# EsperJMX

## Version 5.4.0

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

# Preface

This document describes the EsperJMX Add-on. The document assumes that the reader has prior knowledge of Esper and basic knowledge of Java JMX terminology.

If you are new to Esper, please study some of the tutorials and case studies available on the public web site at `http://www.espertech.com/esper` and skim over the reference documentation.

If you are new to JMX, the tutorials and examples at `http://java.sun.com/jmx` are a good entry point.

The *Chapter 1, Overview* chapter is the best place to start.

# Chapter 1. Overview

EsperJMX is an add-on for use with Esper to provide administrative and runtime functions via JMX MBeans following JMX standards and compatible to any JMX console.

EsperJMX can be configured as part of your regular Esper configuration file and thus be initialized as part of Esper service provider initialization. Or EsperJMX can be used on an already-active Esper service provider to allow JMX-based management.

This is a short summary of the main features:

1. Create new EPL or pattern statements; Start, stop and destroy statements. Obtain statement lists and statement details.
2. Subscribe to statement results delivered as JMX notifications.
3. Iterate over statement results returned as JMX tabular data.
4. Runtime metrics, such as number of events evaluated, for display and graphing.
5. Dynamically execute non-continuous, fire-and-forget queries against named windows (aka. on-demand queries).
6. Named window metrics such as number of events held.
7. Variable get and set.

In the next section, *Chapter 2, Configuration*, we explain the different ways to set up the EsperJMX.

The JMX management beans (aka. *MBeans*) are summarized in *Chapter 3, MBeans*.

*Chapter 4, Examples* shows a Java process with a JMX-enabled Esper service provider and a JMX client that performs an on-demand fire-and-forget query against a named window. The section contains code samples for performing the query and a screenshot of EsperJMX in a JMX console.

## 1.1. Installation

To enable EsperJMX, add the `esperjmx-`*version*`.jar` jar file to the Esper classpath. Your application must employ one of the methods outlined in *Chapter 2, Configuration* to start EsperJMX.

EsperJMX also utilizes the Log4J log library for logging. Thus log configuration for EsperJMX can be added to the same log configuration file used for Esper. All EsperJMX packages begin as `com.espertech.esper.jmx` thereby a log level for EsperJMX may be configured by adding a Log4J logger for EsperJMX as follows:

```
<logger name="com.espertech.esper.jmx">
  <level value="INFO"/>
</logger>
```

# Chapter 2. Configuration

There are two options to start EsperJMX:

1. As an Esper plug-in:

   This requires your application to add EsperJMX to the Esper configuration (XML or API). EsperJMX is then started automatically as part of Esper engine initialization.

2. As a JMX endpoint via the EsperJMX endpoint API:

   Allows JMX management to be added to an already running Esper engine instance and provides more options for JMX connectivity.

The above options are mutually exclusive. Use the EsperJMX endpoint API if you have your own JMX MBeanServer (such as when running in an application server) or require secure JMX connections.

## 2.1. Esper Plug-in Configuration

By adding EsperJMX as a plug-in adapter to an Esper configuration, Esper initialization also initializes and starts the EsperJMX.

Under this option EsperJMX is initialized when your application first obtains an `EPServiceProvider` instance for a given URI or when your application calls the `initialize` method on an `EPServiceProvider`. EsperJMX is destroyed when your application calls the `destroy` method on an `EPServiceProvider` instance or when it calls the `initialize` method on `EPServiceProvider` instance that had the adapter in its configuration.

This option configures the MBeanServer and JMX connector via properties passed in XML or `Properties` object. If passing no configuration properties or an empty list of properties, EsperJMX uses the platform MBeanServer provided by your Java VM.

### 2.1.1. Via Esper Configuration API

This section shows how to register EsperJMX as part of Esper configuration using the Esper configuration API.

Your application must populate a `Properties` object to set EsperJMX configuration options. The available configuration property names are found in `JMXEndpointConfiguration` and are documented below.

The sample code here sets the properties to use an RMI registry at port 1099 and provide a JMX connector for RMI:

```
Configuration configuration = new Configuration();      // Esper Configuration
 class
```

```
// Properties are used to pass EsperJMX configuration
Properties properties = new Properties();
properties.put("use-platform-mbean-server", false);
properties.put("rmi-registry-port", 1099);

configuration.addPluginLoader(
  "EsperJMX",
  "com.espertech.esper.jmx.client.EsperJMXPlugin",
  properties);
```

The JMX service URL is thus:

`service:jmx:rmi:///jndi/rmi://`*hostname:1099*`/com.espertech.esper`.

### 2.1.2. Via Esper Configuration XML

The XML as below configures an engine instance with EsperJMX. It specifies the same configuration options as outlined above.

```
<esper-configuration>
  <plugin-loader name="EsperJMX"
        class-name="com.espertech.esper.jmx.client.EsperJMXPlugin">
    <init-arg name="use-platform-mbean-server" value="false" />
    <init-arg name="rmi-registry-port" value="1099" />
   </plugin-loader>
</esper-configuration>
```

### 2.1.3. Plug-In Configuration Properties

The next table outlines the properties that can be passed to the EsperJMX plug-in.

Further configuration options are available via the EsperJMX endpoint API (not using the Esper plug-in). The endpoint API can also accept a preconfigured application-provided JMX MBeanServer.

#### Table 2.1. JMX Plug-in Configuration Properties

| Option | Default | Description |
|---|---|---|
| use-platform-mbean-server | true | If false, the plug-in creates an RMI registry and connector via `LocateRegistry.createRegistry` and `JMXConnectorServerFactory.newJMXConnectorServer`.<br><br>If true, the plug-in uses the JMX MBeanServer provided by the Java VM (the platform MBeanServer) and its connector |

| Option | Default | Description |
|---|---|---|
| | | (if configured for the JVM), obtained via `ManagementFactory.getPlatformMBeanServer()`.<br><br>The Sun Java VM platform MBeanServer can be enabled, with disabled security, via these JVM properties: `-Dcom.sun.management.jmxremote.port=1099 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false`.<br><br>If using the Sun Java VM platform MBeanServer, the service URL for use by clients is `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`.<br><br>Note that when using the platform MBeanServer the domain name setting below cannot be changed as the value is `jmxrmi`. |
| rmi-registry-port | 1099 | If `use-platform-mbean-server` is false, the setting indicates the port for the RMI registry.<br><br>Make sure the port is not already in use. The port can also not be in use by the platform mbean server if specified via `-Dcom.sun.management.jmxremote.port=port`. |
| service-url | (see description) | The service URL for use by `JMXConnectorServerFactory.newJMXConnectorServer`.<br><br>Sample:<br><br>`service:jmx:rmi:///jndi/rmi://localhost:1099/com.espertech.esper`<br><br>The port number must match the RMI registry port number. |
| domain-name | com.espertech.esper | When specified overrides the default domain name for the MBeanServer and all MBean object names.<br><br>The domain name does not need to match the default domain name that is part of the `service-url`. |
| create-named-window-mbean | true | If true EsperJMX creates a MBean for each named window. |

| Option | Default | Description |
|---|---|---|
| create-stmt-mbean | true | If true EsperJMX creates a MBean for each statement. |
| create-stmt-listener-mbean | false | Default is false. If true EsperJMX creates a MBean for each statement listener. |
| num-notification-threads | 1 | The number of threads in the threadpool for use in broadcasting notifications. |

## 2.2. EsperJMX Endpoint Configuration

Instead of the Esper plug-in configuration as outlined above, your application can use the EsperJMX endpoint API classes. The advantages are:

- Start JMX management on a running Esper engine instance.
- Use an application-provided MBeanServer such as when running within an application server.
- Set up a secure JMX connector.

The class `JMXEndpoint` manages the JMX management lifecycle for a single Esper `EPServiceProvider` instance. Use the `start` and `destroy` methods to control EsperJMX lifecycle.

The `JMXEndpointConfiguration` class is the configuration information for `JMXEndpoint`. Your application must provide one of the EsperJMX connector configuration instances via the `setConnectorConfiguration` method.

The sections below describe the options available for JMX connector configuration.

The `JMXEndpointConfiguration` and the `ConnectorConfig` classes also provide all settings described in *Section 2.1.3, “Plug-In Configuration Properties”*.

### 2.2.1. Steps to Use the JVM Platform MBeanServer

The next code explains how to start a JMX endpoint using the Java VM platform MBeanServer provided by your Java virtual machine.

```
EPServiceProvider engine = EPServiceProviderManager.getDefaultProvider();
// ...register statements and event types

// Indicate that the platform MBeanServer should be used
ConnectorConfigPlatform platformConfig = new ConnectorConfigPlatform();

// Configure and start EsperJMX endpoint
JMXEndpointConfiguration jmxConfig = new JMXEndpointConfiguration();
jmxConfig.setConnectorConfiguration(platformConfig);
```

```
JMXEndpoint endpoint = new JMXEndpoint(engine, jmxConfig);
endpoint.start();
```

When JMX management is not longer needed, the `destroy` method provided by the endpoint un-registers all MBean objects:

```
endpoint.destroy();
```

## 2.2.2. Steps to Use a RMI Registry and Connector

This code snippet leads to the JMX endpoint to create a RMI registry and a JMX connector.

The resulting URL is:

`service:jmx:rmi://`*[hostname:port]*`/jndi/rmi://`*[hostname:port]*`/com.espertech.esper`.

```
EPServiceProvider engine = EPServiceProviderManager.getDefaultProvider();
// ...register statements and event types

ConnectorConfigRMIRegistry rmiConfig = new ConnectorConfigRMIRegistry();
rmiConfig.setConnectorPort(1071);
rmiConfig.setRegistryPort(1080);

JMXEndpointConfiguration jmxConfig = new JMXEndpointConfiguration();
jmxConfig.setConnectorConfiguration(rmiConfig);
JMXEndpoint endpoint = new JMXEndpoint(engine, jmxConfig);
endpoint.start();
```

The service URL for the example is:

`service:jmx:rmi://localhost:1071/jndi/rmi://localhost:1080/com.espertech.esper`

The `JMXEndpoint` class creates an RMI registry via `LocateRegistry.createRegistry` and a connecor via `JMXConnectorServerFactory. newJMXConnectorServer`.

For secure RMI connections, the `ConnectorConfigRMIRegistrySecure` connector configuration is provided.

## 2.2.3. Steps to Use an Application-Provided MBeanServer

This example demonstrates how your application may pass any `MBeanServer` instance it may obtain from its environment:

```
EPServiceProvider engine = EPServiceProviderManager.getDefaultProvider();
```

```
// ...register statements and event types

ConnectorConfigProvided             providedConfig             =            new
 ConnectorConfigProvided(myMBeanServer);

JMXEndpointConfiguration jmxConfig = new JMXEndpointConfiguration();
jmxConfig.setConnectorConfiguration(providedConfig);
JMXEndpoint endpoint = new JMXEndpoint(engine, jmxConfig);
endpoint.start();
```

# Chapter 3. MBeans

EsperJMX provides MBeans as summarized in this chapter.

For detailed information on the functions available, please consult the JavaDoc documentation.

By default, EsperJMX registers a new `StatementMBean` for each new statement created within an Esper engine. When a statement is destroyed, EsperJMX unregisters the `StatementMBean` representing the statement.

Named window MBeans and listener MBeans also also dynamically registered and unregistered. However by default configuration, EsperJMX does not create MBeans for listeners.

## 3.1. AdministratorMBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=Administrator`.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider.

This MBean provides statement management functions, metrics concerning statement and listener counts as well as configuration information.

## 3.2. RuntimeMBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=Runtime`.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider.

This MBean provides runtime metrics, variable and named window information and allows on-demand fire-and-forget queries against named windows.

### Table 3.1. Attributes

| Name | Description |
|---|---|
| TimeHandleCount | The number of schedule entries unique by time of schedule entry. For example, a schedule may have 3 points of interest (TimeHandleCount is 3) such as at 1pm, 2pm and 3pm, wherein at any of these 3 times one or more statements must perform some processing. |
| FurthestTimeHandle | The furthest outstanding time evaluation of any schedule entry, i.e. 3pm in the example above. |
| EvaluatedCount | The number of events evaluated, including insert-into events. |
| RoutedInternalCount | Number of insert-into events. |

| Name | Description |
|---|---|
| RoutedExternalCount | Number of routed events, i.e. when a listener or other application code calls one of the `route` methods. |
| TimerDriftAverage | For use with internal timer only. The *drift* is the absolute value of the `ScheduledFuture.getDelay` calls (please see the Java documentation for more information on the delay computation). The `TimerDriftAverage` is the sum of the absolute value of the `ScheduledFuture.getDelay` calls divided by the number of timer invocations. |
| TimerMaxDrift | For use with internal timer only. The highest value of the drift as defined above. |
| TimerLastDrift | For use with internal timer only. The last value of the drift as defined above. |
| EvaluatedAvgPerSecond | The `EvaluatedCount` as defined above divided by the number of seconds elapsed since. |
| RoutedInternalAvgPerSecond | The `RoutedInternalCount` as defined above divided by the number of seconds elapsed since. |
| RoutedExternalAvgPerSecond | The `RoutedExternalCount` as defined above divided by the number of seconds elapsed since. |
| ElapsedSecondsSinceLastReset | The number of seconds elapsed since. |

## 3.3. StatementMBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=Statement,name=`*[statement_name]*.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider. The value of *statement_name* is the name of the `EPStatement` statement.

This MBean provides statement details, allows iteration over statement result and allows subscription to statement results via `StmtStreamNotifierType1MBean`.

## 3.4. NamedWindowMBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=NamedWindow,name=`*[named_window_name]*.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider. The value of *named_window_name* is the name of the named window.

This MBean provides the number of events held by named window.

## 3.5. StmtStreamNotifierType1MBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=StmtStreamNotifierType1MBean,name=`*[statement_name]*.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider. The value of *statement_name* is the name of the `EPStatement` statement that provides statement results as JMX notifications.

This MBean is dynamically registered upon subscription to statement results via `StatementMBean` and provides JMX notifications of statement results as an object array in the notification object user data. The object array follow the same data format as returned by `fireAndForgetQueryType1` and as explained in more detail by the example.

Use the `addStreamNotifierType1` method to add a subscriber for statement results for a statement by providing a subscriber id. EsperJMX registers the `StmtStreamNotifierType1MBean` when the first subscriber is added, and deregisters it when the last subscriber is removed via `removeStreamNotifierType1`.

## 3.6. ListenerMBean

The object name for this MBean is `com.espertech.esper-`*[service_uri]*`:type=Listener, name=`*[statement_name]-[listener_classname]@[listener_hash]*.

The value of *service_uri* is the URI of the `EPServiceProvider` instance, or `default-provider` for the default provider. The value of *statement_name* is the name of the `EPStatement` statement. The value of *listener_classname* and *listener_hash* are class name and hash code of the listener object.

By default the option to register listener MBeans is disabled. If enabled, the MBean provides listener information and the names of the statements the listener is registered for.

# Chapter 4. Examples

In order to compile and run the samples please follow the below instructions:

1. Make sure Java 1.5 or greater is installed and the JAVA_HOME environment variable is set.
2. Copy the Esper distribution jar file and its runtime dependencies to `esperjmx/lib`.
3. Open a console window and change directory to examples/etc.
4. Run "setenv.bat" (Windows) or "setenv.sh" (Unix) to verify your environment settings.
5. Run "compile.bat" (Windows) or "compile.sh" (Unix) to compile the examples.
6. Now you are ready to run the examples. Further information to running each example can be found in the "examples/etc" folder in file "readme.txt".
7. Modify the logger logging level in the "log4j.xml" configuration file changing DEBUG to INFO on a class or package level to reduce the volume of text output.

## 4.1. JMX-enabled Esper Engine and JMX Client

This example demonstrates:

- How to set up EsperJMX.
- How to connect with a JMX console such as "jconsole".
- How to connect with a Java client and invoke methods on MBeans remotely.
- How to run an on-demand fire-and-forget query against a named window.

### 4.1.1. Overview

After following above instructions, use the script `run_trafficsim_server.sh` (Linux) or `run_trafficsim_server.bat` (Windows) to start the server. The server uses EsperJMX configured to start an RMI registry and JMX connector on the port provided by the properties file `trafficexample_config.properties` (port 1099).

After starting the server, you may verify server operation with a JMX console such as `jconsole`. To connect, enter the service url `service:jmx:rmi:///jndi/rmi://localhost:1099/com.espertech.esper` into the Remote Process textbox and press Ok. Replace the hostname `localhost` in the service URL with the host you have started the server.

Use the script `run_trafficsim_client.sh` (Linux) or `run_trafficsim_client.bat` (Windows) to start the client. The client connects to the same port, obtains a proxy to `RuntimeMBean` and executes a fire-and-forget on-demand query on a named window that contains real-time traffic data.

This example is out of the transportation domain: events processed are `TrainLeaveEvent` events indicating a train leaving a station. The example creates a named window to hold the last 10 minutes of events for querying.

## 4.1.2. Example Configuration

Since every Java VM has its own means of starting and providing a connector for the Java VM platform MBeanServer, this example does not use the JavaVM platform MBeanServer and instead starts an RMI registry and JMX connector on the port provided by the properties file `trafficexample_config.properties` (port 1099). This section outlines the steps to use the Sun JavaVM platform MBeanServer with the example.

The platform MBeanServer provides additional information about the Java VM, such as memory use and threading. It may therefore be desired by your application to use the platform MBeanServer with EsperJMX.

To use the platform MBeanServer instead of the RMI registry, edit the file `trafficexample_config.properties`:

1. Set the property `use-platform-mbean-server` to `true`.
2. Set the property `service-url` to `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`. This value is used by the example client to connect to the platform MBeanServer.

If using the Sun Java VM, edit the file `run_trafficsim_server` and add the following system properties: `-Dcom.sun.management.jmxremote.port=1099 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false`

## 4.1.3. On-Demand Query Example

The example client first obtains the `RuntimeMBean` proxy as this code snippet shows:

```
JMXServiceURL jmxServiceURL = new JMXServiceURL(serviceURL);
JMXConnector jmxc = JMXConnectorFactory.connect(jmxServiceURL, null);
MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

RuntimeMBean             runtimeMBean             =            (RuntimeMBean)
 MBeanServerInvocationHandler.newProxyInstance(
  mbsc, new ObjectName("com.espertech.esper-default-provider:type=Runtime"),
  RuntimeMBean.class, false);
```

The `fireAndForgetQueryType1` method on `RuntimeMBean` returns a object array in which the first item is the property names and types and the second item the 2-dimensional array of rows and columns.

The client executes the query and prints a table of query results:

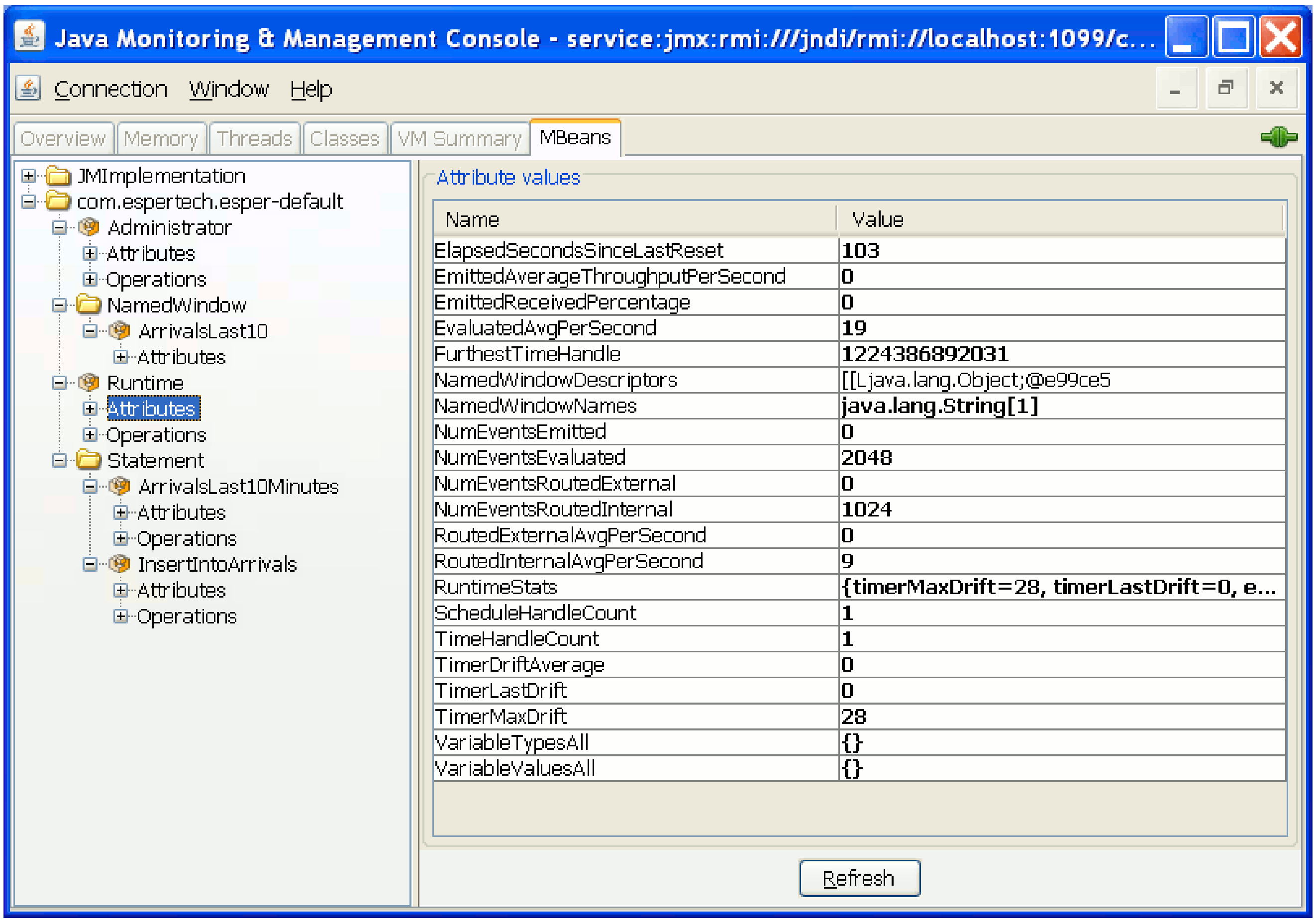
```
String query = "select stationName, count(*) from ArrivalsLast10 " +
  "group by stationName order by count(*) desc";

Object[] result = runtimeMBean.fireAndForgetQueryType1(query, -1);
String[][] propertyNames = (String[][]) result[0];
Object[][] rows = (Object[][]) result[1];

String line = String.format("%15s %10s", "Station", "count");
for (int i = 0; i < rows.length; i++) {
  line = String.format("%15s %10s", rows[i][0], rows[i][1]);
  log.info(line);
}
```

### 4.1.4. JConsole View

A screenshot of `jconsole` attached to the server is shown here:



| Name | Value |
|---|---|
| ElapsedSecondsSinceLastReset | 103 |
| EmittedAverageThroughputPerSecond | 0 |
| EmittedReceivedPercentage | 0 |
| EvaluatedAvgPerSecond | 19 |
| FurthestTimeHandle | 1224386892031 |
| NamedWindowDescriptors | [[Ljava.lang.Object;@e99ce5 |
| NamedWindowNames | java.lang.String[1] |
| NumEventsEmitted | 0 |
| NumEventsEvaluated | 2048 |
| NumEventsRoutedExternal | 0 |
| NumEventsRoutedInternal | 1024 |
| RoutedExternalAvgPerSecond | 0 |
| RoutedInternalAvgPerSecond | 9 |
| RuntimeStats | {timerMaxDrift=28, timerLastDrift=0, e... |
| ScheduleHandleCount | 1 |
| TimeHandleCount | 1 |
| TimerDriftAverage | 0 |
| TimerLastDrift | 0 |
| TimerMaxDrift | 28 |
| VariableTypesAll | {} |
| VariableValuesAll | {} |