEP server resource.

An example structure for `sources` is shown below:

```
[
   {
      "sourceId":"0",
      "resourceId":1,
      "manifest":{
         "typeId":1,
         "definition":{
            "properties":[
```

```
               {
                  "propertyName":"somePropertyValue",
                  "propertyType":"int",
                  "writable":true,
                  "requiresIndex":false,
                  "requiresMapkey":false,
                  "indexed":false,
                  "mapped":false
               }
            ]
         },
         "target":{
            "statement":{
               "statement":{
                  "engineURI":"default",
                  "statementName":"MyStatement",
                     "epl":"@Name('MyStatement') select somePropertyValue from
 MyEventType",
                   "eplNoAnnotation":"select somePropertyValue from MyEventType",
                  "state":"STARTED",
                  "lastStateChange":"2013-07-27T11:43:33.632+02:00",
                  "pattern":false,
                  "type":"SELECT",
                  "properties":[
                     {
                        "propertyName":"somePropertyValue",
                        "propertyType":"int",
                        "writable":true,
                        "requiresIndex":false,
                        "requiresMapkey":false,
                        "indexed":false,
                        "mapped":false
                     }
                  ]
               }
            }
         }
      }
   }
]
```

12.1.2. Function `eventletInitialize`

The eventlet runtime invokes the `eventletInitialize` method always after `eventletConfigure` and after loading HTML resources. The eventlet runtime passes a context object.

Properties of the object passed to the function are:

Table 12.3. Initialization Object Properties

Name	Description
`containerEventTarget`	For listening to menu commands such as the `Clear` button, and for sending trace and error messages.
`rootDOM`	The root DOM element of the loaded HTML, as a jQuery object.

Register a callback function that is invoked when a user selects `Clear` from the eventlet menu:

```
context.containerEventTarget.addListener("clear", function() {
        console.log("Clear called");
});
```

Send trace messages that become visible in the menu `About` dialog:

```
context.containerEventTarget.fireTrace("A trace message");
```

Send error messages that become visible in the menu `About` dialog:

```
context.containerEventTarget.fireError("An error message");
```

12.1.3. Function `eventletUpdate`

The `eventletUpdate` method is always called after `eventletInitialize` and indicates arriving events.

Properties of the update object passed to the function are:

Table 12.4. Update Object Properties

Name	Description
`entries`	Array of event objects.

Each event object contains:

- The `resourceIds` property is an array of integer values. The resource id is the same as the `resourceId` received in the `sources` property passed to `eventletConfigure`.
- The `activationIds` property is an array of integer values that lists activation ids and is mostly for internal use.
- The `typeId` property contains the event type id and is the same as the `typeId` received in the `sources` property passed to `eventletConfigure`.
- The `arrivalTime` property is the long-type server time.

- The `newRows` property is an array of events arriving. Each array is itself an object that contains all event property values as well as the builtin `__rowid` property. The builtin `__rowid` property is a long-type client-assigned event number.

A sample JSON representation of the object passed to the `eventletUpdate` method is:

```
{"entries": [
   {
      "resourceIds":[
         1
      ],
      "activationIds":[
         "A_1"
      ],
      "typeId":1,
      "arrivalTime":1374918222732,
      "newRows":[
         {
            "somePropertyValue":100,
            "__rowid":0
         }
      ]
   }
]}
```

12.1.4. Function `eventletManifest`

The eventlet runtime invokes the `manifest` method when new event type information is available and always after `eventletInitialize`. The eventlet code can, for example, adjust the display when new types of events become known.

Properties of the manifest object passed to the function are:

Table 12.5. Manifest Object Properties

Name	Description
`manifests`	Array of manifest entry objects.

Each manifest entry object contains:

- The `resourceIds` property is an array of integer values. The resource id is the same as the `resourceId` received in the `sources` property passed to `eventletConfigure`.
- The `activationIds` property is an array of integer values that lists activation ids and is mostly for internal use.
- The `manifest` property contains a `typeId` property that contains the event type id and is the same as the `typeId` received in the `sources` property passed to `eventletConfigure`. The `definition` contains a list of property descriptors.

- The `target` is a description of the CEP server resource.

A sample JSON representation of the object passed to the `manifest` method is:

```
{"manifests": [
   {
      "resourceIds":[
         1
      ],
      "activationIds":[
         "A_1"
      ],
      "manifest":{
         "typeId":3,
         "definition":{
            "properties":[
               {
                  "propertyName":"someProperty",
                  "propertyType":"java.lang.String",
                  "writable":false,
                  "requiresIndex":false,
                  "requiresMapkey":false,
                  "indexed":false,
                  "mapped":false
               }
            ]
         },
         "target":{
            "statement":{
               "statement":{
                  "engineURI":"default",
                  "statementName":"MyStatement",
                  "epl":"@Name('MyStatement') select * from MySampleEvent",
                  "eplNoAnnotation":"select * from MySampleEvent",
                  "state":"STARTED",
                  "lastStateChange":"2013-07-27T13:21:48.656+02:00",
                  "pattern":false,
                  "type":"CREATE_WINDOW",
                  "properties":[
                     {
                        "propertyName":"someProperty",
                        "propertyType":"java.lang.String",
                        "writable":false,
                        "requiresIndex":false,
                        "requiresMapkey":false,
                        "indexed":false,
                        "mapped":false
                     }
                  ]
```

```
                }
             }
          },
          "endpoint":"defaultendpoint"
       }
    }
]}
```

12.1.5. Function `eventletExpire`

The eventlet runtime invokes the `eventletExpire` method only when retain-all is unchecked and a retain period or number of events is set through the menu or through launch parameters.

The object passed to `eventletExpire` has the following properties:

Table 12.6. Expire Object Properties

Name	Description
`rowIdOffset`	Contains the row id (`__rowid`) of the newest event that expired, i.e. that no longer is retained according to retain settings.

12.2. Running

This section outlines the steps required to run your own JavaScript eventlet.

The below sequence of steps saves an eventlet that refers to the JavaScript eventlet object:

1. In the EsperHQ client application, start the Eventlet Wizard dialog.
2. Select `Plugin` among the list of visualization types and click `Next`.
3. Select CEP resources such as a statement that provides the data to display, click `Next`.
4. As the classname, enter the prototype object name, for example `EventletExampleMinimal`. Save the eventlet to the file system using the `Save` button and assign the name `example_minimal.eventlet`.

The below sections provide information how to use the jQuery eventlet plugin or the faceless launcher and EsperHQ GUI to launch the eventlet.

12.2.1. Run Using jQuery Eventlet Plugin

A sample HTML file that uses the jQuery plugin to launch the eventlet is provided below and also in the `examples-eventlet-javascript` folder by name `EventletExampleMinimal_example.html`.

The sample HTML file assumes that you saved the eventlet under the name `example_minimal.eventlet`. If you chose a different name, please edit the HTML before pointing your browser to the page.

Copy the `EventletExampleMinimal_example.html` and `EventletExampleMinimal.js` to the root folder of the web server at *installation root*`/webapps/esperhqapp`.

Launch the example by navigating the browser to `http://localhost:8400/esperhqapp/EventletExampleMinimal_example.html`.

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML running EventletExampleMinimal</title>
    <link rel="stylesheet" href="styles/release/vendor.css">
    <link rel="stylesheet" href="styles/common.css">
    <link rel="stylesheet" href="styles/eventlet.css">
    <script src="scripts/release/vendor.min.js"></script>
    <script src="scripts/release/common.min.js"></script>
    <script src="scripts/release/eventlet.min.js"></script>
    <script src="EventletExampleMinimal.js"></script>
    <script type="text/javascript">
      $(document).ready(function() {
       $("#example").eventlet({eventlet_filename: "example_minimal.eventlet"});
      } );
      window.onbeforeunload = function() { $.eventletUnload(); };
    </script>
</head>
<body>
<div id="example"></div>
</body>
</html>
```

12.2.2. Run Using GUI and Launcher

To run the eventlet within the EsperHQ GUI, except for Open-in-new-Window, you must edit the file `index.html` in the folder *installation root*`/webapps/esperhqapp` and add your JavaScript code to the scripts.

To run the eventlet in the launcher or in the EsperHQ GUI using Open-in-new-Window, you must edit the file `launcher.html` in the folder *installation root*`/webapps/esperhqapp` and add your JavaScript code to the scripts.

For the example above, add `<script src="EventletExampleMinimal.js"</script>` to each file and make sure the JavaScript exists in the top folder.

12.3. Providing HTML Content

Your eventlet application can provide HTML content by specifying the `url` parameter of the eventlet configuration. The eventlet runtime loads the HTML content from the provided URL and places the content in the container. The root DOM element becomes the parent DOM element of the content.

The HTML content should be a partital HTML content, not a full HTML page with body. Typically the HTML content contains a template or form to populate or a placeholder that gets replaced with event data.

If you specify a URL to load HTML content for the eventlet, you must specify the attribute `ng-controller="EventletController"` on the outermost HTML element.

If you do not specify a URL parameter for the plugin eventlet, the eventlet runtime loads the following content:

```
<div ng-controller="EventletController">
    <div hqeventletmenu="yes"></div>
</div>
```

You may specify `<div hqeventletmenu="yes"/>` to place the eventlet menu anywhere in your HTML content.

12.4. Example Eventlet

The sample eventlet described next can be found in the `examples-eventlet-javascript` folder of the distribution under name `EventletExampleTiledDisplay`. The sample consists of the following files:

- The file `EventletExampleTiledDisplay.js` contains the JavaScript prototype that provides the eventlet functions.
- The file `EventletExampleTiledDisplay.html` contains the content HTML that the eventlet runtime loads when the eventlet activates.
- The file `EventletExampleTiledDisplay.css` contains CSS stylesheets.
- The file `EventletExampleTiledDisplay_example.html` contains HTML for running the eventlet and uses the jQuery eventlet plugin.

The example eventlet displays events in a grid or 3 rows and 3 cells per row. Each cell in the grid will show one event. Among the information in the cell is the source information i.e. the statement name of the event or the engine metric name or stream name providing the input event, as well as a list of event properties and their values.

The grid will begin populating with events from the upper right corner. Once all grid cells are populated the eventlet returns to the first cell and renders the newest arriving events from the first cell onwards again.

12.4.1. Example Eventlet - Launch HTML

The sample `EventletExampleTiledDisplay_example.html` loads the eventlet using the jQuery eventlet plugin. It provides the eventlet XML document as a property of the initialization object that it passes to the jQuery eventlet plugin.

```
$(document).ready(function() {
  $("#example").eventlet({eventlet_xml:
    '<eventlet xmlns="http://www.espertech.com/esper-eventletxml">\
      <composition>\
        <plugin url="EventletExampleTiledDisplay.html"
            classname="EventletExampleTiledDisplay">\
         <configuration>some configuration can be passed here</configuration>\
        </plugin>\
      </composition>\
      <sources>\
        <source sourceId="0">\
        <statement name="MyStatement"
            engineUri="default" endpoint="defaultendpoint"/>\
        </source>\
      </sources>\
    </eventlet>'});
} );
```

Use the eventlet wizard to generate an eventlet file that holds the xml-formatted document. Note how the plugin configuration specifies a `url` property. The URL property is the HTML file that contains the HTML content loaded for the eventlet.

12.4.2. Example Eventlet - Content HTML

The eventlet runtime loads the file `EventletExampleTiledDisplay.html` because the `url` parameter in the eventlet XML document provides the url to load. The content of the file is shown below and provides the markup of the grid to display.

```
<div ng-controller="EventletController">
    <div hqeventletmenu="yes"></div>
    <div class="clear"></div>
    <div class="row-fluid tiled-display-row">
        <div class="span4" id="cell0"></div>
        <div class="span4" id="cell1"></div>
        <div class="span4" id="cell2"></div>
    </div>
```

```
    <div class="spacer"></div>
    <div class="row-fluid tiled-display-row">
        <div class="span4" id="cell3"></div>
        <div class="span4" id="cell4"></div>
        <div class="span4" id="cell5"></div>
    </div>
    <div class="spacer"></div>
    <div class="row-fluid tiled-display-row">
        <div class="span4" id="cell6"></div>
        <div class="span4" id="cell7"></div>
        <div class="span4" id="cell8"></div>
    </div>
</div>
```

The `row-fluid` and `span4` CSS styles are Bootstrap styles.

12.4.3. Example Eventlet - JavaScript

The file `EventletExampleTiledDisplay.js` provides the logic to handle event display.

The code displays events in a grid of 3 rows and 3 cells per row and retains the last 9 events in memory. When events expire because retain-settings indicate event expiry, the code empties the cell accordingly.

The code retains type information that it receives in manifest objects. The eventlet displays this type information, which may consist of any of statement, variable, on-demand query or engine metric information, as part of the header text of each cell.

The code builds cell contents from the type information and from event property values using jQuery.

12.4.4. Running the Example Eventlet

Copy all four example files, the JavaScript, CSS, content HTML and launch HTML files, to the web app root directory at *installation root*`/webapps/esperhqapp`.

Create an EPL statement by name `MyStatement` or change the launch HTML file to refer to a different EPL statement name.

Launch the example by navigating the browser to `http://localhost:8400/esperhqapp/EventletExampleTiledDisplay_example.html`.

12.5. Troubleshooting

Common configuration problems will result in the eventlet runtime displaying an alert JavaScript popup that displays a message.

Errors raised by eventlet code are displayed on the menu `About` dialog and also the console. Therefore the first step in troubleshooting is to bring up the browser console and inspect messages.

There are also a number of good tools for JavaScript debugging that can be used in conjunction with the jQuery eventlet plugin, for example.