Table of Contents

1. Working with CEP Tooling ................................................................. 6
   1.1 Installing the CEP Tooling Plug-In ............................................. 6
   1.2 Creating an Analytics Project ....................................................... 10
2. WSO2 Complex Event Processor Documentation ...................................... 16
   2.1 About CEP .................................................................................. 18
      2.1.1 Introduction to CEP ............................................................... 19
      2.1.2 Features .............................................................................. 19
      2.1.3 Architecture ....................................................................... 22
      2.1.4 About this Release ............................................................... 23
   2.2 Getting Started ........................................................................... 23
      2.2.1 Quick Start Guide ................................................................. 24
      2.2.2 Downloading the Product ..................................................... 54
      2.2.3 Installation Prerequisites ....................................................... 55
   2.3 User Guide .................................................................................. 251
      2.3.1 Understanding Event Streams ............................................... 252
      2.3.2 Collecting Data ..................................................................... 260
         2.3.2.1 Publishing Data to CEP .................................................. 260
            2.3.2.1.1 Publishing Data Using Event Simulation .................. 260
            2.3.2.1.2 Publishing Binary/Thrift Data using Java Client ...... 265
            2.3.2.1.3 Publishing JSON/REST Data using Java Client ...... 276
            2.3.2.1.4 Publishing Data Through Other Protocols ........... 281
      2.3.2.2 Configuring CEP to Receive Data ....................................... 288
         2.3.2.2.1 Configuring Event Receivers .................................... 288
         2.3.2.2.2 Input Mapping Types ............................................... 336
         2.3.2.2.3 Building Custom Event Receivers ............................. 348
      2.3.3 Analyzing Data ..................................................................... 355
         2.3.3.1 Creating a Standalone Execution Plan ............................. 355
         2.3.3.2 Creating a Storm Based Distributed Execution Plan .... 360
         2.3.3.3 SiddhiQL Guide 3.1 ......................................................... 364
            2.3.3.3.1 Inbuilt Functions .................................................... 396
            2.3.3.3.2 Inbuilt Windows ....................................................... 403
            2.3.3.3.3 Inbuilt Aggregate Functions ................................... 411
            2.3.3.3.4 Writing Extensions to Siddhi ................................. 414
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3.5</td>
<td>Siddhi Extensions</td>
<td>440</td>
</tr>
<tr>
<td>2.3.3.6</td>
<td>Using Siddhi as a Library</td>
<td>499</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Communicating Results</td>
<td>502</td>
</tr>
<tr>
<td>2.3.4.1</td>
<td>Configuring CEP to Create Alerts</td>
<td>502</td>
</tr>
<tr>
<td>2.3.4.1.1</td>
<td>Event Publisher Types</td>
<td>507</td>
</tr>
<tr>
<td>2.3.4.1.2</td>
<td>Output Mapping Types</td>
<td>565</td>
</tr>
<tr>
<td>2.3.4.1.3</td>
<td>Building Custom Event Publishers</td>
<td>578</td>
</tr>
<tr>
<td>2.3.4.2</td>
<td>Visualizing Results</td>
<td>588</td>
</tr>
<tr>
<td>2.3.4.2.1</td>
<td>Visualizing Results in the Analytics Dashboard</td>
<td>588</td>
</tr>
<tr>
<td>2.3.4.2.2</td>
<td>Geo Dashboard</td>
<td>596</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Packaging Artifacts as C-App Archive</td>
<td>597</td>
</tr>
<tr>
<td>2.3.6</td>
<td>Monitoring and Debugging</td>
<td>601</td>
</tr>
<tr>
<td>2.3.6.1</td>
<td>Event Metrics</td>
<td>606</td>
</tr>
<tr>
<td>2.3.6.2</td>
<td>Event Tracer</td>
<td>610</td>
</tr>
<tr>
<td>2.3.6.3</td>
<td>Siddhi Try It Tool</td>
<td>612</td>
</tr>
<tr>
<td>2.3.7</td>
<td>Managing CEP Artifacts Via the Template Manager</td>
<td>614</td>
</tr>
<tr>
<td>2.3.7.1</td>
<td>Configuring a Template for Template Manager</td>
<td>615</td>
</tr>
<tr>
<td>2.3.7.2</td>
<td>Using Templates</td>
<td>628</td>
</tr>
<tr>
<td>2.4</td>
<td>Admin Guide</td>
<td>633</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Performance Tuning Recommendations</td>
<td>634</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Monitoring WSO2 CEP</td>
<td>640</td>
</tr>
<tr>
<td>2.4.2.1</td>
<td>JMX-Based Monitoring</td>
<td>640</td>
</tr>
<tr>
<td>2.4.2.2</td>
<td>System Statistics</td>
<td>648</td>
</tr>
<tr>
<td>2.4.2.3</td>
<td>Event Monitoring</td>
<td>650</td>
</tr>
<tr>
<td>2.4.2.4</td>
<td>SOAP Tracer</td>
<td>651</td>
</tr>
<tr>
<td>2.4.2.5</td>
<td>SOAP Message Flows</td>
<td>652</td>
</tr>
<tr>
<td>2.4.2.6</td>
<td>Monitoring with WSO2 Carbon Metrics</td>
<td>654</td>
</tr>
<tr>
<td>2.4.2.6.1</td>
<td>Enabling Metrics and Storage Types</td>
<td>654</td>
</tr>
<tr>
<td>2.4.2.6.2</td>
<td>Configuring Metrics Properties</td>
<td>660</td>
</tr>
<tr>
<td>2.4.2.6.3</td>
<td>Using JVM Metrics</td>
<td>662</td>
</tr>
<tr>
<td>2.4.2.6.4</td>
<td>Using Event Metrics</td>
<td>665</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Deployment Automation Guide</td>
<td>668</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Clustered Deployment</td>
<td>671</td>
</tr>
<tr>
<td>2.4.5</td>
<td>User Management</td>
<td>671</td>
</tr>
<tr>
<td>2.4.5.1</td>
<td>Introduction to User Management</td>
<td>671</td>
</tr>
<tr>
<td>2.4.5.2</td>
<td>Adding and Managing Users and Roles</td>
<td>673</td>
</tr>
<tr>
<td>2.4.5.3</td>
<td>Realm Configuration</td>
<td>677</td>
</tr>
<tr>
<td>2.4.5.4</td>
<td>Default JDBC User Store Configuration</td>
<td>678</td>
</tr>
<tr>
<td>2.4.5.5</td>
<td>Changing the RDBMS</td>
<td>681</td>
</tr>
<tr>
<td>2.4.5.6</td>
<td>Default LDAP User Store Configuration</td>
<td>681</td>
</tr>
<tr>
<td>2.4.5.7</td>
<td>Configuring Secondary User Stores</td>
<td>684</td>
</tr>
<tr>
<td>2.4.5.7.1</td>
<td>Configuring Primary User Stores</td>
<td>686</td>
</tr>
<tr>
<td>2.4.6</td>
<td>Working with Databases</td>
<td>706</td>
</tr>
<tr>
<td>2.4.6.1</td>
<td>Setting up the Physical Database</td>
<td>707</td>
</tr>
<tr>
<td>2.4.6.1.1</td>
<td>Setting up IBM DB2</td>
<td>707</td>
</tr>
<tr>
<td>2.4.6.1.2</td>
<td>Setting up Derby</td>
<td>713</td>
</tr>
<tr>
<td>2.4.6.1.3</td>
<td>Setting up H2</td>
<td>720</td>
</tr>
<tr>
<td>2.4.6.1.4</td>
<td>Setting up IBM Informix</td>
<td>729</td>
</tr>
<tr>
<td>2.4.6.1.5</td>
<td>Setting up Microsoft SQL</td>
<td>732</td>
</tr>
<tr>
<td>2.4.6.1.6</td>
<td>Setting up MySQL</td>
<td>736</td>
</tr>
</tbody>
</table>
2.4.6.1.7 Setting up MySQL Cluster .......................................................... 741
2.4.6.1.8 Setting up OpenEdge ................................................................. 741
2.4.6.1.9 Setting up Oracle ..................................................................... 744
2.4.6.1.10 Setting up Oracle RAC ............................................................ 748
2.4.6.1.11 Setting up PostgreSQL ............................................................ 753
2.4.6.1.12 Setting up MariaDB ............................................................... 757
2.4.6.2 Managing Datasources ................................................................. 761
  2.4.6.2.1 Adding Datasources .............................................................. 762
  2.4.6.2.2 Configuring an RDBMS Datasource ......................................... 762
  2.4.6.2.3 Configuring a Custom Datasource ........................................... 771
2.4.7 Registry ......................................................................................... 773
  2.4.7.1 Introduction to Registry ............................................................. 773
  2.4.7.2 Managing the Registry ............................................................... 774
    2.4.7.2.1 Managing Breadcrumb .......................................................... 774
    2.4.7.2.2 Managing Metadata ............................................................. 775
    2.4.7.2.3 Managing Properties ............................................................ 778
    2.4.7.2.4 Managing Entries and Content ............................................. 780
    2.4.7.2.5 Managing Role Permissions .................................................. 787
  2.4.7.3 Searching the Registry .............................................................. 788
  2.4.7.4 Sharing Registry Space Among Multiple Products ....................... 790
    2.4.7.4.1 All Partitions in a Single Server ........................................... 791
    2.4.7.4.2 Config and Governance Partitions in a Remote Registry .......... 791
    2.4.7.4.3 Governance Partition in a Remote Registry ............................ 796
    2.4.7.4.4 Config and Governance Partitions in Separate Nodes ............... 801
2.4.8 Using Asymmetric Encryption ....................................................... 811
  2.4.8.1 Configuring Keystores in WSO2 Products ................................... 814
  2.4.8.2 Creating New Keystores ............................................................ 816
  2.4.8.3 Managing Keystores with the UI ............................................... 819
2.4.9 Fixing Security Vulnerabilities ....................................................... 822
2.4.10 Logging ....................................................................................... 823
2.4.11 Feature Management ................................................................... 827
  2.4.11.1 Introduction to Feature Management ...................................... 827
  2.4.11.2 Installing and Managing Features ......................................... 828
  2.4.11.3 Recovering from Unsuccessful Feature Installation ................... 831
2.4.12 Transports .................................................................................... 833
  2.4.12.1 Introduction to Transports ....................................................... 833
  2.4.12.2 Server Transports ................................................................. 834
    2.4.12.2.1 HTTP Servlet Transport ...................................................... 834
    2.4.12.2.2 HTTPS Servlet Transport .................................................... 837
    2.4.12.2.3 Local Transport ................................................................. 838
    2.4.12.2.4 TCP Transport ................................................................. 839
  2.4.12.3 Configuring Transports Globally ............................................ 839
  2.4.12.4 Configuring Transport Level Security .................................... 844
2.4.13 Enabling Java Security Manager .................................................. 846
2.4.14 Multitenancy ............................................................................... 849
  2.4.14.1 Managing Tenants ................................................................. 850
2.4.15 Integration ................................................................................... 851
  2.4.15.1 Installing R to work with WSO2 CEP ...................................... 851
  2.4.15.2 Supporting Different Transports ............................................ 852
  2.4.15.3 Installing Machine Learner Features ....................................... 855
Working with CEP Tooling

This section walks you through how to install the CEP tooling plug-in, based on WSO2 Developer Studio Kernel, using which you can create and manage CEP artifacts.

- Installing the CEP Tooling Plug-In
- Creating an Analytics Project
Installing the CEP Tooling Plug-In

The CEP tooling plug-in is based on WSO2 Developer Studio Kernel, which gives the capabilities of a complete Eclipse-based development environment for the CEP. You can develop services, features and artifacts and manage their links and dependencies through a simplified graphical editor.

You can install multiple WSO2 product plug-ins on top of the same Eclipse instance.

There are 3 ways to install the tooling plug-in. The following two methods require you to install Eclipse Luna separately in your system, if you do not have it already:

- Install the plug-in on Eclipse Luna using the P2 URL
- Install the plug-in on Eclipse Luna using the P2.zip file

The third method uses a complete plug-in installation with pre-packaged Eclipse, so that you do not have to install Eclipse separately:

- Install the plug-in with pre-packaged Eclipse

Install the plug-in on Eclipse Luna using the P2 URL

1. Make sure you have Eclipse IDE for Java EE Developers (Luna SR2) installed.
2. Open Eclipse and click Help > Install New Software.
3. On the dialog box that appears, click Add.

4. Give Analytics Tool as the name and http://product-dist.wso2.com/p2/developer-studio-
kernel/4.0.0/analytics-tools/releases/ as the location and click OK.

5. Select all the check boxes and click Next.

6. Read and accept the license agreements and click Finish.

7. If a security warning appears saying that the authenticity or validity of the software cannot be established, click OK.

8. Restart Eclipse to complete the installation.

Install the plug-in on Eclipse Luna using the P2.zip file
1. Make sure you have Eclipse IDE for Java EE Developers (Luna SR2) installed.
3. Open Eclipse and click Help > Install New Software.
4. On the dialog box that appears, click Add.

5. Give Analytics Tool as the name and click Archive.

6. Navigate to the downloaded .zip file and click OK.
7. Select all the check boxes and click Next.
8. Read and accept the license agreements and click Finish.

9. If a security warning appears saying that the authenticity or validity of the software cannot be established, click OK.

10. Restart Eclipse to complete the installation.
Install the plug-in with pre-packaged Eclipse

Download the distribution according to your operating system from http://product-dist.wso2.com/products/developer-studio-for-analytics/4.0.0/wso2-developer-studio-for-analytics_4.0.0.zip.
Creating an Analytics Project

This page describes how to create artifacts for an Analytics project. It contains the following sections:

- Creating an Analytics project
- Creating a stream
- Creating an execution-plan
- Creating a publisher
- Creating a receiver
- Deploying the Analytics project

The Analytics plug-in currently works only for WSO2 Complex Event Processor artifacts such as event stream, event publisher, event receiver and execution-plan. In the future, the functionality will include WSO2 Data Analytics Server artifacts as well.

Creating an Analytics project

You can create an Analytics project to save all your Analytics related artifacts such as execution-plans, streams, publishers and receivers.

To create an Analytics project

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Analytics Project in the Analytics category.
2. If you want to create this Analytics project using artifacts from the deployment server, select the **New Analytics Project By Artifacts in Deployment Server** (<CARBON_HOME>/repository/deployment/server/) option. Otherwise, leave the **Create New Analytics Project** option selected. Click **Next**.

3. Do the following:
   1. Type a unique name for the project.
   2. Optionally specify the working set, if any, that you want to include in this project.
   3. Optionally specify the location where you want to save the project (or leave the default location specified).
   4. If you selected the **New Analytics Project By Artifacts in Deployment Server** option in the previous step, click **Browse** and navigate to the <PRODUCT_HOME>/repository/deployment/server folder containing the artifact file folders.

4. A Maven POM file will be generated automatically for this project. If you want to include parent POM information in the file from another project in this workspace, click **Next**, click the **Specify Parent from Workspace** check box, and then select the parent project.

5. Click **Finish**.

6. If you specified a deployment server folder containing artifact file folders, specify whether you want to open those files now.

The new project has now been created in the workspace. If you browse inside the project, you will see a project structure as shown below, with folders created for different resources such as execution-plan, publisher, receiver, and stream.
Creating a stream

Events are the lifeline of WSO2 CEP/DAS. Data is not only processed as events, but interactions with external systems also uses events. An event is a unit of data, and an event stream is a sequence of events of a particular type. The type of events can be defined as an event stream definition. Definitions of the event streams are stored in .json files as deployable artifacts in the <PRODUCT_HOME>/repository/deployment/server/eventstreams directory. For more information, see Understanding Event Streams.

Creating a new stream

Follow the steps below to create a new stream. Alternatively, you can import an existing stream.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Stream in the Analytics category.
2. Select Create New Stream and click Next.
3. Type a unique name for the stream.
4. Do one of the following:
   - To save the stream in an existing Analytics project in your workspace, click Browse and select that project.
   - To save the stream in a new Analytics project, click Create new Project and create the new project.
5. Click Finish. The stream is created in the stream folder under the Analytics project you specified, and the stream is open in the editor.

Importing a stream

Follow the steps below to import an existing stream from a .json file into an Analytics project. Alternatively, you can create a new stream.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Stream in the Analytics category.
2. Select Import Stream and click Next.
3. Type a unique name for the stream.
4. Specify the stream file by typing its full path name or clicking Browse and navigating to the file.
5. In the Save Stream In field, specify an existing Analytics project in your workspace where you want to save the stream, or click Create new Project to create a new Analytics project and save the stream there.
6. Click Finish. The stream is created in the stream folder under the Analytics project you specified, and the stream is open in the editor.

Creating an execution-plan

WSO2 CEP uses execution-plans to store event processing logic. An execution-plan is bound to an instance of the Siddhi complex event processing runtime, which is responsible for the actual processing of the events. The event processor allows users to configure multiple execution-plans by providing multiple isolated event processing environments per execution-plan. A typical execution-plan consists of a set of queries and related input and output event streams. Definitions of the event streams are stored in .siddhiql files as deployable artifacts in the <PRODUCT_HOME>/repository/deployment/server/eventstreams directory.
CT _HOME>/repository/deployment/server/executionplans directory. For more information, see Creating a Standalone Execution Plan.

Creating a new execution-plan

Follow the steps below to create a new execution-plan. Alternatively, you can import an existing execution-plan.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Execution-plan in the Analytics category.
2. Select Create New Execution-plan and click Next.
3. Type a unique name for the execution-plan.
4. Do one of the following:
   - To save the execution-plan in an existing Analytics project in your workspace, click Browse and select that project.
   - To save the execution-plan in a new Analytics project, click Create new Project and create the new project.
5. Click Finish. The execution-plan is created in the execution-plan folder under the Analytics project you specified, and the execution-plan is open in the editor.

Importing an execution-plan

Follow the steps below to import an existing execution-plan from a .siddhiql file into an Analytics project. Alternatively, you can create a new execution-plan.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Execution-plan in the Analytics category.
2. Select Import Execution-plan and click Next.
3. Type a unique name for the execution-plan.
4. Specify the execution-plan file by typing its full path name or clicking Browse and navigating to the file.
5. In the Save Execution-plan in field, specify an existing Analytics project in your workspace where you want to save the execution-plan, or click Create new Project to create a new Analytics project and save the execution-plan there.
6. Click Finish. The execution-plan is created in the execution-plan folder under the Analytics project you specified, and the execution-plan is open in the editor.

Creating a publisher

Event publishers publish events via various transport protocols. Event publisher configurations are stored in .xml files as deployable artifacts in the <PRODUCT _HOME>/repository/deployment/server/eventpublishers directory. For more information, see Event Publisher Types.

Creating a new publisher

Follow the steps below to create a new publisher. Alternatively, you can import an existing publisher.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Publisher in the Analytics category.
2. Select Create New Publisher and click Next.
3. Type a unique name for the publisher.
4. Do one of the following:
   - To save the publisher in an existing Analytics project in your workspace, click Browse and select that project.
   - To save the publisher in a new Analytics project, click Create new Project and create the new project.
5. Click Finish. The publisher is created in the publisher folder under the Analytics project you specified, and the publisher is open in the editor.

Importing a publisher
Follow the steps below to import an existing publisher from a .xml file into an Analytics project. Alternatively, you can create a new publisher.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Publisher in the Analytics category.
2. Select Import Publisher and click Next.
3. Type a unique name for the publisher.
4. Specify the publisher file by typing its full path name or clicking Browse and navigating to the file.
5. In the Save Publisher In field, specify an existing Analytics project in your workspace where you want to save the publisher, or click Create new Project to create a new Analytics project and save the publisher there.
6. Click Finish. The publisher is created in the publisher folder under the Analytics project you specified, and the publisher is open in the editor.

Creating a receiver

Events are received by the WSO2 CEP/DAS server using event receivers, which manage the event retrieval process. Event receiver configurations are stored in .xml files as deployable artifacts in the <PRODUCT_HOME>/repository/deployment/server/eventreceivers directory. For more information, see Configuring Event Receivers.

Creating a new receiver

Follow the steps below to create a new receiver. Alternatively, you can import an existing receiver.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Receiver in the Analytics category.
2. Select Create New Receiver and click Next.
3. Type a unique name for the receiver.
4. Do one of the following:
   - To save the receiver in an existing Analytics project in your workspace, click Browse and select that project.
   - To save the receiver in a new Analytics project, click Create new Project and create the new project.
5. Click Finish. The receiver is created in the receiver folder under the Analytics project you specified, and the receiver is open in the editor.

Importing a receiver

Follow the steps below to import an existing receiver from a .xml file into an Analytics project. Alternatively, you can create a new receiver.

1. Open the Developer Studio Dashboard (click Developer Studio > Open Dashboard) and click Receiver in the Analytics category.
2. Select Import Receiver and click Next.
3. Type a unique name for the receiver.
4. Specify the receiver file by typing its full path name or clicking Browse and navigating to the file.
5. In the Save Receiver In field, specify an existing Analytics project in your workspace where you want to save the receiver, or click Create new Project to create a new Analytics project and save the receiver there.
6. Click Finish. The receiver is created in the receiver folder under the Analytics project you specified, and the receiver is open in the editor.

Deploying the Analytics project

Once you have created the necessary artifacts in the Analytics project, you can create a Composite Application (C-App) project to group them and create a Composite Application Archive (CAR) file. Before creating the CAR file, make sure to change the server role of the created bundle to ComplexEventProcessor. To deploy it to WSO2 Complex Event Processor, you need to change it to the correct server role.
For more information, see Packaging Artifacts Into Deployable Archives.
WSO2 Complex Event Processor Documentation

**WSO2 Complex Event Processor** (CEP) identifies the most meaningful events within an event cloud, analyzes their impacts, and acts on them in real time. Built to be extremely high performing and massively scalable, it offers significant time saving and affordable acquisition.

**Get started with WSO2 CEP**

If you are new to using WSO2 Complex Event Processor, follow the steps given below to get started:

1. **Get familiar with WSO2 CEP**
   Understand the basics of the CEP and its architecture.

2. **Quick Start Guide**
   Download, install and run WSO2 CEP.

3. **Try out the Samples**
   Try out the CEP use cases.

**Deep dive into WSO2 CEP**

To know more about WSO2 CEP use the descriptions below to find the section you need, and then browse the topics in the left navigation panel. You can also use the Search box on the left to find a term or phrase in this documentation, or use the box in the top right-hand corner to search in all WSO2 product documentation.

To download a PDF of this document or a selected part of it, click [here](#) (only generate one PDF at a time). You can also use this link to export to HTML or XML.
About CEP

The topics in this section introduce WSO2 CEP, including the business cases it solves, its features, and architecture.

- Introduction to CEP
- Features
- Architecture
- About this Release

Introduction to CEP

Every enterprise’s transactions and activities consist of a constant stream of events. Enterprises that monitor these events in real time and quickly respond to their environments have greater business advantage over competitors. Complex event processing is about listening to events and detecting patterns in real-time, with minimum or no storage of events.

WSO2 Complex Event Processor (CEP) is a lightweight, easy-to-use, open source Complex Event Processing server (CEP). It identifies the most meaningful events within the event cloud, analyzes their impact, and acts on them in real-time. It’s built to be extremely high performing with WSO2 Siddhi and massively scalable using Apache Storm. WSO2 CEP is built up on the award-winning, WSO2 Carbon platform, which is based on the OSGi framework enabling better modularity for your service oriented architecture (SOA). WSO2 Carbon framework contains many enhanced features and optional components to customize the behaviour of the server through simple, point-and-click provisioning. The CEP can be tightly integrated with WSO2 Data Analytics Server, by adding support for recording and post processing events with Map-Reduce via Apache Spark, and WSO2 Machine Learner for predictive analytics.

WSO2 Complex Event Processor is released under Apache Software License Version 2.0, one of the most business-friendly licenses available today.

For information on WSO2 CEP features and architecture, see the next sections.

Features

For the relevant versions of the applications used as features, see Compatibility of WSO2 Products.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| Extremely High Performant Processing Engine | - Process about 100K+ events per second on single-server commodity hardware  
- Powered by WSO2 Siddhi |
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| Powerful and Extensible Query Language for Temporal Event Stream Processing | - SQL-like query language  
- Filters events by conditions  
- Creates new event streams by merging existing event streams  
- Executes temporal queries using various windows  
- Detects and responds to various event patterns and sequences  
- Instigates with In-Memory and RDBMS data stores for correlating historical data with real-time  
- Partitioning support to achieve parallel processing  
- Maintenance of execution plan templates |
| User-friendly Execution Management                                      | - Suitable for business users to edit and manage realtime execution logic  
- Users interact with forms instead of with SQL Queries  
- Queries and logics are templated to hide the complexity |
| Support for Rich Event Model                                           | - Events are modelled as tuples of metadata, correlation data and payload data.  
- Support for different property types including integral types, floating types, string and Boolean |
| Extremely High Performant Event Capturing and Delivery Framework Over Binary and Apache Thrift | - Java data publisher agent to plug into any Java based system  
- Data publisher agent support in other languages (C/C++/C#) via Thrift language bindings  
- Horizontally scalable to support very large event volumes |
| Easily Integrates with any Enterprise System for Event Capture         | - RESTful HTTP protocol with JSON, XML and Text message formats  
- Map, JSON, XML and Text messages support via JMS  
- SOAP over any transport protocol  
- Kafka, MQTT, File, Websocket and Email protocols with JSON, XML and Text messages |
| Supports Multiple Alert Notifying Mechanisms                           | - XML, JSON, Map, Text events via JMS protocol  
- E-mail, SMS notifications  
- Service calls to notify RESTful and Web services  
- MySQL and Cassandra writers  
- Kafka, MQTT, File, Websocket protocols supporting JSON, XML and Text messages alerts |
| IoT (Internet of Things) Integration                                   | - MQTT protocol |
## Debugging Support
- Event tracing, Event Flow visualization and Event Simulation capabilities
- Siddhi TryIt to try out Siddhi queries

## Massively Scalable
- Distributed Real Time Processing
- Support highly scalable deployments with Apache Storm and WSO2 Siddhi
- Deploys numbers of queries partitioning them into different servers

## Highly Available Deployment
- Run CEP in active-passive deployment with zero downtime
- Coordinates event and state sharing

## Support for Long Duration Queries
- Queries can have lifetimes that are much greater than server uptime
- Supported via HA deployment and persistence snapshot deployment
- Support periodic snapshots that can store all state information and windows to a scalable persistence store
- Pluggable persistent stores

## Tightly Integrates with WSO2 Data Analytics Server
- Leverage Lambda architecture to integrate post-processed data with real-time data processing
- Supports recording and post processing of events with Map-Reduce via Apache Spark

## Built-in Support for WSO2 Products
- Pre-built event sources from other WSO2 products

## Lightweight, Developer Friendly and Easy to Deploy
- Server customization via provisioning of features
- Deployment options from on-premise to private or public clouds, without configuration changes
- Integrated with WSO2 Developer Studio, the Eclipse-based IDE for all WSO2 products

## Manage and Monitor
- Comprehensive management and monitoring Web console with enterprise-level security
- Built-in collection and monitoring of standard access and performance statistics
- Flexible logging support with integration to enterprise logging systems
- In build support for rich representation of data via a dashboard
- Simulating capabilities via the Event Simulator.
Architecture

The diagram below depicts the architecture with its components.

The WSO2 CEP architecture consists of the following components:

- **Event receivers**
- **Event streams**
- **Event processors**
- **Event publishers**

**Event receivers**

Event receivers receive events that are coming to the CEP. WSO2 CEP supports the most common adapter implementations by default. For specific use cases, you can also plug custom adapters. For more information, see **Configuring Event Receivers**.

**Event streams**

Event streams contain unique sets of attributes of specific types that provide a structure based on which the events processed by the relevant event flow are selected. Event streams are stored as stream definitions in the file system via the data bridge stream definition store.

**Event processors**

Event processor handles actual event processing. It is the core event processing unit of the CEP. It manages different execution plans and processes events based on logic, with the help of different Siddhi queries. Event Processor gets a set of event streams from the Event Stream Manager, processes them using Siddhi engine, and triggers new events on different event streams back to the Event Stream Manager. For more information, see **Creating a Standalone Execution Plan**.

**Event publishers**

Event publishers publish events to external systems and store data to databases for future analysis. Like the event receivers, this component also has different adapter implementations. The most common ones are available by default in the CEP. You can implement custom adapters for specific use cases. For more information, see **Configuring Event Publishers**.
About this Release

What is new in this release

WSO2 CEP version 4.2.0 is the successor of version 4.1.0. It contains the following new features and enhancements.

- Performance improvements to achieve reduced latency per event.
- Introduction of new inbuilt windows in Siddhi including `externalTimeBatch` and `externalTime`.
- Introduction of new aggregators in Siddhi including `Distinct Count`, `Forever Maximum` and `Forever Minimum`.
- Introduction of new functions in Siddhi including `Split`.
- Implementation of `Event Window` in Siddhi.
- Improvements to the Analytics Dashboard and the gadget generating tool.

Compatible WSO2 product versions

The following products were tested for compatibility with WSO2 CEP 4.2.0.

- Data Analytics Server 3.1.0
- Enterprise Service Bus 5.0.0
- Machine Learner 1.2.0
- Message Broker 3.1.0
- Identity Server 5.2.0

What is changed in this release

- The Execution Manager tool is replaced by the Template Manager tool with enhanced functionality.

Fixed issues

This release includes several bug fixes.

For a complete list of issues fixed in this release, see CEP 4.2.0 - Fixed Issues.

Known issues

For a complete list of known issues in this release, see CEP 4.2.0 - Known Issues.
Getting Started

The following topics show how to download, install, run and get started quickly with WSO2 CEP.

- Quick Start Guide
- Downloading the Product
- Installation Prerequisites
- Installing the Product
- Building from Source
- Upgrading from WSO2 CEP 4.1.0
- Running the Product
- Samples Guide

Quick Start Guide

To view a screencast of the Quick Start Guide, click here.

WSO2 Complex Event Processor monitors the transactions and activities of an enterprise (referred to as events), analyses them and presents the results to a range of interfaces in real time. Not having to store this information enables it to process and present information with greater speed.

The following sections walk you through the basic features of the CEP to get you started.

- Creating a simple event flow
  - Step 1: Add an event stream
  - Step 2: Add an event receiver
  - Step 3: Add an Event Publisher
  - Step 4: View the simple event flow
- Receive and log events
  - Step 5: Receive events via HTTP transport
- Processing events with an execution plan
  - Step 6: Add another event stream
  - Step 7: Add an execution plan
- Publish events in dashboard
  - Step 8: Add a UI event publisher
  - Step 9: Create a dashboard and a gadget
  - Step 10: Send Events to the HTTP Receiver via Curl Command
- Deploying execution plans using templates
  - Step 11: Create and deploy a template
  - Step 12: Configure a template
  - Step 13: View the elements of the event flow

Getting Started Video

You can also find the content here as the Getting Started Video

Before you begin,
Creating a simple event flow

An event flow refers to a specific combination of event streams, event receivers, event publishers and execution plans. The following steps describe how to define these elements of an event flow.

**Step 1: Add an event stream**

Event stream defines the events which goes through a particular flow by defining the event's attributes and it's types. An event stream can be created in the CEP Management Console as follows.

1. Log into the CEP Management Console and click on the **Main** tab. Under **Manage**, click **Streams** to open the **Available Event Streams** page.
2. Click **Add Event Stream** to open the **Define New Event Stream** page.
3. Enter information as follows to create a new event stream named `org.wso2.event.sensor.stream`.

---

1. Install *Oracle Java SE Development Kit (JDK)* version 1.7* or 1.8 and set the `JAVA_HOME` environment variable.
2. **Download** WSO2 CEP.
3. Start the CEP by going to `<CEP_HOME>/bin` using the command-line and executing `wso2server.bat` (for Windows) or `wso2server.sh` (for Linux).
### Event Stream Details

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream Name</td>
<td>org.wso2.event.sensor.stream</td>
</tr>
<tr>
<td>Event Stream Version</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

### Stream Attributes

**Click to add the attribute after entering the attribute name and attribute type.**

<table>
<thead>
<tr>
<th>Attribute Category</th>
<th>Attribute</th>
<th>Attribute Type</th>
</tr>
</thead>
</table>
Step 2: Add an event receiver

Events received by the CEP server have different formats such as XML, JSON and Map. Event receivers transform these events to WSO2 Events that can be directed to an event stream defined in the CEP.

In this step, you will create an event receiver named `httpReceiver` which directs events to the event stream named `org.wso2.event.sensor.stream` that was created in Step 1. The receiver can be created using the Management Console as follows.

1. Log into the CEP Management Console and click on the **Main** tab. Under **Manage**, click **Receivers** to open the **Available Receivers** page.
2. Click **Add Event Receiver** to open the **Create a New Event Receiver** page.
3. Enter information as follows to create the new event receiver named `httpReceiver`:

<table>
<thead>
<tr>
<th>Meta Data</th>
<th>timestamp</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>isPowerSaverEnabled</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>sensorId</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>sensorName</td>
<td>string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correlation Data</th>
<th>longitude</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>latitude</td>
<td>double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payload Data</th>
<th>humidity</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sensorValue</td>
<td>double</td>
</tr>
</tbody>
</table>

4. Click **Add Event Stream** to save the information.
Create a New Event Receiver

Parameter Name | Value
---|---
Event Receiver Name | httpReceiver
Input Event Adapter Type | http
Transports | all
Event Stream | org.wso2.event.sensor.stream
Message Format | json

4. Click **Add Event Receiver** to save the information.

**Step 3: Add an Event Publisher**

Event publishers publish events processed by the WSO2 servers to external applications. These events are published via HTTP, Kafka, JMS, etc. in **JSON, XML, Map, text, and WSO2Event formats** to various endpoints and data stores.
In this step, you will create an event publisher named loggerPublisher to publish events from the event stream named org.wso2.event.sensor.stream that was created in Step 1. Since the output event adapter type of this publisher is logger, the events published will be logged in the CEP CLI in text format. The publisher can be created using the Management Console as follows.

1. Log into the CEP Management Console and click on the Main tab. Under Manage, click Publishers to open the Available Publishers page.
2. Click Add Event Publisher to open the Create a New Event Publisher page.
3. Enter information as follows to create the new event publisher named loggerPublisher.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>loggerPublisher</td>
</tr>
<tr>
<td>Event Source</td>
<td>org.wso2.event.sensor.stream</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>logger</td>
</tr>
</tbody>
</table>

Create a New Event Publisher
4. Click **Add Event Publisher** to save the information.

**Step 4: View the simple event flow**

This step involves viewing the event flow you created and understanding how the different elements in it are connected. The event flow can be viewed as follows.

1. Log into the CEP Management Console if you are not already logged in.
2. Click the **Main** tab and then click **Flow** to open the **CEP Event Flow** page. The event flow you created is displayed as follows.

   ![Event Flow Diagram]

   This diagram indicates that `httpReceiver` forwards events to the `org.wso2.event.sensor.stream.1.0.0` stream. These events are then published by the `loggerPublisher`.

**Receive and log events**

**Step 5: Receive events via HTTP transport**

Navigate to `<CEP_HOME>/samples/cep/producers/http` and run the following command which sends events to the CEP via the HTTP transport.

```shell
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0001
```

This builds the HTTP client and sends the events in the `<CEP_HOME>/samples/cep/artifacts/0001/httpReceiver.txt` file to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below. These logs are published by the `publisher created in Step 3`. 
The logs of the JSON events received by the CEP server will be displayed in the CLI as shown in the example below.
Processing events with an execution plan

**Step 6: Add another event stream**

1. Log into the CEP Management Console and click on the Main tab. Under Manage, click Streams to open the Available Event Streams page.
2. Click Add Event Stream to open the Define New Event Stream page.
3. Enter information as follows to create the event stream named `org.wso2.event.sensor.filtered.stream`.
### Event Stream Details

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream Name</td>
<td>org.wso2.event.sensor.filtered.stream</td>
</tr>
<tr>
<td>Event Stream Version</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

### Stream Attributes

<table>
<thead>
<tr>
<th>Attribute Category</th>
<th>Attribute</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Date</td>
<td>timestamp</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>sensorName</td>
<td>string</td>
</tr>
<tr>
<td>Correlation Data</td>
<td>longitude</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>latitude</td>
<td>double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payload Data Attributes</th>
<th>Attribute Name</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sensorValue</td>
<td>double</td>
</tr>
</tbody>
</table>

---

Copyright © WSO2 Inc. 2005-2016
4. Click **Add Event Stream** to save the information.

**Step 7: Add an execution plan**

An Execution Plan can import one or more streams from the server for processing and push zero or more output streams back to the server. For more information, see **Analyzing Data**.

1. Log into the CEP Management Console and click on the **Main** tab. Under **Manage**, click **Execution Plans** to open the **Available Execution Plans** page.
2. Click **Add Execution Plan** to open the **Create a New Execution Plan** page.
3. Enter information as follows to create the new execution plan:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import Stream</td>
<td>org.wso2.event.sensor.stream:1.0.0</td>
</tr>
<tr>
<td>As</td>
<td>sensorStream</td>
</tr>
<tr>
<td>Value Of</td>
<td>filteredStream</td>
</tr>
<tr>
<td>StreamId</td>
<td>org.wso2.event.sensor.filtered.stream:1.0.0</td>
</tr>
</tbody>
</table>

4. Click **Import** and then click **Export**. The section for query expressions will be updated as shown below.
5. Add the following query expression.
This query includes the value `sensorValue > 100`. Therefore, when the execution plan forwards events from `org.wso2.event.sensor.stream:1.0.0` to `org.wso2.event.sensor.filtered.stream:1.0.0`, events in which the value for the `sensorValue` attribute is less than 100 will be dropped.

6. Click **Validate Query Expressions**. Once you get a message to confirm that the queries are valid, click **Add Execution Plan**.

**Publish events in dashboard**

**Step 8: Add a UI event publisher**

In this step, you will add another publisher named `uiPublisher` to publish events from the stream named `org.wso2.event.sensor.filtered.stream` to the Analytics Dashboard.

1. Log into the CEP Management Console and click on the **Main** tab. Under **Manage**, click **Publishers** to open the **Available Publishers** page.
2. Click **Add Event Publisher** to open the **Create a New Event Publisher** page.
3. Enter information as follows to create the new event publisher named **UIPublisher**.

![Create a New Event Publisher](image)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>uiPublisher</td>
</tr>
<tr>
<td>Event Source</td>
<td>org.wso2.event.sensor.filtered.stream:1.0.0</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>ui</td>
</tr>
<tr>
<td>Message Format</td>
<td>wso2event</td>
</tr>
</tbody>
</table>

4. Click **Add Event Publisher** to save the information.

**Step 9: Create a dashboard and a gadget**

WSO2 Analytics Dashboard will be used as the tool to analyse the output of the event flow you created in this guide. This step creates a dashboard and a gadget which analyses events from the **org.wso2.event.sensor.filtered.stream** stream published by the **uiPublisher** publisher.

1. Log into the CEP Management Console. In the **Main** tab, click **Analytics Dashboard**.
2. Log into the Analytics Dashboard with your username and password.
3. Click the menu icon and then click **Gadgets** to open the **Gadgets** page as demonstrated below.
4. Click **GENERATE GADGET**, and enter values in the **Generate a Gadget** wizard as follows.
Generate a Gadget

1. In the **Select Provider** field, select **Realtime Data Source**. Then click **Next**.
2. In the **Event Stream** field, select **org.wso2.event.sensor.filtered.stream:1.0.0**. Then click **Next**.
3. Configure a chart as follows.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadget Name</td>
<td>Sensor Value VS Timestamp</td>
</tr>
<tr>
<td>Select Chart Type</td>
<td>Line Chart</td>
</tr>
<tr>
<td>X-Axis</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>X type</td>
<td>time</td>
</tr>
<tr>
<td>Y-Axis</td>
<td>sensorValue</td>
</tr>
<tr>
<td>Y type</td>
<td>default</td>
</tr>
<tr>
<td>Color domain</td>
<td>sensorName</td>
</tr>
<tr>
<td>Max length</td>
<td>30</td>
</tr>
</tbody>
</table>

4. Click **Add to Store**, and then click **Go to Portal**. The **Dashboards** page appears again.
5. Click **CREATE DASHBOARD** to open the **Create a Dashboard** page. Configure a new dashboard as follows.

![Create Dashboard Button](image)

### Dashboards

- **No dashboards found.**

  To create a dashboard click [here](#)

---

1. Enter a name and a description for the new dashboard as follows, and click **Next**.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of your Dashboard</td>
<td>Sensor Statistics</td>
</tr>
<tr>
<td>Description</td>
<td>This dashboard indicates the sensor value at different times in a particular location.</td>
</tr>
</tbody>
</table>

2. Select the **Single Column** layout. A message appears to indicate that the dashboard is successfully created.

3. Click the **+** icon for gadgets. Then select and drag **Sensor Value VS Timestamp** gadget to the first column as demonstrated above.

---

**Step 10: Send Events to the HTTP Receiver via Curl Command**

This step sends events to the receiver named `httpReceiver` using a curl command. These events are processed by the event flow you created, and published in the CEP CLI by the publisher created in Step 3.

1. Issue the following curl command.
1. Issue the following command to send the sample event from the task in Exercise 1.

   ```
curl -X POST -d "{"event": { "metaData": { "timestamp": 1439468145264, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 96.5 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"
```

   The following log will appear in the CEP CLI.

   ![Log output]

   Note that the **Sensor Statistics** dashboard you created does not get updated. This is because the value for the **sensorValue** attribute is less than 100 in this event. Therefore, it gets dropped by the filter you created in the execution plan, and as a result, it is not forwarded to the `org.wso2.event.sensor.filtered.stream:1.0.0` stream.

2. Issue another command with a value greater than 100 for the **sensorValue** attribute as follows.

   ```
curl -X POST -d "{"event": { "metaData": { "timestamp": 1439467524120, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 156 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"
```

   The following log will appear in the CEP CLI.

   ![Log output]

   The event is forwarded to the `org.wso2.event.sensor.filtered.stream:1.0.0` stream since the value for the **sensorValue** attribute is greater than 100. Therefore, the **Sensor Statistics** dashboard will be updated as shown below.
3. Issue more curl commands as follows with several timestamps and sensor values.

```bash
curl -X POST -d "{ "event": { "metaData": { "timestamp": 1439467524120, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 156 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"

curl -X POST -d "{ "event": { "metaData": { "timestamp": 1439467890957, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 170 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"

curl -X POST -d "{ "event": { "metaData": { "timestamp": 1439467951518, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 131 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"

curl -X POST -d "{ "event": { "metaData": { "timestamp": 1439467992936, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 126 } } }" http://localhost:9763/endpoints/httpReceiver --header "Content-Type:application/json"
```
Since the value for the `sensorValue` attribute is greater than 100 in all these events, the dashboard will be updated as shown below. These events will also be logged in the CEP CLI.

Send sensor events with several timestamps and sensorValues. The dashboard will now get effected with the sent event since sensorValue is over 100 and the event gets sent to the dashboard. These events will get logged in the CEP CLI as well.

### Deploying execution plans using templates

**Step 11: Create and deploy a template**

In this step, the configurations of the WSO2 CEP artifacts that you previously created are added as templates via the Template Manager tool. This allows you to reuse the same artifacts for different scenarios where the sensor value differs.

Copy the following template, and save it with the `SensorStatistics.xml` file name in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory.

The `$sensorValue` attribute in this template is defined as a configurable parameter by using the `$` sign in the attribute name. Therefore, in each scenario you create from this template, you can specify a different sensor value based on which the events are to be filtered.

```xml
<?xml version="1.0"?>
<!--
~ Copyright (c) 2016, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
~ WSO2 Inc. licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.
~ You may obtain a copy of the License at
~
```
<domain name="SensorStatistics">
   <description>Domain for sensor data analysis</description>
   <scenarios>
      <scenario type="AnalyzeSensorStatistics">
         <description>Configure a sensor analytics scenario to display statistics for a given stream of your choice</description>
         <templates>
            <!--Note: These will be deployed in the order they appear here-->
            <!-- Input Event Stream-->
            <template type="eventstream">
               {
                  "streamId": "org.wso2.event.sensor.stream:1.0.0",
                  "name": "org.wso2.event.sensor.stream",
                  "version": "1.0.0",
                  "nickName": ",",
                  "description": "",
                  "metaData": [
                     {
                        "name": "timestamp",
                        "type": "LONG"
                     },
                     {
                        "name": "isPowerSaverEnabled",
                        "type": "BOOL"
                     },
                     {
                        "name": "sensorId",
                        "type": "INT"
                     },
                     {
                        "name": "sensorName",
                        "type": "STRING"
                     }
                  ],
                  "correlationData": [
                     {
                        "name": "longitude",
                        "type": "DOUBLE"
                     },
                     {
                        "name": "latitude",
                        "type": "DOUBLE"
                     }
                  ]
               }
            </template>
         </templates>
      </scenario>
   </scenarios>
</domain>
"type": "DOUBLE"
"name": "humidity",
"type": "FLOAT"
"name": "sensorValue",
"type": "DOUBLE"

</template>
</template>
<!-- Output Event Stream-->
<template type="eventstream">
{
"streamId": "org.wso2.event.sensor.filtered.stream:1.0.0",
"name": "org.wso2.event.sensor.filtered.stream",
"version": "1.0.0",
"nickName": "",
"description": "",
"metaData": [
{
"name": "timestamp",
"type": "LONG"
},
{
"name": "sensorName",
"type": "STRING"
}
],
"correlationData": [
{
"name": "longitude",
"type": "DOUBLE"
},
{
"name": "latitude",
"type": "DOUBLE"
}
],
"payloadData": [
{
"name": "sensorValue",
"type": "DOUBLE"
}
]

</template>
<!-- Realtime Execution Plan-->
<template type="realtime">
<![CDATA[
/* Enter a unique ExecutionPlan */
@Plan:name('ExecutionPlan')
/* Enter a unique description for ExecutionPlan */
-- @Plan:description('ExecutionPlan')
/* define streams/tables and write queries here ... */
@Import('org.wso2.event.sensor.stream:1.0.0')
define stream sensorStream (meta_timestamp long,
meta_isPowerSaverEnabled bool,
meta_sensorId int, meta_sensorName string,
correlation_longitude double, correlation_latitude double,
humidity float, sensorValue double);
@Export('org.wso2.event.sensor.filtered.stream:1.0.0')
define stream filteredStream (meta_timestamp long,
meta_sensorName string, correlation_longitude double,
correlation_latitude double, sensorValue double);
from sensorStream [sensorValue > $filteringVal]
select meta_timestamp, meta_sensorName,
correlation_longitude, correlation_latitude, sensorValue
insert into filteredStream
]]>
</template>
<!-- Event Receiver-->
<template type="eventreceiver">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<eventReceiver name="httpReceiver" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="http">
    <property name="basicAuthEnabled">true</property>
    <property name="transports">all</property>
  </from>
  <mapping customMapping="disable" type="json"/>
  <to streamName="org.wso2.event.sensor.stream" version="1.0.0"/>
</eventReceiver>]]>
</template>
<!-- Event Publisher-->
<template type="eventpublisher">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher name="loggerPublisher" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
  <from streamName="org.wso2.event.sensor.stream" version="1.0.0"/>
  <mapping customMapping="disable" type="text"/>
  <to eventAdapterType="logger"/>
</eventPublisher>]]>
<template type="eventpublisher">
  <![CDATA[
  <?xml version="1.0" encoding="UTF-8"?>
  <eventPublisher name="uiPublisher" statistics="disable"
  trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
  <from
  streamName="org.wso2.event.sensor.filtered.stream" version="1.0.0"/>
  <mapping customMapping="disable" type="wso2event"/>
  <to eventAdapterType="ui"/>
  </eventPublisher>
  ]]>}
</template>

<!-- Gadget line chart -->
<template type="gadget">
  <config>
    <properties>
      <property name="directoryName">$sensorType-line-chart</property>
      <property name="templateDirectory">lineChart</property>
    </properties>
    <artifacts>
      <artifact file="gadget.json">
        <![CDATA[
          {
            "id": "$sensorType-line-chart",
            "title": "$sensorType-line-chart",
            "type": "gadget",
            "thumbnail": "gadget/$sensorType-line-chart/thumbnail.png",
            "data": {
              "url": "gadget/$sensorType-line-chart/gadget.xml"
            }
          }
        ]]>
      </artifact>
      <artifact file="conf.json">
        <![CDATA[
          {
            "provider-conf": {
              "streamName": "org.wso2.event.sensor.filtered.stream:1.0.0",
              "provider-name": "realtime"
            },
            "chart-conf": {
              "x": "TIMESTAMP",
              "xType": "time",
              "y": "sensorValue",
              "yType": "default",
              "color": "sensorName",
              "maxLength": "30",
              "gadget-name": "$sensorType-line-chart",
            }}}}>
      </artifact>
    </artifacts>
  </config>
</template>
var getGadgetLocation = function (callback) {
  var gadgetLocation = "/portal/store/carbon.super/fs/gadget/$sensorType-line-chart";
  var PATH_SEPERATOR = "/";
  if (gadgetLocation.search("store") != -1) {
    wso2.gadgets.identity.getTenantDomain(function (tenantDomain) {
      var gadgetLocation = gadgetLocation.split(PATH_SEPERATOR);
      var modifiedPath = '';
      for (var i = 1; i < gadgetPath.length; i++) {
        if (i === 3) {
          modifiedPath = modifiedPath.concat(PATH_SEPERATOR, tenantDomain);
        } else {
          modifiedPath = modifiedPath.concat(PATH_SEPERATOR, gadgetPath[i])
        }
      }
      callback(modifiedPath);
    });
  } else {
    callback(gadgetLocation);
  }
};
"title": "Analytics $sensorType Dashboard",
"description": "This dashboard indicates the sensor value at different times in a particular location",
"permissions": {
  "viewers": [
    "Internal\sensor-statistics-viewer"
  ],
  "editors": [
    "Internal\sensor-statistics-editor"
  ],
  "owners": [
    "Internal\sensor-statistics-owner"
  ]
},
"pages": [
  {
    "id": "landing",
    "title": "Home",
    "layout": {
      "content": {
        "loggedIn": {
          "blocks": [
            {
              "id": "90dfe9100dc10dca1ae562e7f7451a4a",
              "x": 0,
              "y": 0,
              "width": 12,
              "height": 3,
              "banner": false
            }
          ]
        }
      },
      "fluidLayout": false
    },
    "isanon": false,
    "content": {
      "default": {
        "90dfe9100dc10dca1ae562e7f7451a4a": [
          {
            "id": "$sensorType-line-chart-0",
            "content": {
              "id": "$sensorType-line-chart",
              "title": "$sensorType-line-chart",
              "type": "gadget",
              "thumbnail": "fs:\gadget\$sensorType-line-chart\thumbnail.png",
              "data": {
                "url": "fs:\gadget\$sensorType-line-chart\gadget.xml"
              },
              "styles": {
                "title": "$sensorType-line-chart",
                "data": {
                  "url": "fs:\gadget\$sensorType-line-chart\gadget.xml"
                }
              }
            }
          }
        ]
      }
    }
  }
]
"borders": true,
"options": {
  "windowSize": {
    "type": "STRING",
    "title": "Window Size",
    "value": "10",
    "options": [
      
    ],
    "required": false
  }
},
"locale_titles": {
  
}
},
"anon": {
  
}
],
"menu": [
{
  "id": "landing",
  "isanon": false,
  "ishidden": false,
  "title": "Home",
  "subordinates": [

  ]
}
],
"hideAllMenuItems": false,
"identityServerUrl": "",
"accessTokenUrl": "",
"apiKey": "",
"apiSecret": "",
"theme": "Default Theme",
"shareDashboard": false,
"isUserCustom": false,
"isEditorEnable": true,
"banner": {
  "globalBannerExists": false,
  "customBannerExists": false
},
"landing": "landing",
"isanon": false
]]>
</content>
</config>
</template>
</templates>
<parameters>
  <parameter name="filteringVal" type="string">
    <displayName>Filtering Value</displayName>
    <description>Only the sensor values below filtering value will be dropped</description>
    <defaultValue>100</defaultValue>
  </parameter>
  <parameter name="sensorType" type="string">
    <displayName>Sensor Type Name</displayName>
    <description>The name of the sensor type</description>
    <defaultValue>temperature</defaultValue>
  </parameter>
</parameters>
Step 12: Configure a template

Before you carry out this step

1. Copy Sensor Value VS Timestamp gadget which resides in {PRODUCT_HOME}/repository/deployment/server/jaggeryapps/portal/store/carbon.super/fs/gadget/ and copy the folder to wso2cep-4.2.0/repository/conf/template-manager/gadget-templates and rename it as lineChart
2. delete the following artifacts that you have already configured in steps 1 - 9.
   - org.wso2.event.sensor.stream event stream
   - org.wso2.event.sensor.filtered.stream event stream
   - httpReceiver event receiver
   - ExecutionPlan execution plan
   - loggerPublisher event publisher
   - uiPublisher event publisher
   - Sensor Value VS Timestamp gadget
   - Sensor Statistics dashboard

This step involves adding execution plan and stream configurations using the template you created and added in the previous step.

1. If the CEP server was running when you created and deployed the template, restart the CEP server.
2. Log into the CEP Management Console. Click the Main tab and then click Template Manager. Create a new scenario as demonstrated below.
1. Click on SensorStatistics to open the Deployed Scenarios page. Then click Create New Scenario to open the Edit Scenario page.
2. Enter information as shown in the table below and click Add scenario.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario Type</td>
<td>AnalyzeSensorStatistics</td>
</tr>
<tr>
<td>Scenario Name</td>
<td>FilterSensorValues</td>
</tr>
<tr>
<td>Description</td>
<td>Filter events with a sensor value greater than 120</td>
</tr>
<tr>
<td>Sensor Value</td>
<td>120</td>
</tr>
</tbody>
</table>

A message appears to inform you that the scenario is successfully created. Close the message. The scenario you configured is displayed in the Deployed Scenarios page.

Step 13: View the elements of the event flow

Log into the CEP Management Console and click the Main tab. Then click Flow to open the CEP Event Flow page. The complete event flow you created in this guide is displayed as follows.

This event flow is displayed when you create the artifacts manually, as well as when you deploy them in a template and then create a scenario.
The following is a summary of this guide which describes each element in the event flow.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>httpReceiver</td>
<td>Event Receiver</td>
<td>Receives CEP events in multiple formats and converts them all into the WSO2 Event format before forwarding them to the org.wso2.event.sensor.stream:1.0.0 event stream.</td>
</tr>
<tr>
<td>org.wso2.event.sensor.stream:1.0.0</td>
<td>Event Stream</td>
<td>Defines the attributes on which selection of events to be processed by the event flow is based.</td>
</tr>
<tr>
<td>loggerPublisher</td>
<td>Event Publisher</td>
<td>Logs events from the org.wso2.event.sensor.stream:1.0.0 event stream in the CEP CLI.</td>
</tr>
<tr>
<td>ExecutionPlan</td>
<td>Execution Plan</td>
<td>Applies a filter criteria to events in the org.wso2.event.sensor.stream:1.0.0 event stream and forwards filtered events to org.wso2.event.sensor.filtered.stream:1.0.0 event stream.</td>
</tr>
<tr>
<td>SensorStatistics-FilterSensorValues</td>
<td>Execution Plan</td>
<td>This is an execution plan created from a template.</td>
</tr>
<tr>
<td>org.wso2.event.sensor.filtered.stream:1.0.0</td>
<td>Event Stream</td>
<td>Imports attributes from the org.wso2.event.sensor.stream:1.0.0 event stream and receives events filtered from that event stream by the execution plan.</td>
</tr>
<tr>
<td>uiPublisher</td>
<td>Event Publisher</td>
<td>Publishes events from the org.wso2.event.sensor.filtered.stream:1.0.0 event stream in the Analytics Dashboard.</td>
</tr>
</tbody>
</table>

**Downloading the Product**

Follow the instructions below to download the binary distribution of the Complex Event Processor.

The binary distribution contains the binary files for both MS Windows, and Linux-based operating systems. It is recommended for most users. You can also download, and build the source code.

2. Click the **Download** button in the upper right-hand corner of the page to download the latest version. To download an older version, click the **Previous Releases** link and then select the version that you want.
3. Enter the required details in the form, and click **Download**.

Next, go to Installation Prerequisites for instructions on installing the necessary supporting applications.

## Installation Prerequisites

Prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. Verify that the computer has the supported operating system and development platforms before starting the installation.

### System requirements

| Memory | ~ 2 GB minimum  
~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently. |
| Disk | ~ 1 GB minimum, excluding space allocated for log files and databases. |

### Environment compatibility

| Operating Systems / Databases | All WSO2 Carbon-based products are Java applications that can be run on **any platform that is Oracle JDK 1.8.* compliant**. Also, **we do not recommend OpenJDK** as we do not support it or test our products with it.  
All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see Working with Databases.  
If you have difficulty in setting up any WSO2 product in a specific platform or database, please contact us.  
It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.  
WSO2 CEP does not support SunOS 5.10 and IBM AIX. For environments that WSO2 products are tested with, see Compatibility of WSO2 Products. |

### Required applications

The following applications are required for running the Complex Event Processor or for building from the source code. Mandatory installs are marked with *.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Version</th>
</tr>
</thead>
</table>

Copyright © WSO2 Inc. 2005-2016
**Oracle Java SE Development Kit (JDK)**
- To launch the product as each product is a Java application.
- To **build the product from the source distribution** (both JDK and Apache Maven are required).
- To run Apache Ant.

**Git**
- Required to check out the source from the Git repository.

**Apache Maven**
- To **build the product from the source distribution** (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do **not** need to install Maven.

**Web Browser**
- To access each product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console.
- **Firefox**
- **Firefox 39.0**
- **Firefox 40.0.3**
- **Chrome**
- **Chrome 43.0.2357.125**
- **Chrome 45.0.2454.93**
- **Safari 8.0.7**

**Apache Storm**
- To deploy WSO2 CEP in a distributed mode.

If you want to run custom extensions in the CEP distributed deployment with Apache Storm, edit the dependencies in the `<CEP_HOME>/samples/cep/utils/storm-dependencies-jar/pom.xml` file accordingly, and build only this `pom.xml` file.

---

You are now ready to install. Click one of the following links for instructions:

- Installing on Linux
- Installing on Windows
- Installing as a Windows Service
Installing the Product

Installing WSO2 is very fast and easy. Before you begin, be sure you have met the installation prerequisites, and then follow the installation instructions for your platform.

- Installing on Linux
- Installing on Windows
- Installing as a Linux Service
- Installing as a Windows Service

Installing on Linux

Follow the instructions below to install CEP on Linux.

**Installing the required applications**

1. Establish an SSH connection to the Linux machine or log in on the text Linux console.
2. Be sure your system meets the Installation Prerequisites. Java Development Kit (JDK) is essential to run the product.

**Installing the CEP**

1. Download the latest version of the CEP as described in Downloading the Product.
2. Extract the archive file to a dedicated directory for the CEP, which will hereafter be referred to as `<CEP_HOME>`.

**Setting up JAVA_HOME**

You must set your JAVA_HOME environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the operating system.

1. In your home directory, open the BASHRC file in your favorite Linux text editor, such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.7 in your system, add the following two lines at the bottom of the file, replacing `/u sr/java/jdk1.7` with the actual directory where the JDK is installed.

   ```
   export JAVA_HOME=/usr/java/jdk1.7
   export PATH=${JAVA_HOME}/bin:${PATH}
   ```

3. Save the file.

   If you do not know how to work with text editors in a Linux SSH session, run the following command:
   ```
   cat >> .bashrc
   Paste the string from the clipboard and press "Ctrl+D."
   ```

4. To verify that the JAVA_HOME variable is set correctly, execute the following command:
   ```
   echo $JAVA_HOME
   ```
5. The system returns the JDK installation path.

**Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script**: Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- **Set the properties from an external registry**: If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the secure vault.

You are now ready to run the product.

**Installing on Windows**

Follow the instructions below to install the product on Windows.

**Installing the required applications**

1. Be sure your system meets the Installation Prerequisites. Java Development Kit (JDK) is essential to run the product.

**Installing the CEP**

1. Download the latest version of the product as described in Obtaining the Product.
2. Extract the archive file to a dedicated directory for the product, which will hereafter be referred to as `<CEP_HOME>`.

**Setting up JAVA_HOME**

You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:/Program Files/Java`, such as `C:/Program Files/Java/jdk1.7`. If you have multiple versions installed, choose the latest one, which you can find by sorting by date.

Environment variables are global system variables accessible by all the processes running under the operating system. You can define an environment variable as a system variable, which applies to all users, or as a user variable, which applies only to the user who is currently logged in.

You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, set it at the command prompt.

**Setting up JAVA_HOME using the system properties**

1. Right-click the **My Computer** icon on the desktop and choose **Properties**.
2. In the System Properties window, click the **Advanced** tab, and then click the **Environment Variables** button.

3. Click the New button under **System variables** (for all users) or under **User variables** (just for the user who is currently logged in).

4. Enter the following information:
• In the **Variable name** field, enter: JAVA_HOME
• In the **Variable value** field, enter the installation path of the Java Development Kit, such as: c:/Program Files/Java/jdk1.7

The JAVA_HOME variable is now set and will apply to any subsequent command prompt windows you open. If you have existing command prompt windows running, you must close and reopen them for the JAVA_HOME variable to take effect, or manually set the JAVA_HOME variable in those command prompt windows as described in the next section. To verify that the JAVA_HOME variable is set correctly, open a command window (from the **Start** menu, click **Run**, and then type **CMD** and click **Enter**) and execute the following command:

```
set JAVA_HOME
```

The system returns the JDK installation path. You are now ready to **run the product**.

**Setting JAVA_HOME temporarily using the Windows command prompt (CMD)**

You can temporarily set the JAVA_HOME environment variable within a Windows command prompt window (CMD). This is useful when you have an existing command prompt window running and you do not want to restart it.

1. In the command prompt window, enter the following command where `<JDK_INSTALLATION_PATH>` is the JDK installation directory and press **Enter**.

   ```
   set JAVA_HOME=<JDK_INSTALLATION_PATH>
   ```

   For example: `set JAVA_HOME=c:/Program Files/java/jdk1.7`

   The JAVA_HOME variable is now set for the current CMD session only.

2. To verify that the JAVA_HOME variable is set correctly, execute the following command:

   ```
   set JAVA_HOME
   ```

3. The system returns the JDK installation path.

**Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

• **Set the properties from a script**: Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.

• **Set the properties from an external registry**: If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the **secure vault**.

You are now ready to **run the product**.

**Installing as a Linux Service**

WSO2 Carbon and any Carbon-based product can be run as a Linux service as described in the following sections:

• **Prerequisites**
• **Setting up CARBON_HOME**
• **Running the product as a Linux service**

**Prerequisites**
Install JDK and set up the `JAVA_HOME` environment variable. For more information, see Installation Prerequisites.

**Setting up CARBON_HOME**

Extract the WSO2 product that you want to run as a Linux service and set the environment variable `CARBON_HOME` to the extracted product directory location.

**Running the product as a Linux service**

1. To run the product as a service, create a startup script and add it to the boot sequence. The basic structure of the startup script has three parts (i.e., start, stop and restart) as follows:

```bash
#!/bin/bash

case "\$1" in
  start)
    echo "Starting Service"
  ;;
  stop)
    echo "Stopping Service"
  ;;
  restart)
    echo "Restarting Service"
  ;;
  *)
    echo "$Usage: \$0 {start|stop|restart}"
  esac

echo 1
exit 1
```

For example, given below is a startup script:
#! /bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='/opt/WSO2/wso2cep-4.0.0/bin/wso2server.sh start > /dev/null &'
restartcmd='/opt/WSO2/wso2cep-4.0.0/bin/wso2server.sh restart > /dev/null &'
stopcmd='/opt/WSO2/wso2cep-4.0.0/bin/wso2server.sh stop > /dev/null &'

case "$1" in
  start)
    echo "Starting WSO2 Complex Event Processor ..."
    su -c "${startcmd}" user1
    ;;
  restart)
    echo "Re-starting WSO2 Complex Event Processor ...
    su -c "${restartcmd}" user1
    ;;
  stop)
    echo "Stopping WSO2 Complex Event Processor ...
    su -c "${stopcmd}" user1
    ;;
  *)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
  esac

In the above script, the server is started as a user by the name user1 rather than the root user. For example, su -c "${startcmd}" user1

2. Add the script to /etc/init.d/ directory.

If you want to keep the scripts in a location other than /etc/init.d/ folder, you can add a symbolic link to the script in /etc/init.d/ and keep the actual script in a separate location. Say your script name is cepserver and it is in /opt/WSO2/ folder, then the commands for adding a link to /etc/init.d/ is as follows:

- Make executable: sudo chmod a+x /opt/WSO2/cepserver
- Add a link to /etc/init.d/: sudo ln -snf /opt/WSO2/cepserver /etc/init.d/cepserver

3. Install the startup script to respective runlevels using the command update-rc.d. For example, give the following command for the sample script shown in step1:

  sudo update-rc.d cepserver defaults

The defaults option in the above command makes the service to start in runlevels 2,3,4 and 5 and to stop in runlevels 0,1 and 6.
A runlevel is a mode of operation in Linux (or any Unix-style operating system). There are several runlevels in a Linux server and each of these runlevels is represented by a single digit integer. Each runlevel designates a different system configuration and allows access to a different combination of processes.

4. You can now start, stop and restart the server using `service <service name> {start|stop|restart}` command. You will be prompted for the password of the user (or root) who was used to start the service.

### Installing as a Windows Service

WSO2 Carbon and any Carbon-based product can be run as a Windows service as described in the following sections:

- **Prerequisites**
- Setting up the YAJSW wrapper configuration file
- Setting up CARBON_HOME
- Running the product in console mode
- Working with the WSO2CARBON service

**Prerequisites**

- Install JDK and set up the JAVA_HOME environment variable. For more information, see Installation Prerequisites.
- Download and install a service wrapper library to use for running your WSO2 product as a Windows service. WSO2 recommends Yet Another Java Service Wrapper (YAJSW) version 11.03, and several WSO2 products provide a default wrapper.conf file in their `<PRODUCT_HOME>/bin/yajsw/` directory. The instructions below describe how to set up this file.

**Setting up the YAJSW wrapper configuration file**

The configuration file used for wrapping Java Applications by YAJSW is `wrapper.conf`, which is located in the `YAJSW_HOME/conf/` directory and in the `<PRODUCT_HOME>/bin/yajsw/` directory of many WSO2 products. Following is the minimal `wrapper.conf` configuration for running a WSO2 product as a Windows service. Open your `wrapper.conf` file, set its properties as follows, and save it in `<YAJSW_HOME>/conf/` directory.

If you want to set additional properties from an external registry at runtime, store sensitive information like usernames and passwords for connecting to the registry in a properties file and secure it with secure vault.

```
#********************************************************************
# working directory
#********************************************************************
wrapper.working.dir=${carbon_home}\%
# Java Main class.
# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
#********************************************************************
# tmp folder
# yajsw creates temporary files named in_.. out_.. err_.. jna..
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
```

---

**Minimal wrapper.conf configuration**

#********************************************************************
# working directory
#********************************************************************
wrapper.working.dir=${carbon_home}\%
# Java Main class.
# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
#********************************************************************
# tmp folder
# yajsw creates temporary files named in_.. out_.. err_.. jna..
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
# wrapper.tmp.path = ${jna_tmpdir}
#
# Application main class or native executable
# One of the following properties MUST be defined
#
# Java Application main class
wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap
#
# Log Level for console output.  (See docs for log levels)
wrapper.console.loglevel=INFO
#
# Log file to use for wrapper output logging.
wrapper.logfile=${wrapper_home}/log/wrapper.log
#
# Format of output for the log file.  (See docs for formats)
#wrapper.logfile.format=LPTM
#
# Log Level for log file output.  (See docs for log levels)
#wrapper.logfile.loglevel=INFO
#
# Maximum size that the log file will be allowed to grow to before
# the log is rolled.  Size is specified in bytes.  The default value
# of 0, disables log rolling by size.  May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix.  For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLNUM it will be
# automatically added as suffix of the file name
wrapper.logfile.maxsize=10m
#
# Maximum number of rolled log files which will be allowed before old
# files are deleted.  The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
#
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
#
# Wrapper Windows Service and Posix Daemon Properties
#
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
#
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
#
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
#
# Wrapper System Tray Properties
#
# enable system tray
wrapper.tray = true
#
# TCP/IP port.  If none is defined multicast discovery is used to find the
# port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
#
# Exit Code Properties
#
# Restart on non zero exit code
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
#
# Trigger actions on console output
#
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts/trayMessage.gv
wrapper.filter.script.0.args=Exception
#
# genConfig: further Properties generated by genConfig
#
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 =
-Xbootclasspath/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 =
-XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 =
-Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=/
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\bin\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\lib\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\lib\tomcat
wrapper.java.additional.15 =
-Djava.util.logging.config.file=${carbon_home}\repository\conf\log4j.properties
wrapper.java.additional.16 =
-Dcarbon.config.dir.path=${carbon_home}\repository\conf
wrapper.java.additional.17 =
-Dcarbon.logs.path=${carbon_home}\repository\logs
wrapper.java.additional.18 =
-Dcomponents.repo=${carbon_home}\repository\components\plugins
wrapper.java.additional.19 =
-Dconf.location=${carbon_home}\repository\conf
wrapper.java.additional.20 =
-Dcom.atomikos.icatch.file=${carbon_home}\lib\transactions.properties
wrapper.java.additional.21 =
-Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.22 =
-Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
wrapper.java.additional.23 =
Setting up CARBON_HOME

Extract the Carbon-based product that you want to run as a Windows service, and then set the Windows environment variable CARBON_HOME to the extracted product directory location. For example, if you want to run ESB 4.5.0 as a Windows service, you would set CARBON_HOME to the extracted `wso2esb-4.5.0` directory.

Running the product in console mode

You will now verify that YAJSW is configured correctly for running the Carbon-based product as a Windows service.

1. Open a Windows command prompt and go to the `<YAJSW_HOME>/bat/` directory. For example:

```
cd C:\Documents and Settings\yajsw_home\bat
```

2. Start the wrapper in console mode using the following command:

```
runConsole.bat
```

For example:

```
C:\Documents and Settings\yajsw_home\bat>runConsole.bat
```

If the configurations are set properly for YAJSW, you will see console output similar to the following and can now access the WSO2 management console from your web browser via https://localhost:9443/carbon.
Working with the WSO2CARBON service

To install the Carbon-based product as a Windows service, execute the following command in the `<YAJSW_HOME>/bat` directory:

```
installService.bat
```

The console will display a message confirming that the WSO2CARBON service was installed.

```
C:\Documents and Settings\yajsw_home\bat\installService.bat
C:\Documents and Settings\yajsw_home\bat\cd C:\Documents and Settings\yajsw_home\bat\n
C:\Documents and Settings\yajsw_home\bat\call setenv.bat
"java" -Xmx30m -Djna.tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -i "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0.06
INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\vfs_cache" as temporary files store.
platform null
**---------------------- INSTALLING "WSO2CARBON" ----------------------**
Service "WSO2CARBON" installed
Press any key to continue . . .
```

To start the service, execute the following command in the same console window:

```
startService.bat
```

The console will display a message confirming that the WSO2CARBON service was started.
To stop the service, execute the following command in the same console window:

```
stopService.bat
```

The console will display a message confirming that the WSO2CARBON service has stopped.

To uninstall the service, execute the following command in the same console window:

```
uninstallService.bat
```

The console will display a message confirming that the WSO2CARBON service was removed.
Building from Source

WSO2 invites you to contribute by downloading the source code from the GitHub source control system, building the product and making changes, and then committing your changes back to the source repository. The following sections describe this process:

- **Downloading the source**
- **Editing the source code**
- **Building the product**
- **Committing your changes**

Building from source is optional. Users who do not want to make changes to the source code can simply download the binary distribution of the product and install it.

**Downloading the source**

WSO2 products are built on top of WSO2 Carbon Kernel, which contains the Kernel libraries used by all products. When there are changes in the Carbon Kernel, they are bundled and released in a new WSO2 Carbon version (for example, WSO2 Carbon 4.3.0). You can download the complete WSO2 Kernel release using the following repository: https://github.com/wso2/carbon4-kernel, which is recommended if you intend to modify the source.

After downloading the source of the Carbon Kernel, execute the following command to download the source of the product:

```
git clone https://github.com/wso2/product-cep
```

After the source code is downloaded, you can start editing. However, it is recommended to run a build prior to changing the source code to ensure that the download is complete.

**Editing the source code**

Now that you have downloaded the source code for the Carbon project from GitHub, you can prepare your development environment and do the required changes to the code.

1. To edit the source code in your IDE, set up your development environment by running one of the following commands:
### Building the product

Ensure that the following prerequisites are in place before you build:

1. Make sure the build server has an active Internet connection to download dependencies while building.
2. Install Maven and JDK. For compatible versions, see [Installation Prerequisites](#).
3. Set the environment variable `MAVEN_OPTS=-Xms1024m -Xmx4096m -XX:MaxPermSize=1024m` to avoid the Maven `OutOfMemoryError`.

Use the following Maven commands to build your product:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvn clean install</td>
<td>The binary and source distributions.</td>
</tr>
<tr>
<td>mvn clean install -Dmaven.test.skip=true</td>
<td>The binary and source distributions, without running any of the unit tests.</td>
</tr>
<tr>
<td>mvn clean install -Dmaven.test.skip=true -o</td>
<td>The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you have already built the source at least once.</td>
</tr>
</tbody>
</table>

### Committing your changes

You can contribute to WSO2 products by committing your changes to GitHub. Whether you are a committer or a non-committer, you can contribute with your code.

### Upgrading from WSO2 CEP 4.1.0

This section provides information on how you can upgrade from WSO2 CEP 4.1.0 to WSO2 CEP 4.2.0. For more information on release versions, see the [Release Matrix](#).

You cannot roll back the upgrade process. However, it is possible to restore a backup of the previous database so that you can restart the upgrade progress.

- Preparing to upgrade
- Migrating configurations
- Migrating artifacts
- Testing the upgrade

### Preparing to upgrade

The following prerequisites should be completed before upgrading.

- Make a backup of the CEP 4.1.0 database and copy the `<CEP_HOME_4.1.0>` directory in order to backup
the product configurations.


### Migrating configurations

Follow the procedure below to migrate configurations from CEP 4.1.0 to CEP 4.2.0.

#### Configuration should not be copied directly from CEP 4.1.0 to CEP 4.2.0 because some of the configuration files were updated with new properties between the two releases.

1. Make the required changes in the datasource files in the `<CEP_HOME>/repository/conf/datasources` directory to make sure that they are configured as required. For more information, see [Datasources](#).
2. Check the configurations in the following files in `<CEP_HOME>/repository/conf/data-bridge` directory for CEP 4.1.0, and apply the same changes in CEP 4.2.0.
   - `data-agent-config.xml`
   - `data-bridge-config.xml`
3. Check the configurations in the following files in CEP 4.1.0, and apply same configurations in CEP 4.2.0.

<table>
<thead>
<tr>
<th>Configuration File</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis2.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/axis2</code></td>
</tr>
<tr>
<td>axis2_client.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/axis2</code></td>
</tr>
<tr>
<td>carbon.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf</code></td>
</tr>
<tr>
<td>master-datasources.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/datasources</code></td>
</tr>
<tr>
<td>identity.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf</code></td>
</tr>
<tr>
<td>log4j.properties</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf</code></td>
</tr>
<tr>
<td>registry.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf</code></td>
</tr>
<tr>
<td>catalina-server.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/tomcat</code></td>
</tr>
<tr>
<td>web.xml</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/tomcat</code></td>
</tr>
</tbody>
</table>

4. If the Cipher tool was configured for CEP 4.1.0, check the configurations in the following files and apply the same changes in CEP 4.2.0.

<table>
<thead>
<tr>
<th>Configuration File</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipher-text.properties</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/security</code></td>
</tr>
<tr>
<td>cipher-tool.properties</td>
<td><code>&lt;CEP_HOME&gt;/repository/conf/security</code></td>
</tr>
</tbody>
</table>

5. Check the configurations in the other files located in the `<CEP 4.1.0_HOME>/repository/conf` directory and apply the same changes to CEP 4.2.0.
6. If there are any third party libraries used with CEP 4.1.0 that you want to migrate, copy the following directories as applicable from CEP 4.1.0 to CEP 4.2.0.
   - If you have used JMS libraries, JDBC libraries etc, copy `<CEP_HOME>/repository/components/lib`.
   - If you have used OSGi bundles such as SVN kit etc, copy `<CEP_HOME>/repository/components/dropins`
7. Start the CEP 4.2.0 server. For more information, see [Running the Product](#).
Migrating artifacts

All artifacts can be migrated from CEP 4.1.0 to CEP 4.2.0 via Composite Application Archive (CAR) files. For detailed instructions package artifacts as C-App archives, see Packaging Artifacts as C-App Archive.

You should manually deploy Composite Application Archive (CAR) files that you have in CEP 4.1.0 to CEP 4.2.0.

To migrate deployment artifacts:

- Copy the `<CEP_HOME>/repository/deployment/server` directory from CEP 4.1.0 to CEP 4.2.0.
- If multi-tenancy is used, copy the `<CEP_HOME>/repository/tenants` directory from CEP 4.1.0 to CEP 4.2.0.

Updating execution plans

WSO2 CEP 4.2.0 uses Siddhi 3.1.2 whereas CEP 4.1.0 uses Siddhi 3.0.5.

All Siddhi versions older than Siddhi 3.1.2 add a length window of size 1 for the initial stream of join queries. From Siddhi 3.1.2 onwards, a length window of size 0 is added. Therefore, once you migrate execution plans from CEP 4.1.0 to CEP 4.2.0, you need to manually add a length window of size 1 for the join queries as shown below (i.e. if you were relying on a length window of size 1 to be added by Siddhi in CEP 4.1.0).

e.g., The following query is included in an execution plan in CEP 4.1.0.

```sql
from TempStream[temp > 30.0]#window.time(1 min) as T join
RegulatorStream[ison == false] as R
  on T.roomNo == R.roomNo
select T.roomNo, R.deviceID, 'start' as action
insert into RegulatorActionStream;
```

When this execution plan is migrated to CEP 4.2.0, `#window.length(1)` property should be manually added to it as shown below.

```sql
from TempStream[temp > 30.0]#window.time(1 min) as T join
RegulatorStream[ison == false]#window.length(1) as R
  on T.roomNo == R.roomNo
select T.roomNo, R.deviceID, 'start' as action
insert into RegulatorActionStream;
```

Migrating dashboard elements

This section covers the additional steps are required to migrate dashboard elements (i.e. gadgets and dashboards).

The Analytics Dashboard functionality in CEP 4.2.0 is powered by WSO2 Dashboard Server 2.1.0 whereas the same functionality in CEP 4.1.0 and earlier versions is powered by WSO2 Gadget Server 1.4.0. This section explains how to migrate the gadgets and dashboards from CEP 4.1.0 to 4.2.0.

There are two methods that can be used to migrate gadgets and dashboards between the two CEP versions.
• Deploying the gadgets and dashboards in a CAR file
• Migrating the dashboards and gadgets pointing your product home directory.

In both methods, the following two files are used to execute the migration.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds-migration-1.0.x-2.0.x.bat or ds-migration-1.0.x-2.0.x.sh</td>
<td>&lt;CEP_HOME&gt;/migration/dashboards/migration-1.0.x-2.0.x</td>
</tr>
<tr>
<td>migration.xml</td>
<td>&lt;CEP_HOME&gt;/migration/dashboards/migration-1.0.x-2.0.x</td>
</tr>
</tbody>
</table>

Click on the relevant tab depending on the method you want to follow.

• Using CAR files
• Pointing to the product home directory

Follow the procedure below to migrate gadgets and dashboards using CAR files.

This method is appropriate when the dashboard elements to be migrated are deployed via a CAR file in CEP 4.1.0.

1. Open the <CEP_HOME>/migration/dashboard/migration1.0x-2.0.x/migration.xml file, and comment out the Portal File Migration section as shown in the example below.

```xml
<!-- Portal File Migration-->
<!-- Uncomment the below part to enable Portal migration -->
<!-- DSMmigration-->

<!-- Mode - Specifiy the migration type CAR or Portal -->
<!-- Mode>Portal</Mode-->

<!-- SourceDir - Specify the older PRODUCT_HOME directory.-->
<!-- <SourceDir></SourceDir> -->

<!-- DestinationDir - Specify the destination directory for migrated Store and Dashboards -->
<!-- <DestinationDir></DestinationDir>-->

<!-- TrustStoreLocation - Specify the trustedstore location. EX:-}
PRODUCT_HOME/repository/resources/security/wso2carbon.jks --
<!--
<TrustStoreLocation>YOUR_PRODUCT_HOME/repository/resources/security/wso2carbon.jks</TrustStoreLocation> -->

<!-- TrustStorePassword - Specify the trustedstore password. EX:- wso2carbon -->
<!-- <TrustStorePassword>wso2carbon</TrustStorePassword> -->

<!-- SourceURL - Specify the source url - Older version of your product. EX:-https://hostname:port -->
<!-- <SourceURL>https://localhost:9443</SourceURL> -->

<!-- SourceUsername - Specify the carbon.super user's username EX:-admin -->
<!-- <SourceUsername>admin</SourceUsername> -->

<!-- SourcePassword - Specify the carbon.super user's password EX:-admin -->
<!-- <SourcePassword>admin</SourcePassword> -->

<!-- DestinationURL - Specify the destination url - Newer version of your product. EX:-https://hostname:port -->

<!-- DestinationUsername - Specify the carbon.super user's username EX:-admin -->
<!-- <DestinationUsername>admin</DestinationUsername> -->

<!-- DestinationPassword - Specify the carbon.super user's password EX:-admin -->
<!-- <DestinationPassword>admin</DestinationPassword> -->

<!-- TenantDomains - Specify the tenant domain - This will migrate all the dashboards within these tenants. If you do not put
2. Run the following command to run the migration script.
   **For Windows**: `\<CEP-4.1.0_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.bat --run`
   
   **For Linux**: `\<CEP-4.1.0_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.sh`

3. Pack the required gadgets and dashboards in WSO2 CEP 4.1.0 to a CAR file and deploy it in WSO2 CEP 4.2.0. For more information, see [Packaging Artifacts as C-App Archive](#).

4. Perform the following steps for each gadget.

   ```xml
   <!-- CAR File Migration-->
   <!-- Uncomment the below part to enable CAR file migration -->
   <Mode>CAR</Mode>
   <!-- SourceDir - Specify the older PRODUCT_HOME directory.-->
   <SourceDir>CAR_FILE_SOURCE_DIRECTORY</SourceDir>
   <!-- DestinationDir - Specify the destination directory for migrated Store and Dashboards -->
   <DestinationDir>CAR_FILE_DESTINATION_DIRECTORY</DestinationDir>
   </DSMigration>
   ```

   In this approach the product home directory is defined as the source file. It migrates gadgets, layouts and widgets into the newer CEP version and copies them into the destination location specified by you.

   Follow the procedure below to migrate gadgets and dashboards using the Portal.

   1. Open the `<CEP_HOME>/migration/dashboard/migration1.0x-2.0.x/migration.xml` file, and comment out the **CAR File Migration** section as shown in the example below.

   ```xml
   <!---- CAR File Migration-->
   <!-- Uncomment the below part to enable CAR file migration -->
   <!--DSMigration>
   <!-- Mode - Specify the migration type CAR or Portal -->
   <Mode>CAR</Mode>
   <!-- SourceDir - Specify the older PRODUCT_HOME directory.-->
   <SourceDir>CAR_FILE_SOURCE_DIRECTORY</SourceDir>
   <!-- DestinationDir - Specify the destination directory for migrated Store and Dashboards -->
   <DestinationDir>CAR_FILE_DESTINATION_DIRECTORY</DestinationDir>
   </DSMigration-->
   ```

   2. Configure the parameters in the **Portal File Migration** section as explained in the table below.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>The migration mode. This should be CAR (if you are migrating the dashboard elements in the CAR file), or Portal (if you are migrating the dashboard elements in via the portal).</td>
<td></td>
</tr>
<tr>
<td>SourceDir</td>
<td>The source of the dashboard elements to be migrated. The <code>&lt;CEP_HOME&gt;</code> should be specified as the source directory.</td>
<td></td>
</tr>
<tr>
<td>DestinationDir</td>
<td>The destination directory for the migrated Store.</td>
<td></td>
</tr>
<tr>
<td>TrustStoreLocation</td>
<td>This parameter specifies the trusted store location.</td>
<td><code>&lt;CEP_HOME&gt;/repository/resources/security/wso2carbon.jks</code></td>
</tr>
<tr>
<td>TrustStorePassword</td>
<td>The password of the trust store.ashboard</td>
<td>wso2carbon</td>
</tr>
<tr>
<td>SourceURL</td>
<td>The source URL. This should be the URL of the CEP 4.1.0 instance from which you are migrating the dashboard elements.</td>
<td><code>https://hostname:port</code></td>
</tr>
<tr>
<td>SourceUsername</td>
<td>Specify the username of the carbon super user.</td>
<td><code>admin</code></td>
</tr>
<tr>
<td>SourcePassword</td>
<td>Specify the password of the carbon super user.</td>
<td><code>admin</code></td>
</tr>
<tr>
<td>DestinationURL</td>
<td>The destination URL. This should be the URL of the CEP 4.2.0 to which you are migrating the dashboard elements.</td>
<td><code>https://hostname:port</code></td>
</tr>
<tr>
<td>DestinationUsername</td>
<td>Specify the username of the carbon super user.</td>
<td><code>admin</code></td>
</tr>
</tbody>
</table>
### DestinationPassword
Specify the password of the carbon super user.

| TenantDomains | This parameter specifies the domains of all the tenants of whom the dashboards should be updated. If no value is specified for this parameter, the dashboards of all the tenants are updated. |

The parameters that are required to have values depend on which dashboard elements you need to migrate as explained in the table below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required Parameters</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| 1      | Convert gadgets, widgets, and layouts in CEP 4.1.0 into the newer version (i.e. based on WSO2 Dashboard Server 2.x.x) and copy them into the destination directory. This allows you to merge the migrated store into the newer version of the CEP dashboards store. | • Mode  
• SourceDir  
• DestinationDir | <DSMmigration>  
<Type>Portal</Type>  
<SourceDir>YOUR_PRODUCT_HOME</SourceDir>  
<DestinationDir>YOUR_DESTINATION_DIRECTORY</DestinationDir>  
<TrustStoreLocation></TrustStoreLocation>  
<TrustStorePassword></TrustStorePassword>  
<SourceURL></SourceURL>  
<SourceUsername></SourceUsername>  
<SourcePassword></SourcePassword>  
<DestinationURL></DestinationURL>  
<DestinationUsername></DestinationUsername>  
<DestinationPassword></DestinationPassword>  
<TenantDomains></TenantDomains>  
</DSMmigration> |
<table>
<thead>
<tr>
<th></th>
<th>Convert gadgets, widgets, and layouts in CEP 4.1.0 into the newer version (i.e. based on WSO2 Dashboard Server 2.x.x) and copy them into the destination directory. In addition, get all the dashboards to reside in your source server (specified via the sourceURL parameter) and migrate it to the newer version. Once migration is done, it is copied into the destination directory defined in the migration.xml file.</th>
</tr>
</thead>
</table>
|   | Mode  
|   | SourceDir  
|   | DestinationDir  
|   | TrustStoreLocation  
|   | TrustStorePassword  
|   | SourceURL  
|   | SourceUsername  
|   | SourcePassword |

```xml
<DSMigration>
  <Type>Portal</Type>
  <SourceDir>YOURPRODUCT_HOME</SourceDir>
  <DestinationDir>YOUR_DESTINATION_DIRECTORY</DestinationDir>
  <TrustStoreLocation>YOURPRODUCT_HOME/repository/resources/security/wso2carbon.jks</TrustStoreLocation>
  <TrustStorePassword>wso2carbon</TrustStorePassword>
  <SourceURL>https://localhost:9443</SourceURL>
  <SourceUsername>admin</SourceUsername>
  <SourcePassword>admin</SourcePassword>
  <DestinationURL></DestinationURL>
  <DestinationUsername></DestinationUsername>
  <DestinationPassword></DestinationPassword>
  <TenantDomains></TenantDomains>
</DSMigration>
```
3. Convert gadgets, widgets, and layouts in CEP 4.1.0 into the newer version (i.e., based on WSO2 Dashboard Server 2.x.x) and copy them into the destination directory. Update the dashboards in the server specified by the DestinationURL.

<table>
<thead>
<tr>
<th>Mode</th>
<th>SourceDir</th>
<th>DestinationDir</th>
<th>TrustStoreLocation</th>
<th>TrustStorePassword</th>
<th>SourceURL</th>
<th>SourceUsername</th>
<th>SourcePassword</th>
<th>DestinationURL</th>
<th>DestinationUsername</th>
<th>DestinationPassword</th>
</tr>
</thead>
</table>

This requires you to maintain the same trust store location and the trust password in both CEP versions.

```
<DSMmigration>
  <Type>Portal</Type>
  <SourceDir>YOUR_PRODUCT_HOME</SourceDir>
  <DestinationDir>YOUR_DESTINATION_DIRECTORY</DestinationDir>
  <TrustStoreLocation>YOUR_PRODUCT_HOME/repository/resources/security/wso2carbon.jks</TrustStoreLocation>
  <TrustStorePassword>wso2carbon</TrustStorePassword>
  <SourceURL>https://localhost:9443</SourceURL>
  <SourceUsername>admin</SourceUsername>
  <SourcePassword>admin</SourcePassword>
  <DestinationUsername>admin</DestinationUsername>
  <DestinationPassword>admin</DestinationPassword>
</DSMmigration>
```

3. Run the following command to run the migration script.
   **For Windows:** `<CEP-4.1.0_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.bat --run`
   **For Linux:** `<CEP-4.1.0_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.sh`

4. Perform the following steps for each gadget.

   `<DESTINATION_DIRECTORY>` is the directory you specified for the `DestinationDir` parameter in the Portal File Migration section of the `<CEP_HOME>/migration/dashboard/migration 1.0.x-2.0.x/migration.xml` file.

1. Navigate to the home directory of the gadget. (e.g., `<DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget`).
2. Replace the `<DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/main.js` file with the `<CEP_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/main.js` file.
3. Similarly, replace the `<DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/outputAdapterUiLibrary.js` file with `<CEP_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/outputAdapterUiLibrary.js` file.
Validating path changes

CEP 4.2.0 has a different directory structure to CEP 4.1.0 that affects the path to the gadget store as shown below.

CEP 4.1.0: `<CEP_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>/gadget`
CEP 4.2.0: `<CEP_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>/<StoreType [fs/es]]/gadget`

Therefore, if a gadget has relative paths included in its `index.xml` file (i.e. in the `<CEP_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>/<StoreType [fs/es]]/gadget/<GADGET_NAME>` path), you should check whether they are updated with the change in the directory structure. This check should be carried out after running the migration script.

e.g., A path is given as follows in the `index.xml` file in CEP 4.1.0.

```xml
<script language="javascript" type="text/javascript"
src="../../../../../js/igviz.js"></script>
```

The same link should be given as follows in CEP 4.2.0 due to the change in the directory structure (i.e. an additional `..` should be added for the new directory added to the structure).

```xml
<script language="javascript" type="text/javascript"
src=" ../../../js/igviz.js"></script>
```

Updating layout configurations

The dashboard layout configurations are defined in the `index.xml` file. This file differs as follows based on the CEP version.

<table>
<thead>
<tr>
<th>CEP Version</th>
<th>Location</th>
<th>Indexing</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.0</td>
<td><code>&lt;CEP_HOME&gt;/repository/deployment/server/jaggeryapps/portal/store/&lt;Tenant_Name&gt;</code></td>
<td>1 based indexing</td>
<td>`{ &quot;blocks&quot;: [ { &quot;id&quot;: &quot;a&quot;, &quot;row&quot;: 1, &quot;col&quot;: 1, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 }, { &quot;id&quot;: &quot;b&quot;, &quot;row&quot;: 1, &quot;col&quot;: 5, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 }, { &quot;id&quot;: &quot;c&quot;, &quot;row&quot;: 1, &quot;col&quot;: 9, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 }, { &quot;id&quot;: &quot;d&quot;, &quot;row&quot;: 4, &quot;col&quot;: 1, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 }, { &quot;id&quot;: &quot;e&quot;, &quot;row&quot;: 4, &quot;col&quot;: 5, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 }, { &quot;id&quot;: &quot;f&quot;, &quot;row&quot;: 4, &quot;col&quot;: 9, &quot;size_x&quot;: 4, &quot;size_y&quot;: 3 } ] }</td>
</tr>
</tbody>
</table>

The layouts migrated from CEP 4.1.0 to CEP 4.2.0 will have the format in which layouts are saved in CEP 4.1.0. To make them compatible with CEP 4.2.0
1. Replace "size_x" and "size_y" attributes with "width" and "height" respectively.
2. Replace "row" and "col" attributes with "x" and "y" respectively, and reduce their values by 1 to match the 0 based indexing.

   e.g., { "id": "a", "row": 1, "col": 1, "size_x": 4, "size_y": 3 } should be replaced with { "id": "a", "x": 0, "y": 0, "width": 4, "height": 3 }.

Testing the upgrade

Verify that all the required scenarios are working as expected. This confirms that the upgrade is successful.

Running the Product

To run WSO2 products, you start the product server at the command line. You can then run the Management Console to configure and manage the product.

   The Management Console uses the default HTTP-NIO transport, which is configured in the <CEP_HOME>/repository/conf/tomcat/catalina-server.xml file. (<CEP_HOME> is the directory where you installed the WSO2 product you want to run.) You must properly configure the HTTP-NIO transport in this file to access the Management Console. For more information on the HTTP-NIO transport, see the related topics section at the bottom of this page.

The following sections describe how to run the product.

• Starting the server
• Accessing the Management Console
• Stopping the server

Starting the server

To start the server, you run <CEP_HOME>/bin/wso2server.bat (on Windows) or <CEP_HOME>/bin/wso2server.sh (on Linux/Solaris/Mac OS) from the command prompt as described below. Alternatively, you can install and run the server as a Windows or Linux service (see the related topics section at the end of this page).

1. Open a command prompt by following the instructions below.
   - On Windows: Click Start -> Run, type cmd at the prompt, and then press Enter.
   - On Linux/Solaris/Mac OS: Establish an SSH connection to the server, log in to the text Linux console, or open a terminal window.
2. Execute one of the following commands:
   - To start the server in a typical environment:
     • On Windows: <CEP_HOME>\bin\wso2server.bat --run
     • On Linux/Solaris/Mac OS: sh <CEP_HOME>/bin/wso2server.sh start
   - To start the server in the background mode of Linux: sh <CEP_HOME>/bin/wso2server.sh start
     To stop the server running in this mode, you will enter: sh <CEP_HOME>/bin/wso2server.sh stop
   - To provide access to the production environment without allowing any user group (including admin) to log into the Management Console:
     • On Windows: <CEP_HOME>\bin\wso2server.bat --run -DworkerNode
     • On Linux/Solaris/Mac OS: sh <CEP_HOME>/bin/wso2server.sh -DworkerNode
   - To check for additional options you can use with the startup commands, type --help after the command, such as:
The operation log appears in the command window. When the product server has successfully started, the log displays the message "WSO2 Carbon started in 'n' seconds".

If you are running the CEP on a computing device that does not have a local display such as Amazon AWS EC2 instance. You need to enable headless property by adding the following system property to the start-up script located in below locations:

- **On Windows**: `<CEP_HOME>/bin/wso2server.bat`
- **On Linux/Solaris/Mac OS**: `sh <CEP_HOME>/bin/wso2server.sh -Djava.awt.headless=true`

### Accessing the Management Console

Once the server has started, you can run the Management Console by typing its URL in a Web browser. The following sections provide more information about running the Management Console:

- **Working with the URL**
- **Signing in**
- **Getting help**
- **Configuring the session time-out**
- **Restricting access to the Management Console and Web applications**

#### Working with the URL

The URL appears next to “Mgt Console URL” in the start script log that is displayed in the command window. For example:

```
2014-12-04 17:53:26,592 | INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} - WSO2 Carbon started in 45 sec.
```

The URL should be in the following format: `https://<Server Host>:9443/carbon`

You can use this URL to access the Management Console on this computer from any other computer connected to the Internet or LAN. When accessing the Management Console from the same server where it is installed, you can type `localhost` instead of the IP address as follows: `https://localhost:9443/carbon`

You can change the Management Console URL by modifying the value of the `<MgtHostName>` property in the `<CEP_HOME>/repository/conf/carbon.xml` file. When the host is internal or not resolved by a DNS, map the hostname alias to its IP address in the `/etc/hosts` file of your system, and then enter that alias as the value of the `<MgtHostName>` property in carbon.xml. For example:
In /etc/hosts:
127.0.0.1       localhost

In carbon.xml:
<MgtHostName>localhost</MgtHostName>

Signing in

At the sign-in screen, you can sign in to the Management Console using admin as both the username and password.

When the Management Console sign-in page appears, the Web browser typically displays an "insecure connection" message, which requires your confirmation before you can continue.

The Management Console is based on the HTTPS protocol, which is a combination of HTTP and SSL protocols. This protocol is generally used to encrypt the traffic from the client to server for security reasons. The certificate it works with is used for encryption only, and does not prove the server identity. Therefore, when you try to access the Management Console, a warning of untrusted connection is usually displayed. To continue working with this certificate, some steps should be taken to "accept" the certificate before access to the site is permitted. If you are using the Mozilla Firefox browser, this usually occurs only on the first access to the server, after which the certificate is stored in the browser database and marked as trusted. With other browsers, the insecure connection warning might be displayed every time you access the server.

This scenario is suitable for testing purposes, or for running the program on the company’s internal networks. If you want to make the Management Console available to external users, your organization should obtain a certificate signed by a well-known certificate authority, which verifies that the server actually has the name it is accessed by and that this server actually belongs to the given organization.

Getting help

The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed. To view information about a particular page, click the Help link at the top right corner of that page, or click the Docs link to open the documentation for full information on managing the product.

Configuring the session time-out

If you leave the Management Console unattended for a defined time, its login session will time out. The default timeout value is 15 minutes, but you can change this in the <session-timeout>15</session-timeout>

Restricting access to the Management Console and Web applications

You can restrict access to the Management Console of your product by binding the Management Console with selected IP addresses. You can either restrict access to the Management Console only, or you can restrict access to all Web applications in your server as explained below.
• To control access only to the Management Console, add the IP addresses to the `<CEP_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml` file as follows:

```xml
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
```

The RemoteAddrValve Tomcat valve defined in this file only applies to the Management Console, and thereby all outside requests to the Management Console are blocked.

• To control access to all Web applications deployed in your server, add the IP addresses to the `<CEP_HOME>/repository/conf/context.xml` file as follows.

```xml
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
```

The RemoteAddrValve Tomcat valve defined in this file applies to each Web application hosted on the WSO2 product server. Therefore, all outside requests to any Web application are blocked.

• You can also restrict access to particular servlets in a Web application by adding a Remote Address Filter to the `<CEP_HOME>/repository/conf/tomcat/web.xml` file and by mapping that filter to the servlet URL. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet. The following example from a `web.xml` file illustrates how access to the Management Console page (`/carbon/admin/login.jsp`) is granted only to one IP address.

```xml
<filter>
    <filter-name>Remote Address Filter</filter-name>
    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
    <init-param>
        <param-name>allow</param-name>
        <param-value>127.0.01</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>Remote Address Filter</filter-name>
    <url-pattern>/carbon/admin/login.jsp</url-pattern>
</filter-mapping>
```

Any configurations (including valves defined in the `<CEP_HOME>/repository/conf/tomcat/catalina-server.xml` file) apply to all Web applications and are globally available across the server, regardless of the host or cluster. For more information about using remote host filters, see the Apache Tomcat documentation.

**Stopping the server**
To stop the server, press Ctrl+C in the command window, or click the Shutdown/R sta rt link in the navigation pane in the Management Console. If you started the server in background mode in Linux, enter the following command instead:

```
sh <CEP_HOME>/bin/wso2server.sh stop
```

**Samples Guide**

This section explains how to set up and run WSO2 CEP samples.

- Setting up CEP Samples
- WSO2 CEP Samples

**Setting up CEP Samples**

For information on understanding the general flow of WSO2 CEP samples, see WSO2 CEP Samples. The following sections explain the generic setup instructions to execute the samples.

- Prerequisites
- Starting sample CEP configurations
- Starting sample consumers
- Starting sample producers
- Passing arguments to sample clients
- Setting up JMS for JMS sample clients
- Setting up MQTT for MQTT sample clients
- Setting up Kafka for Kafka sample clients

**Prerequisites**

Following applications are required for running the WSO2 CEP samples in this documentation.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Requirements</th>
</tr>
</thead>
</table>
| All samples          | - **Oracle Java SE Development Kit (JDK)/JRE** version 1.7.*/1.8.* (to launch the product and to run Apache Ant.)  
                       | - **Apache Ant** 1.7.0 or later (to compile and run the product samples.)  
                       | - WSO2 Complex Event Processor (CEP)                                         |
| JMS related samples  | The JMS samples are explained to be tried out using following JMS providers  |
                       | - **Apache Active MQ** 5.7.0 - 5.9.0 versions.                               |
                       | - **Apache Qpid** 0.32 version.                                              |
                       | - WSO2 Message Broker (MB) 3.0.0-ALPHA2 version. For instructions on WSO2 MB, go to Message Broker documentation. |
                       | - For more information on the prerequisites required for the JMS samples, see Setting up JMS samples. |
### MQTT related samples

The MQTT samples are explained to be tried out using following MQTT-supported servers:

- **Apache Active MQ 5.11.1 version or**
- **Mosquitto 1.4.3 version**
- For more information on the prerequisites required for the MQTT samples, see [Setting up MQTT samples](#).

### Kafka related samples

The Kafka samples are explained to be tried out using following Kafka Broker versions:

- **Apache Kafka 2.10-0.8.2.1 version**
- For more information on the prerequisites required for the Kafka samples, see [Setting up Kafka samples](#).

### WebSocket related samples

Java Development Kit / JRE version 1.7.*

### Apache Storm related samples

Apache Storm version 0.9.3 or later (to run Storm samples.)

---

### Starting sample CEP configurations

To start the CEP with a sample configuration, run the following command with `-sn <n>`, where `<n>` denotes the number assigned to the sample.

```bash
On Linux: ./wso2cep-samples.sh -sn <n>
On Windows: wso2cep-samples.bat -sn <n>
```

**For example,** to start the CEP with the configuration of sample 0101, run the following command inside `<CEP_HOME>/bin` directory:

```bash
On Linux: ./wso2cep-samples.sh -sn 0101
On Windows: wso2cep-samples.bat -sn 0101
```

The `<CEP_HOME>/samples/cep/artifacts` directory contains the sample configurations of CEP. Each configuration is inside a sub directory by the name of the sample numbered `<n>`. For example, the cep artifacts for sample 0101 can be found in the `<CEP_HOME>/samples/cep/artifacts/0101` directory.

In the normal mode the `<CEP_HOME>/bin/wso2server.bat` or `<CEP_HOME>/bin/wso2server.sh` script starts an instance of the CEP using the configuration files in `<CEP_HOME>/repository/deployment/server` directory and any sample configurations passed in as `-sn <n>` is ignored.

These configurations on running the samples point the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/<SAMPLE_NUMBER>/` directory. (<CEP_HOME>/repository/deployment/server/ directory is used as the default Axis2 repo.)

---

### Starting sample consumers
Each sample consumer service is saved in a separate directory as `<CEP_HOME>/samples/cep/consumers/<consumer_name>`. 

1. To start a sample consumer, go to its directory `<CEP_HOME>/samples/cep/consumers/<consumer_name>` and type `ant`. For example,

```bash
user@host:/tmp/wso2cep-4.2.0/samples/consumers/wso2-event$ ant
Buildfile: /home/user/tmp/wso2cep-4.2.0/samples/consumers/wso2-event/build.xml
...
run:[echo] To configure host, port and events use -Dhost=xxxx -Dport=xxx -Devents=xx
[echo] Sending to : http://localhost:7661
[java] Test Server starting on 10.100.0.75
[java] Thrift Server started at 10.100.0.75
[java] Thrift SSL port : 7761
[java] Thrift port : 7661
[java] Test Server Started
```

To write a custom `wso2Event` data publisher (Thrift data publisher), use the pom file given here.

2. Deploy the log service sample consumer, which is a Web service, by specifying the sample number as follows:

```bash
ant -DsSampleNo=<sample no>
```

Running `DsSampleNo` ant script deploys the log service in the `axis2` repository that is relevant to the specified sample. After proper deployment, the Web service is able to receive messages from the CEP server.
Starting sample producers

Starting a sample producer is similar to starting a consumer.

1. Go to the sample producer’s directory `<CEP_HOME>/samples/cep/producers/<producer_name>` and type `ant` with relevant input arguments. For example,

```
user@host:/home/user/wso2cep-4.2.0/samples/producers/pizza-shop$ ant
pizzaOrderClient -Dservice=WSEventLocalAdaptorService
-DtopicName=BatchedPizzaOrder -DbatchedEvents=true
```

Passing arguments to sample clients

Some sample clients take extra arguments. The following table presents the format in which these arguments can be passed.
<table>
<thead>
<tr>
<th>Purpose</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>To specify the publishing topic for the producer client.</td>
<td><code>-DtopicName=XXXX</code></td>
<td><code>ant -DtopicName=AllStockQuotes</code></td>
</tr>
<tr>
<td>To publish to a specific host, which is an IP address.</td>
<td><code>-Dhost=XXXX</code></td>
<td><code>ant -Dhost=org.test.domain</code></td>
</tr>
<tr>
<td>To publish to a specific port.</td>
<td><code>-Dport=XXXX</code></td>
<td><code>ant pizzaOrderClient -Dport=9764</code></td>
</tr>
<tr>
<td>To publish to a specific Web service.</td>
<td><code>-Dservice=XXXX</code></td>
<td><code>ant pizzaOrderClient -Dservice=wsInAd.</code></td>
</tr>
<tr>
<td>To send events in batches (i.e., the adapter receives a batch of events).</td>
<td>`-DbatchedEvents={true</td>
<td>false}`</td>
</tr>
<tr>
<td>To publish events to a specific client URL.</td>
<td><code>-Durl='client url'</code></td>
<td><code>ant</code></td>
</tr>
<tr>
<td>To subscribe events from a JMS topic (consumer).</td>
<td><code>-DtopicName=XXXXXX</code></td>
<td><code>ant topicConsumer -DtopicName=TestTop.</code></td>
</tr>
<tr>
<td>To subscribe events from a JMS queue (consumer).</td>
<td><code>-Dqueue=XXXXX</code></td>
<td><code>ant queueConsumer -Dqueue=DelayedFlightStats</code></td>
</tr>
<tr>
<td>To receive events from a specific format from the text document (producer).</td>
<td><code>-Dformat=xxxx(csv, text, json, xml)</code></td>
<td><code>ant -Dformat=csv</code></td>
</tr>
<tr>
<td>To specify the JMS broker to which the CEP server listens.</td>
<td><code>-Dbroker=xxxx(activemq, mb, qpid)</code></td>
<td><code>ant -Dbroker=activemq</code></td>
</tr>
<tr>
<td>To publish events in a specific event stream (producer).</td>
<td><code>-DstreamId=xxxx:x.x.x</code></td>
<td><code>ant -DstreamId=org.wso2.event.sensor.</code></td>
</tr>
<tr>
<td>To publish events from the specific sample folder (producer).</td>
<td><code>-Dsn='sample number' or -DfilePath=xxxx</code></td>
<td><code>ant -Dsn=00</code></td>
</tr>
<tr>
<td>To specify whether the protocol based on which events are received is thrift or binary.</td>
<td>-Dprotocol='thrift/binary'</td>
<td>ant -Dprotocol=binary</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>To specify the username when an action performed by a sample requires user credentials to be specified.</td>
<td>-Dusername=xxxx</td>
<td>-Dusername=admin</td>
</tr>
<tr>
<td>To specify the password when an action performed by a sample requires user credentials to be specified.</td>
<td>-Dpassword=xxxx</td>
<td>-Dpassword=admin</td>
</tr>
<tr>
<td>When doing a performance test, this argument specifies the number of events with which the test should be carried out.</td>
<td>-Devents=xx or -DnoOfEvents=xxxx or -DeventCount=xxxx</td>
<td>-Devents=2000000</td>
</tr>
<tr>
<td>When doing a performance test, this argument specifies the delay that occurs between events in milli seconds.</td>
<td>-Ddelay='delay between events in ms'</td>
<td>-Ddelay=1000</td>
</tr>
<tr>
<td>When doing a performance test, this argument specifies the number of events after which the throughput/latency should be calculated.</td>
<td>-DelapsedCount=xxxx</td>
<td>-DelapsedCount=10000</td>
</tr>
<tr>
<td>When doing a performance test, this argument specifies the number of publishers that should be used to publish events.</td>
<td>-DnoOfPublishers=xxxx</td>
<td>-DnoOfPublishers=50</td>
</tr>
</tbody>
</table>
When doing a performance test, this argument specifies the number of events that should be sent to the event flow for the CEP server to warm up and reach a stabilize.

- \texttt{DwarmUpCount=xxxx}
- \texttt{DwarmUpCount=200000}

When doing a performance test, this argument is used to specify whether you want to calculate the throughput or the latency.

- \texttt{DcalcType='throughput/latency'}
- \texttt{DcalcType=throughput}

### Setting up JMS for JMS sample clients

Before you run JMS samples, set up and start a JMS provider. Configure JMS providers by copying relevant JMS client libraries to `<CEP_HOME>/samples/cep/lib` folder as mentioned below.

For Apache ActiveMQ, the relevant JAR files are,

- `<ActiveMQ_HOME>/activemq-all-5.7.0.jar`
- `<ActiveMQ_HOME>/lib/geronimo-jms_1.1_spec-1.1.1.jar`

Previous Apache ActiveMQ versions may not contain SLF4J related files in the client JAR. Therefore, if you get an error, add SLF4J related JAR file to `<CEP_HOME>/samples/cep/lib/` directory of the samples.

For Apache Qpid, the relevant JAR files are,

- `<QPID-CLIENT_HOME>/lib/geronimo-jms_1.1_spec-1.1.1.jar`
- `<QPID-CLIENT_HOME>/lib/qpid-client-0.32.jar`
- `<QPID-CLIENT_HOME>/lib/qpid-common-0.32.jar`

For WSO2 Message Broker (MB), the relevant JAR files are,
Setting up MQTT for MQTT sample clients

Before you run MQTT samples, set up and start a MQTT-supported server. Configure MQTT sample clients by copying relevant MQTT client libraries to `<CEP_HOME>/samples/cep/lib` folder as mentioned below.

- Download and add MQTT client library (mqtt-client-0.4.0.jar) to `<CEP_HOME>/samples/cep/lib` directory.

Setting up Kafka for Kafka sample clients

Before you run Kafka samples, set up and start a Kafka broker. Configure Kafka sample clients by copying relevant Kafka client libraries to `<CEP_HOME>/samples/cep/lib` folder as mentioned below.

- Copy all the JAR files, which are located in `<KAFKA_HOME>/libs/` directory to `<CEP_HOME>/samples/cep/lib/` directory.

WSO2 CEP Samples

WSO2 CEP samples explain different use cases of the product using sample clients (producers and consumers). The general flow of all samples is as follows.

1. Each sample starts WSO2 CEP with a different configuration.
2. After WSO2 CEP starts, the sample producers send different types of events to the CEP over different transports.
3. The CEP receives these events and processes them.
4. Finally it pushes different types of notification events based on the processed data to the sample consumers over different transports.

This section includes a set of samples, demonstrating use cases of WSO2 CEP.

- Samples on Receiving Events
- Samples on Processing Events
- Samples on Publishing Events

The following table summarizes the producer/consumer clients and the mapping types of each WSO2 CEP sample. For instructions to set up the samples, see Setting up CEP Samples.

### Samples on receiving events

<table>
<thead>
<tr>
<th>Sample no.</th>
<th>Description</th>
<th>Producer (publish topic)</th>
<th>Event receiver type</th>
<th>Receiver message format</th>
<th>Event publisher type</th>
<th>Publisher message format</th>
<th>Consumer (subscription topic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Receiving JSON events via HTTP transport.</td>
<td>http</td>
<td>HTTP</td>
<td>JSON</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Protocol</td>
<td>Format</td>
<td>Logger</td>
<td>Event Type</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------------</td>
<td>--------</td>
<td>------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>Receiving custom JSON events via HTTP transport.</td>
<td>http</td>
<td>HTTP (JSON)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>Receiving XML events via HTTP transport.</td>
<td>http</td>
<td>HTTP (XML)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>Receiving custom XML events via HTTP transport.</td>
<td>http</td>
<td>HTTP (XML)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>Receiving text events via HTTP transport.</td>
<td>http</td>
<td>HTTP (text)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>Receiving custom text events via HTTP transport.</td>
<td>http</td>
<td>HTTP (text)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td>Receiving WSO2 events via WSO2 Event Receiver.</td>
<td>wso2event</td>
<td>WSO2Event</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>Receiving custom WSO2 events via WSO2 Event Receiver.</td>
<td>wso2event</td>
<td>WSO2Event (custom)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>Receiving map events via JMS transport - ActiveMQ.</td>
<td>jms</td>
<td>JMS (ActiveMQ)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>Receiving custom map events via JMS transport - ActiveMQ.</td>
<td>jms</td>
<td>JMS (ActiveMQ)</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td>Transport</td>
<td>Events</td>
<td>Logger</td>
<td>Type</td>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>Receiving JSON, text, XML events via JMS transport - ActiveMQ.</td>
<td>jms</td>
<td>JMS</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>Receiving map, XML events via JMS transport - Qpid</td>
<td>jms</td>
<td>JMS</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0013</td>
<td>Receiving map, text events via JMS transport - WSO2 Message Broker.</td>
<td>jms</td>
<td>JMS</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>Receiving XML events via SOAP transport.</td>
<td>soap</td>
<td>SOAP</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td>Receiving text events via Email transport.</td>
<td>None</td>
<td>Email</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>Receiving JSON events via MQTT transport.</td>
<td>mqtt</td>
<td>MQTT</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>Receiving custom text events via file tail transport.</td>
<td>file</td>
<td>file-tail</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>Receiving JSON events via Kafka transport.</td>
<td>kafka</td>
<td>kafka</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>Receiving JSON events via WebSocket transport.</td>
<td>websocket</td>
<td>websocket</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
### Simple JSON pass-through with WebSocket local input event adaptor.

- **0020**
  - Producer type: None
  - Event receiver type: WebSocket local
  - Event message format: JSON
  - Event publisher type: logger
  - Publisher message format: text
  - Consumer (subscription topic): None

#### Receiving map events via JMS transport - ActiveMQ (For Queue)

- **0021**
  - Producer type: jms
  - Event receiver type: JMS (ActiveMQ)
  - Event message format: map
  - Event publisher type: logger
  - Publisher message format: text
  - Consumer (subscription topic): None

#### Receiving custom RegEx events via text events via file tail transport.

- **0022**
  - Producer type: file
  - Event receiver type: file-tail
  - Event message format: text (custom)
  - Event publisher type: logger
  - Publisher message format: text
  - Consumer (subscription topic): None

---

### Samples on processing events

<table>
<thead>
<tr>
<th>Sample no.</th>
<th>Description</th>
<th>Producer (publish topic)</th>
<th>Event receiver type</th>
<th>Receiver message format</th>
<th>Event publisher type</th>
<th>Publisher message format</th>
<th>Consumer (subscription topic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>Pass-through/projection query in an execution plan.</td>
<td>wso2-event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0102</td>
<td>Projections, transformations and enrichment for events.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0103</td>
<td>Using filters for generating alerts.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0104</td>
<td>Calculations over time using windows.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0105</td>
<td>Performing joins with windows.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0106</td>
<td>Using in-memory event tables.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
<td>Type</td>
<td>Source</td>
<td>Action</td>
<td>Log</td>
<td>Format</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>0107</td>
<td>Using RDBMS event tables.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0108</td>
<td>Using patterns to detect ATM transaction frauds.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0109</td>
<td>Detecting trends with sequences.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0110</td>
<td>Sequences with partitioning to detect trends.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0111</td>
<td>Detecting non-occurrences with patterns.</td>
<td>http</td>
<td>HTTP</td>
<td>text</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0112</td>
<td>Analyzing Twitter feeds using partitions.</td>
<td>wso2-event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>logger</td>
<td>JSON</td>
<td>None</td>
</tr>
<tr>
<td>0113</td>
<td>Limiting the output rate of an event stream.</td>
<td>http</td>
<td>HTTP</td>
<td>XML</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0114</td>
<td>Using external time windows.</td>
<td>wso2-event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0115</td>
<td>Quartz scheduler based alerts.</td>
<td>http</td>
<td>HTTP</td>
<td>XML</td>
<td>logger</td>
<td>JSON</td>
<td>None</td>
</tr>
<tr>
<td>0116</td>
<td>Performing linear regression.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0117</td>
<td>Filtering and outputting to multiple streams.</td>
<td>file</td>
<td>file-tail</td>
<td>text</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0118</td>
<td>Using Hazlecast event tables.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>Sample no.</td>
<td>Description</td>
<td>Producer (publish topic)</td>
<td>Event receiver type</td>
<td>Receiver message format</td>
<td>Event publisher type</td>
<td>Publisher message format</td>
<td>Consumer (subscription topic)</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>--------------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>0119</td>
<td>Trigger events at defined time intervals.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0301</td>
<td>Network intruder detection with PMML extension predictions.</td>
<td>None</td>
<td>(Event Simulator)</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0501</td>
<td>Processing a simple filter query with Apache Storm Deployment.</td>
<td>wso2-event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>wso2-event</td>
<td></td>
</tr>
<tr>
<td>0502</td>
<td>Processing a window state over server restart.</td>
<td>http</td>
<td>HTTP</td>
<td>XML</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0503</td>
<td>Processing a window query in high availability mode.</td>
<td>http</td>
<td>HTTP</td>
<td>XML</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>0504</td>
<td>Processing a distributed Siddhi query with partitioning by integrating with Apache Storm.</td>
<td>wso2-event</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>logger</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>1001</td>
<td>WSO2 CEP Geo Dashboard.</td>
<td>http</td>
<td>HTTP</td>
<td>JSON</td>
<td>websocket-local</td>
<td>JSON</td>
<td>None</td>
</tr>
</tbody>
</table>

Samples on publishing events

<table>
<thead>
<tr>
<th>Sample no.</th>
<th>Description</th>
<th>Producer (publish topic)</th>
<th>Event receiver type</th>
<th>Receiver message format</th>
<th>Event publisher type</th>
<th>Publisher message format</th>
<th>Consumer (subscription topic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0051</td>
<td>Publishing JSON events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>JSON</td>
<td>None</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
<td>Logger Type</td>
<td>Transport Type</td>
<td>Transport Details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>--------------</td>
<td>---------------</td>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0052</td>
<td>Publishing custom JSON events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>JSON (custom)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0053</td>
<td>Publishing XML events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>XML</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0054</td>
<td>Publishing custom XML events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>XML (custom)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0055</td>
<td>Publishing text events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>text</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0056</td>
<td>Publishing custom text events via logger transport.</td>
<td>None</td>
<td>None</td>
<td>logger</td>
<td>text (custom)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>0057</td>
<td>Publishing WSO2 events via WSO2Event transport.</td>
<td>None</td>
<td>None</td>
<td>WSO2Event</td>
<td>WSO2Event</td>
<td>wso2-event</td>
<td></td>
</tr>
<tr>
<td>0058</td>
<td>Publishing custom WSO2 events via WSO2Event transport.</td>
<td>None</td>
<td>None</td>
<td>WSO2Event</td>
<td>WSO2Event (custom)</td>
<td>wso2-event</td>
<td></td>
</tr>
<tr>
<td>0059</td>
<td>Publishing map and text events via JMS transport - ActiveMQ.</td>
<td>None</td>
<td>None</td>
<td>JMS (ActiveMQ)</td>
<td>map/text</td>
<td>jms</td>
<td></td>
</tr>
<tr>
<td>0060</td>
<td>Publishing custom map and JSON events via JMS transport - Qpid.</td>
<td>None</td>
<td>None</td>
<td>JMS (Qpid)</td>
<td>map (custom)/JSON (custom)</td>
<td>jms</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>0061</td>
<td>Publishing map and XML events via JMS transport - WSO2 MB.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>JMS (WSO2 MB)</td>
<td>map/XML</td>
<td>jms</td>
</tr>
<tr>
<td>0062</td>
<td>Publishing XML, JSON and custom text events via HTTP transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>HTTP</td>
<td>JSON/text (custom)/XML</td>
<td>generic-log-service</td>
</tr>
<tr>
<td>0063</td>
<td>Publishing XML events via SOAP transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>SOAP</td>
<td>XML</td>
<td>axis2-log-service</td>
</tr>
<tr>
<td>0064</td>
<td>Publishing text events via Email transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>email</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>0065</td>
<td>Publishing JSON events via SMS transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>SMS</td>
<td>JSON</td>
<td>SMSC Simulator</td>
</tr>
<tr>
<td>0066</td>
<td>Publishing JSON events via MQTT transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>MQTT</td>
<td>JSON</td>
<td>mqtt</td>
</tr>
<tr>
<td>0067</td>
<td>Publishing map events via Cassandra transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Cassandra</td>
<td>map</td>
<td></td>
</tr>
<tr>
<td>0068</td>
<td>Publishing XML events via Kafka transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Kafka</td>
<td>XML</td>
<td></td>
</tr>
<tr>
<td>0069</td>
<td>Publishing JSON events via WebSocket transport.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Websocket</td>
<td>JSON</td>
<td>websocket</td>
</tr>
<tr>
<td>0070</td>
<td>Publishing JSON events via WebSocket local output event adapter.</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>WebSocket local</td>
<td>JSON</td>
<td>websocket</td>
</tr>
</tbody>
</table>
Samples on Receiving Events

- Sample 0001 - Receiving JSON Events via HTTP Transport
- Sample 0002 - Receiving Custom JSON Events via HTTP Transport
- Sample 0003 - Receiving XML Events via HTTP Transport
- Sample 0004 - Receiving Custom XML Events via HTTP Transport
- Sample 0005 - Receiving Text Events via HTTP Transport
- Sample 0006 - Receiving Custom Text Events via HTTP Transport
- Sample 0007 - Receiving WSO2 Events Via WSO2Event Receiver
- Sample 0008 - Receiving Custom WSO2 Events via WSO2Event Receiver
- Sample 0009 - Receiving Map Events via JMS Transport - ActiveMQ
- Sample 0010 - Receiving Custom Map Events via JMS Transport - ActiveMQ
- Sample 0011 - Receiving JSON, Text, XML Events via JMS Transport - ActiveMQ
- Sample 0012 - Receiving Map, XML Events via JMS Transport - Qpid
- Sample 0013 - Receiving Map, Text Events via JMS Transport - WSO2 MB
- Sample 0014 - Receiving XML Events via Soap Transport
- Sample 0015 - Receiving Text Events via Email Transport
- Sample 0016 - Receiving JSON Events via MQTT Transport
- Sample 0017 - Receiving Custom Text Events via File Tail Transport
- Sample 0018 - Receiving JSON Events via Kafka Transport
- Sample 0019 - Receiving JSON Events via WebSocket Transport
- Sample 0020 - Simple JSON Pass-through with Websocket-Local Input Event Adapter
- Sample 0021 - Receiving Map Events via JMS Transport - ActiveMQ (For Queue)
- Sample 0022 - Receiving Custom RegEx Text Events via File Tail

Sample 0001 - Receiving JSON Events via HTTP Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive incoming JSON events that adheres to the WSO2 Event format via the HTTP transport. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

Instead of using logger publishers, you can also use the Event Tracer or Event Metrics to monitor received and published events as well as the memory consumption of each execution plan.

Prerequisites

Set up the prerequisites required for all samples.
Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0001. For instructions, see Starting sample CEP configurations. The sample configuration does the following.

- Creates an event stream named org.wso2.event.sensor.stream:1.0.0.
- Creates an event receiver named httpReceiver.
- Creates an event publisher named httpLogger to log the received messages.

Executing the sample

Open a new tab in the CLI and execute the following ant command from the <CEP_HOME>/samples/cep/produ cers/http/ directory.

```
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0001
```

The other optional parameters that can be used in the above command are defined in the <CEP_HOME>/sa mples/cep/ producers/http/build.xml file.

This builds the HTTP client and publishes the events in the <CEP_HOME>/samples/cep/artifacts/0001/htt pReceiver.txt file to the httpReceiver endpoint. You can view the details of the events that are sent as shown in the log below.
The logs of the JSON events received by the CEP server will be displayed in the CLI as shown in the example below.
Sample 0002 - Receiving Custom JSON Events via HTTP Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to receive incoming custom JSON events via the HTTP transport. Custom events are events with custom mappings that do not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not process the incoming events. The log event publisher is used to log the received events, and to verify the messages.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0002. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0002`.
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `httpReceiver` with custom mapping.
- Creates an event publisher named `httpLogger` to log the received messages.

**Executing the sample**

Open another tab in the CLI, and issue the following `ant` command from the `<CEP_HOME>/samples/cep/producers/http` directory.
It builds the http client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0002/httpReceiver.txt` to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.

```
[echo] INFO org.wso2.carbon.sample.Http.Http -
[echo] { "sensorData": {
  "timestamp": 19000813115534,
  "powerSaved": false,
  "so": 581,
  "name": temperature,
  "long": 98.34344,
  "lat": 20.44345,
  "humidity": 2.3,
  "temp": 20.44345
}
```

```
[echo] INFO org.wso2.carbon.sample.Http.Http -
[echo] { "sensorData": {
  "timestamp": 19000813115534,
  "powerSaved": false,
  "so": 582,
  "name": temperature,
  "long": 98.34344,
  "lat": 20.44345,
  "humidity": 2.3,
  "temp": 20.44345
}
```

```
[echo] INFO org.wso2.carbon.sample.Http.Http -
[echo] { "sensorData": {
  "timestamp": 19000813115534,
  "powerSaved": false,
  "so": 583,
  "name": temperature,
  "long": 98.34344,
  "lat": 20.44345,
  "humidity": 2.3,
  "temp": 20.44345
}
```

```
[echo] INFO org.wso2.carbon.sample.Http.Http -
[echo] { "sensorData": {
  "timestamp": 19000813115534,
  "powerSaved": false,
  "so": 583,
  "name": temperature,
  "long": 98.34344,
  "lat": 20.44345,
  "humidity": 2.3,
  "temp": 20.44345
}
```

Events received by WSO2 CEP are logged as follows.
Sample 0003 - Receiving XML Events via HTTP Transport

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive incoming XML events that adhere to the WSO2 Event format via HTTP transport. This sample does not process the incoming events. A log event publisher is used to log the received event.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0003. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0003`.
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `httpReceiver`.
- Creates an event publisher named `httpLogger` to log the received messages.

**Executing the sample**
Open another tab in the CLI, and issue the following ant command from the `<CEP_HOME>/samples/cep/producers/http` directory.

```shell
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0003
```

It builds the http client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0003/httpReceiver` file to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.

```
[echo] Configure -Durl=xxxx and (-DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -
Dpassword=xxxx
[echo] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[echo] [event]
[java] <event>
[java]  <metaData>
[java]   <timestamp>199008131245</timestamp>
[java]   <isPowerSaverEnabled>true</isPowerSaverEnabled>
[java]   <sensorId>4</sensorId>
[java]   <sensorName>temperature</sensorName>
[java] </metaData>
[java] <correlationData>
[java]   <longitude>4.504343</longitude>
[java]   <latitude>1.23434</latitude>
[java] </correlationData>
[java] <payloadData>
[java]   <humidity>6.6</humidity>
[java]   <sensorValue>28.44345</sensorValue>
[java] </payloadData>
[java] </event>
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] [event]
[java]  <metaData>
[java]   <timestamp>199008131245</timestamp>
[java]   <isPowerSaverEnabled>true</isPowerSaverEnabled>
[java]   <sensorId>4</sensorId>
[java]   <sensorName>temperature</sensorName>
[java] </metaData>
[java] <correlationData>
[java]   <longitude>4.504343</longitude>
[java]   <latitude>1.23434</latitude>
[java] </correlationData>
[java] <payloadData>
[java]   <humidity>6.6</humidity>
[java]   <sensorValue>28.44345</sensorValue>
[java] </payloadData>
[java] </event>
[java] [event]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] [event]
[java]  <metaData>
[java]   <timestamp>199008131245</timestamp>
[java]   <isPowerSaverEnabled>true</isPowerSaverEnabled>
[java]   <sensorId>4</sensorId>
[java]   <sensorName>temperature</sensorName>
[java] </metaData>
[java] <correlationData>
[java]   <longitude>4.504343</longitude>
[java]   <latitude>1.23434</latitude>
[java] </correlationData>
[java] <payloadData>
[java]   <humidity>6.6</humidity>
[java]   <sensorValue>28.44345</sensorValue>
[java] </payloadData>
[java] </event>
[java] [event]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] [event]
[java]  <metaData>
[java]   <timestamp>199008131245</timestamp>
[java]   <isPowerSaverEnabled>true</isPowerSaverEnabled>
[java]   <sensorId>4</sensorId>
[java]   <sensorName>temperature</sensorName>
[java] </metaData>
[java] <correlationData>
[java]   <longitude>4.504343</longitude>
[java]   <latitude>1.23434</latitude>
[java] </correlationData>
[java] <payloadData>
[java]   <humidity>6.6</humidity>
[java]   <sensorValue>28.44345</sensorValue>
[java] </payloadData>
[java] </event>
```

The events received by WSO2 CEP are logged as follows.
Sample 0004 - Receiving Custom XML Events via HTTP Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive incoming custom XML events via the HTTP transport. Custom events are events with custom mappings that do not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

Prerequisites

Set up the prerequisites required for all samples

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0004. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0004`.
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `httpReceiver` with custom mapping.
- Creates an event publisher named `httpLogger` to log the received messages.

Executing the sample

Open another tab in the CLI and issue the following `ant` command from the `<CEP_HOME>/samples/cep/producers/http` directory.

```bash
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0004
```
This builds the HTTP client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0004/httpReceiver.txt` file to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.

The events received by WSO2 CEP are logged as shown below.
Sample 0005 - Receiving Text Events via HTTP Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive incoming text events that adhere to the WSO2 Event format via the HTTP transport. This sample does not process the incoming events. A log event publisher is used to log the received events.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0005. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0005`.
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `httpReceiver`.
- Creates an event publisher named `httpLogger` to log the received messages.

Executing the sample

Open another tab in the CLI and issue the following ant command from the `<CEP_HOME>/samples/cep/produc
The events received by WSO2 CEP are logged as shown below.

Sample 0006 - Receiving Custom Text Events via HTTP Transport
Introduction

This sample demonstrates how to receive incoming custom text events via the HTTP transport. Custom events are events with custom mappings that do not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0006. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0006`.
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `httpReceiver` with custom mapping.
- Creates an event publisher named `httpLogger` to log the received messages.

Executing the sample

Open another tab in the CLI and issue the following ant command from the `<CEP_HOME>/cep/samples/produce` directory.

```
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0006
```

This builds the HTTP client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0006/httpReceiver.txt` file to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.
The events received by WSO2 CEP are logged as shown below.

Sample 0007 - Receiving WSO2 Events Via WSO2Event Receiver

- Introduction
- Prerequisites
- Building the sample
- Executing the sample
**Introduction**

This sample demonstrates how to receive incoming WSO2 events via the WSO2Event receiver. WSO2Event receiver is implemented based on Apache Thrift. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0007. For instructions, see [Starting sample CEP configurations](#). This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0007`
- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `wso2eventReceiver`.
- Creates an event publisher named `wso2eventLogger` to log the received messages.

**Executing the sample**

Open another tab in the CLI and issue the following `ant` command from the `<CEP_HOME>/samples/cep/producers/wso2-event` directory.

```
ant -DstreamId=org.wso2.event.sensor.stream:1.0.0 -Dsn=0007
```

This builds the `wso2event` client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0007/org_wso2_event_sensor_stream_1_0_0.csv` to the `wso2eventReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.
The events received by WSO2 CEP are logged as shown below.

Sample 0008 - Receiving Custom WSO2 Events via WSO2Event Receiver

- Introduction
Introduction

This sample demonstrates how to receive custom incoming WSO2 event objects via the WSO2Event receiver and then map a few attributes with different names. Custom events are events with custom mappings that do not adhere to the default WSO2 Event format. For more information on event formats, see Event Formats. WSO2Event receiver is implemented based on Apache Thrift. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0008. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0008`.
- Creates the event streams `org.wso2.event.sensor.stream:1.0.0` and `org.wso2.mapped.sensor.data:1.0.0`.
- Creates an event receiver named `wso2eventReceiver`.
- Creates an event publisher named `wso2eventLogger` to log the received messages.

Executing the sample

Open a new tab in the CLI and issue the following `ant` command from the `<CEP_HOME>/samples/cep/produce` directory.

```
ant -DstreamId=org.wso2.event.sensor.stream:1.0.0 -Dsn=0008
```

This builds the `wso2event` client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0008/org_wso2_event_sensor_stream_1_0_0.csv` file to the `wso2eventReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.
The events received by WSO2 CEP are logged as shown below.

```
[2016-03-10 10:39:05,546] INFO {org.wso2.carbon.databridge.core.Databridge} - user admin connected
meta_id:501,
meta_isPowerSavingMode: false,
correlation_latitude:90.34344,
correlation_longitude:20.44345,
humidity:23.3,
value:20.44345
meta_id:502,
meta_isPowerSavingMode: false,
correlation_latitude:90.34344,
correlation_longitude:20.44345,
humidity:23.3,
value:20.44345
meta_id:503,
meta_isPowerSavingMode: false,
correlation_latitude:90.34344,
correlation_longitude:20.44345,
humidity:23.3,
value:20.44345
```

Note the following in the above log.

- The events received are being mapped to a new stream named `org.wso2.mapped.sensor.data:1.0.0`.
- In the event stream `org.wso2.mapped.sensor.data:1.0.0`, the attribute named `sensorName` has been dropped. The attributes `isPowerEnabled` and `sensorValue` are mapped to two different attributes named...
isPowerSavingMode and value respectively.

Sample 0009 - Receiving Map Events via JMS Transport - ActiveMQ

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive incoming map events that adhere to the WSO2 Event format via the JMS transport. This sample does not process the incoming events. A log event publisher is used to log the received events, and to verify the messages.

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Set up the prerequisites required for all samples.
2. Navigate to <ACTIVEMQ_HOME>/bin/ directory, and execute the following command to start the Apache ActiveMQ server: ./activemq console
3. Configure WSO2 CEP by adding relevant libraries to support JMS transport
4. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0009. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from <CEP_HOME>/repository/deployment/server to <CEP_HOME>/samples/cep/artifacts/0009
- Creates a stream with id org.wso2.event.sensor.stream:1.0.0.
- Creates an event receiver with name jmsReceiver.
- Creates an event publisher to log the received messages with name jmsLogger.

Executing the sample

1. Wait until the CEP terminal prompts a message similar as follows.

   
   2015-05-17 23:02:55,153 INFO org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTsreamListener Connection attempt 1 for JMS Provider for listener jmsReceiverMap#topicMap was successful!
   
   2015-05-17 23:02:55,168 INFO org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTsreamListener - Task manager for event adapter jmsReceiverMap [re:] initialized
   
   2015-05-17 23:02:56,161 INFO org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTsreamListener - Started to listen on destination: topicMap of type topic for listener jmsReceiverMap#topicMap

2. Open another terminal, go to <CEP_HOME>/samples/cep/producers/jms
and run the following command:

```
ant -DtopicName=topicMap -Dformat=map -Dbroker=activemq -Dsn=0009
```

It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0009/topicMap.csv` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console.

Sample 0010 - Receiving Custom Map Events via JMS Transport - ActiveMQ

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive incoming custom map events via JMS transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see [Event Formats](#). This sample does not do any processing on the incoming event. The log event publisher is used to log the received events, and to verify the messages.

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Set up the prerequisites required for all samples.

2. Navigate to `<ACTIVEMQ_HOME>/bin/` directory, and execute the following command to start the Apache ActiveMQ server: `./activemq console`

This guide uses ActiveMQ versions 5.7.0 - 5.9.0. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to [Apache ActiveMQ Documentation](#).
3. Configure WSO2 CEP by adding relevant libraries to support JMS transport

4. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0010. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0010`
- Creates a stream with id org.wso2.event.sensor.stream:1.0.0.
- Create and event receiver with name jmsReceiver with custom mapping.
- Create and event publisher to log the received messages with name jmsLogger.

**Executing the sample**

1. Wait until the CEP terminal prompts a message similar as follows.

2. Open another terminal, go to `<CEP_HOME>/samples/cep/producers/jms/` directory, and run the following command:

   ```
   ant -DtopicName=topicMap -Dformat=map -Dbroker=activemq -Dsn=0010
   ```

   It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0010/topicMap.csv` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console
Sample 0011 - Receiving JSON, Text, XML Events via JMS Transport - ActiveMQ

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive incoming JSON, Text, XML events adheres WSO2 Event format via jms transport. Here we do not do any processing on the incoming event and to verify the messages we use log event publisher to log the received event.

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Set up the prerequisites required for all samples.
2. Navigate to `<ACTIVEMQ_HOME>/bin/` directory, and execute the following command to start the Apache ActiveMQ server: `./activemq console`

This guide uses ActiveMQ versions 5.7.0 - 5.9.0. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to Apache ActiveMQ Documentation.

3. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
4. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0011. For instructions, see Starting sample CEP configurations. This sample configuration does the following:
Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0011`

- Creates a stream with id `org.wso2.event.sensor.stream:1.0.0`.
- Create and event receivers namely `jmsReceiverJSON`, `jmsReceiverText`, `jmsReceiverXML`.
- Create and event publisher to log the received messages with name `jmsLogger`.

**Executing the sample**

1. Wait until the CEP terminal prompts a message similar as follows.

2. Open another terminal, go to `<CEP_HOME>/samples/cep producers/jms` and run one of the following commands according to the type of event format to be published:

   ```
   ant -DtopicName=topicJSON -Dformat=json -Dbroker=activemq -Dsn=0011
   ant -DtopicName=topicText -Dformat=text -Dbroker=activemq -Dsn=0011
   ant -DtopicName=topicXML -Dformat=xml -Dbroker=activemq -Dsn=0011
   ```

   It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep artifacts/0011/topic Map.csv` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console.

   **for JSON formatted events:**

   ```
   ```

   **for Text formatted events:**

   ```
   ```
Sample 0012 - Receiving Map, XML Events via JMS Transport - Qpid

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive incoming JSON, XML events adheres WSO2 Event format via JMS transport. Here we do not do any processing on the incoming event and to verify the messages we use log event publisher to log the received event.

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install JMS-Qpid Broker and JMS-Qpid Client. Start the JMS-Qpid Broker before running this sample.
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

4. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the register some connection factories section. (the "default" is the name of the virtually hosted node in Qpid)

   ```
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/default?brokerlist='tcp://localhost:5672'
   ```

   By default `jndi.properties` file may contain a connection factory named `TopicConnectionFactory`. You can simply replace it with above to register for Qpid as shown below.

   ```
   # register some connection factories
   # connectionfactory.[jndiname] = [ConnectionURL]
   
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/default?brokerlist='tcp://localhost:5672'
   ```

5. Start JMS-Qpid Broker using command 'bin\qpid-server' from Qpid Broker home.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0012. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0012`
- Creates a stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates three event receivers named `jmsReceiverJSON`, `jmsReceiverText`, and `jmsReceiverXML`.
- Creates an event publisher named `jmsLogger` to log the received messages.

**Executing the sample**

1. Wait until the CEP terminal prompts a message similar as follows.

   ```
   INFO (org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener) - Started to listen on destination : topicMap of type topic for listener JmsReceiverMap\#topicMap
   INFO (org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener) - Started to listen on destination : topicXML of type topic for listener JmsReceiverXML\#topicXML
   ```

2. Open another terminal, go to `<CEP_HOME>/samples/cep/producers/jms` and run one of the following commands according to the type of event format to be published:

   ```
   ant -DtopicName=topicMap -Dformat=map -Dbroker=qpid -Dsn=0012
   ant -DtopicName=topicXML -Dformat=xml -Dbroker=qpid -Dsn=0012
   ```

   It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0012/topicMap.csv` and `topicXML.txt` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console. For Map formatted events:
Sample 0013 - Receiving Map, Text Events via JMS Transport - WSO2 MB

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive incoming Map, Text events adheres WSO2 Event format via jms transport. Here we do not do any processing on the incoming event and to verify the messages we use log event publisher to log the received event.

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Before you configure the WSO2 CEP:
   - Start WSO2 MB 3.1.0
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

4. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the `register some connection factories` section.

```
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/carbon?brokerlist='tcp://localhost:5672'
```

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0013 with an off-port since the WSO2 MB has started in the default port. For instructions, see Starting sample CEP configurations and append `-DportOffset=1` to the command as follows.

```
./wso2cep-samples.sh -sn 0013 -DportOffset=1 -Dqpid.dest_syntax=BURL
```

This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0013`
- Creates a stream with id `org.wso2.event.sensor.stream:1.0.0`
- Create two event receivers named `jmsReceiverMap` and `jmsReceiverText`
- Create and event publisher to log the received messages with name `jmsLogger`
- The `portOffset=1` property, the ports used in CEP server are offset by 1 to avoid port conflicts with MB
- forcing the usage of Binding URL(BURL) address syntax by setting `qpid.dest_syntax` system property to BURL.

**Executing the sample**

1. Wait until the CEP terminal prompts a message similar as follows.

```
[2015-05-18 02:00:50,649] INFO org.wso2.carbon.event.input.adapter.jms.InternalUtil.JMSListener - Started to listen on destination: topicMap of type topic for listener jmsReceiverMap
[2015-05-18 02:00:50,649] INFO org.wso2.carbon.event.input.adapter.jms.InternalUtil.JMSListener - Started to listen on destination: topicText of type topic for listener jmsReceiverText
```

2. Open another terminal and navigate to the `<CEP_HOME>/samples/cep/producers/jms` directory.

and run one of the following commands according to the type of event format to be published:

```
ant -DtopicName=topicMap -Dformat=map -Dbroker=mb -Dsn=0013
ant -DtopicName=topicText -Dformat=text -Dbroker=mb -Dsn=0013
```

It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0013/topicMap.csv` and `topicText.txt` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console. for Map formatted events:
3. Text formatted events:

Sample 0014 - Receiving XML Events via Soap Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to receive incoming XML events that adheres to the WSO2 Event format via SOAP transport. This sample does not do any processing on the incoming event. The log event publisher is used to log the received events, and to verify the messages.

**Prerequisites**
Set up the prerequisites required for all samples

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0014**. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- A stream with the ID `org.wso2.event.sensor.stream:1.0.0`
- An event receiver named `soapReceiver`
- An event publisher named `soapLogger` to log the received messages

**Executing the sample**

Navigate to the `<CEP_HOME>/samples/cep/producers/soap/` directory, and execute the following Ant command using another tab in the CLI:

```
ant -Durl=http://localhost:9763/services/soapReceiver/receive -Dsn=0014
```

The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/soap/build.xml` file.

Sample 0015 - Receiving Text Events via Email Transport

- Prerequisites
- Building the sample
- Executing the sample

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0015**. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- An event stream with the ID `org.wso2.event.sensor.stream_1.0.0`
• An event receiver named `emailReceiver` to fetch events from the configured receiver email address
• An event publisher named `emailEventLogger` to log the received messages

**Executing the sample**

Send an email to `cep2015test@gmail.com` address, with the subject of it as `cep`. Enter the below content as the body of the email in plain text format.

```
meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:601,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
```

You view the CEP server receiving the email events in the logs of it in the CLI as shown below.

**Sample 0016 - Receiving JSON Events via MQTT Transport**

• **Prerequisites**
• **Building the sample**
• **Executing the sample**

**Prerequisites**

Follow the steps below before starting this MQTT sample configurations.

1. Set up the prerequisites required for all samples
2. Configure WSO2 CEP by adding relevant jars to support MQTT transport.
3. Configure sample client by adding relevant jars. See setting up MQTT for MQTT sample clients.
4. Start the MQTT-supported server. (E.g. Mosquitto)

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0016**. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

• An event stream with the ID `org.wso2.event.sensor.stream`
• An event receiver named `mqttEventReceiver` to fetch events from the configured MQTT server
• An event publisher named `loggerEventPublisher` to log the received messages

**Executing the sample**
After you view the message on successful connection to the MQTT-supported server on the WSO2 CEP logs in the CLI, navigate to the `<CEP_HOME>/samples/cep/producers/mqtt/` directory, and execute the following Ant command using another tab in the CLI:

```
ant -Durl=tcp://localhost:1883 -Dtopic=sensordata -Dsn=0016
```

Replace the value of the `Durl` parameter in the above command with your MQTT broker URL accordingly. The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/mqtt/build.xml` file.

This builds the MQTT client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0016/sensordata.txt` file to the `mqttEventReceiver` endpoint. You view the details of the events that are sent as shown below.
You view the CEP server receiving the JSON events in the logs of it in the CLI as shown below.
Sample 0017 - Receiving Custom Text Events via File Tail Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to receive custom incoming text events via file tail transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the incoming event. The log event publisher is used to log the received events, and to verify the messages.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0017. For instructions, see Starting sample CEP configurations. This sample configuration creates the following:

- A stream named org.wso2.event.sensor.stream:1.0.0.
- An event receiver named fileReceiver.
- An event publisher named httpLogger to log the received messages.

Executing the sample

Open another terminal console window and navigate to the <CEP_HOME>/samples/cep/artifacts/0017 directory. Then execute following command to write data to the file.
echo
"timestamp:19900813115534,isPowerSaverEnabled:false,sensorId:103,sensorName:temperature,longitude:20.44345,latitude:5.443435,humidity:8.9,sensorValue:1.23434" >> fileReceiver.txt

CEP console will show the received event as following screenshot.

Sample 0018 - Receiving JSON Events via Kafka Transport

- Prerequisites
- Building the sample
- Executing the sample

**Prerequisites**

Follow the steps below before starting this Kafka sample configurations.

1. Set up the prerequisites required for all samples.
2. Configure WSO2 CEP by adding relevant jars to support Kafka transport.
3. Configure sample client by adding relevant jars. See setting up Kafka for Kafka sample clients
4. Start the Apache ZooKeeper server with the following command:  
   ```sh
   bin/zookeeper-server-start.sh config/zookeeper.properties.
   ```
   You view the below logs. For more information, see Apache Kafka documentation.

5. Then start the Kafka server with the following command:  
   ```sh
   bin/kafka-server-start.sh config/server.properties.
   ```
   You view the below logs.
5. Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0018. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- An event stream with the ID org.wso2.event.sensor.stream:1.0.0
- An event receiver named kafkaReceiver to fetch events from the configured receiver email address
- An event publisher named loggerEventPublisher to log the received messages

Executing the sample

Navigate to the `<CEP_HOME>/samples/cep/producers/kafka/` directory, and execute the following Ant command using another tab in the CLI:

```
ant -Durl=localhost:9092 -Dtopic=sensordata -Dsn=0018
```

This builds the Kafka client and publishes the JSON events in the `<CEP_HOME>/samples/cep/artifacts/0018/sensordata.txt` file to the kafkaReceiver endpoint. You can view the details of the events that are sent as shown below.
You view the CEP server receiving the JSON events in the logs of it in the CLI as shown below.

```
```

Sample 0019 - Receiving JSON Events via WebSocket Transport

- Prerequisites
- Building the sample
- Executing the sample
**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0019**. For instructions, see Starting sample CEP configurations. This sample configuration creates the following:

- An event stream with the ID `org.wso2.event.sensor.stream:1.0.0`
- An event receiver named `wsReceiver` to fetch events from the configured receiver email address
- An event publisher named `loggerEventPublisher` to log the received messages

**Executing the sample**

1. Navigate to the `<CEP_HOME>/samples/cep/producers/websocket/` directory, and execute the following Ant command using another tab in the CLI: `ant -Dport=9099 -Dsn=0019`

```
The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/websocket/build.xml` file.
```

This builds the WebSocket client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0019/sensorStreamEvents.txt` file to the `wsReceiver` endpoint. You view the details of the events that are sent as shown below.
2. Start the WSO2 CEP server with the sample configuration numbered 0019. For instructions, see Starting sample CEP configurations.

The CEP server receiving the JSON events is logged in the CLI as shown below.
Sample 0020 - Simple JSON Pass-through with Websocket-Local Input Event Adapter

- **Introduction**
- **Prerequisites**
- **Starting sample CEP configurations**
- **Connect to CEP via a Websocket client**

**Introduction**

This sample demonstrates how the WSO2 CEP can receive (and then process if required) events from a Websocket client.

Refer to below image to get an overview of the event flow.

Refer to the below steps, which we are going to do in this sample:

- Step 1: Start CEP server. This will start a Websocket server which is in-built to CEP.
- Step 2: Define an Event Stream in CEP.
- Step 3: Create an Input Websocket-local Event Adapter, so CEP can receive events of type defined in step 1.
- Step 4: Create an Output Logger Event Adapter, so whenever CEP receives an event, it will be logged on the console on which CEP server runs.
- Step 5: Now we can connect to the CEP-websocket server and publish events to it, using a Websocket client. As we publish messages, those messages will be logged in the CEP console.
Note that, for simplicity, we do not set up CEP to process the events received by the Websocket-local Input Adapter. Instead of processing events using execution plans, those will be directly handed over to the Output Logger Adapter for logging. This is indicated in the above diagram by making the ‘Event Processor’ component almost transparent: to show that it does not do any processing.

**Prerequisites**

Set up the prerequisites required for all samples.

In addition to that, go to `<CEP_HOME>/samples/cep/utils/input-websocket-local-adapter` directory and execute the following command:

```
ant -Dsn=0020
```

This will copy `inputwebsocket.war` webapp to `<CEP_HOME>/sample/cep/artifacts/0020/webapps` folder.

**Starting sample CEP configurations**

Start the WSO2 CEP server with the sample configuration numbered 0020. For instructions, see Starting sample CEP configurations.

This sample configuration points the default Axis2 repo to `<CEP_HOME>/sample/cep/artifacts/0020` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`). As a result, the artifacts in the `<CEP_HOME>/sample/cep/artifacts/0020` directory are deployed.

When these sample configurations are deployed:

- An event stream will be defined.
- An Input Websocket-local Event Adapter will be created.
- An Output Logger Event Adapter will be created.

In other words, all the first four steps listed in Introduction section will be done, when the server is started with sample configuration number 0020.

Now the remaining task is connect to the CEP-websocket server and publish events to it.

**Connect to CEP via a Websocket client**

In this sample, you will use your web browser as the Websocket client. However, any Websocket client can be used for this purpose.

**Step I:** Start a web browser and go to its Javascript console. In most browsers, such as Chrome and Firefox, you can load the Javascript console by simply pressing the keys Ctrl+Shift+J (or Cmd+Option+J on a Mac).

Ensure that the page on which you open the Javascript Console is loaded over HTTP. For example, you can open the page `http://wso2.com` and load the Javascript console on that page.

**Step II:** On the browser console, type:
var ws = new WebSocket("ws://localhost:9763/inputwebsocket/wsLocalInputAdapter");

Note: If you have started CEP server on a different host and a port, replace 'localhost' in the above command with that host, and '9763' with that port, respectively. See Running the Product page for more details.

This step is shown in the screenshot below.

Upon successful connection, you will see an output in the same console as shown in the below screenshot.

Now we’re ready to publish events to the CEP.

**Step III:** Next and final step is to publish three events to the CEP. Following are the three events which we will publish to CEP:
To publish the above three events to the CEP, we will use the following three commands, each of which will publish an event to CEP.

Copy and paste each command on the browser console and press enter.
As you publish events to the CEP using the web browser client, check the CEP console. It will print the logs, indicating that it received the events sent by the browser. This is shown in the screenshot below:
Sample 0021 - Receiving Map Events via JMS Transport - ActiveMQ (For Queue)

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive incoming Map events adheres WSO2 Event format via jms transport using JMS Queue. Here we do not do any processing on the incoming event and to verify the messages we use log event publisher to log the received event.

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Start Apache ActiveMQ server.

   This guide uses ActiveMQ versions 5.7.0 - 5.9.0. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to Apache ActiveMQ Documentation.

2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Configure sample client by adding relevant jars. See Setting up JMS for JMS sample clients.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0021. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0021`
- Creates a stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event receiver named `jmsReceiverMap`.
- Creates an event publisher named `jmsLogger` to log the received messages.

**Executing the sample**

1. Wait until the CEP terminal prompts a message similar as follows.
2. Open another terminal, go to `<CEP_HOME>/samples/cep/producers/jms` and run one of the following commands according to the type of event format to be published:

```
ant -DqueueName=queueMap -Dformat=csv -Dbroker=activemq -Dsn=0021
```

It builds the jms client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0021/queueMap.csv` to the jms receiver endpoint.

3. You can see the events getting received by CEP by the logs in its console, for Map formatted events:

```
meta_timestamp:19990813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:4683,
meta_sensorName:temperature,
correlation_longitude:98.343444,
correlation_latitude:28.44345,
humidity:2.3,
sensorValue:28.44345
```

Sample 0022 - Receiving Custom RegEx Text Events via File Tail

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to receive custom messages with common CSV data standard and how to map the values for the specified event stream. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see [Event Formats](#). Also, there are some of the Regular Expressions that are used to classify the text which is logged by the logger publisher. This sample does not do any processing on the incoming event. The log event publisher is used to log the received events which read from a CSV file using the file tail receiver.

Common RegEx to clarify inputs and outputs in WSO2 CEP

1. Receiving the CLI logging as events
1. timestamp: \((.+)\)

2. information type: \}\s+(\w*)\s+
3. class name: \{(.+}\)
4. information: } \s* \s*(.*)

2. Receiving events from a standard CSV file (data is separated by a comma and events are separated by a new line character)
   1. first element: \([\^,]+\)
   2. second element: \([\^,]+\),\([\^,]+\)
   3. third element: \([\^,]+\),\([\^,]+\),\([\^,]+\)

**Prerequisites**

Set up the prerequisites required for all samples

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0022. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- A stream named org.wso2.event.sensor.stream:1.0.0.
- An event receiver named fileTailerReceiver.
- An event publisher named fileLogger to log the messages.

**Executing the sample**

Navigate to the `<CEP_HOME>/samples/cep/artifacts/0022/fileReceiver.csv` file, and append several lines in similar format as the first line which is already entered.

```
1 19900813115534,false,103,temperature,20.44345,5.443435,8.9,1.23434
2 19900813115534,false,103,temperature,20.44345,5.443435,8.9,1.23444
```

You can use a command similar to below to add the lines to the fileReceiver.csv. Please do not edit the file directly and save as this will not work when tailing the file.

```
echo '19900813115834,false,110,temperature,20.4435,5.43435,8.9,1.23434' >> fileReceiver.csv
```

The file tail receiver reads from the CSV file as it gets updated and the input values mapped by the receiver are logged by the logger.

You view the CEP server receiving the events in the logs of it in the CLI as shown below.
Samples on Processing Events

- Sample 0101 - Pass-Through/Projection Query in an Execution Plan
- Sample 0102 - Projections, Transformations and Enrichment for events
- Sample 0103 - Using filters for generating alerts
- Sample 0104 - Calculations over time using Windows
- Sample 0105 - Performing Joins with windows
- Sample 0106 - Using in-memory event tables
- Sample 0107 - Using RDBMS event tables
- Sample 0108 - Using patterns to detect ATM transaction frauds
- Sample 0109 - Detecting trends with sequences
- Sample 0110 - Sequences with partitioning to detect trends
- Sample 0111 - Detecting non-occurrences with Patterns
- Sample 0112 - Analyzing Twitter Feeds Using Partitions
- Sample 0113 - Limiting the Output Rate of an Event Stream
- Sample 0114 - Using External Time Windows
- Sample 0115 - Quartz scheduler based alerts
- Sample 0116 - Performing Linear Regression
- Sample 0117 - Filtering and Outputting to Multiple Streams
- Sample 0118 - Using Hazelcast Event Tables
- Sample 0119 - Trigger Events at Defined Time Intervals
- Sample 0120 - Using the Map Extension
- Sample 0121 - Using Event Windows
- Sample 0301 - Network Intruder Detection with PMML Extension Predictions
- Sample 0501 - Processing a Simple Filter Query with Apache Storm Deployment
- Sample 0502 - Processing a Window State Over Server Restart
- Sample 0503 - Processing a Window Query in High Availability Mode
- Sample 0504 - Processing a Distributed Siddhi Query with Partitioning by integrating with Apache Storm
- Sample 1001 - WSO2 CEP Geo Dashboard
- Sample 1501 - Viewing Real Time Analytics

Sample 0101 - Pass-Through/Projection Query in an Execution Plan

- Introduction
- Prerequisites
- Building the sample
- Executing the sample
**Introduction**

This sample demonstrates how to set up an execution plan with a basic pass-through/projection query. It selects some of the attributes of each incoming event and emits an output event with the selected attributes. This sample uses `wso2event` for both inputs and outputs.

The query used in this sample is as follows:

```sql
from sensor_stream
select meta_sensorId, correlation_longitude, correlation_latitude, humidity, sensorValue as value
insert into sensor_value_projected_stream;
```

The above query,

- Processes the events received through the `sensor_stream`
- Selects the attributes (meta_sensorId, correlation_longitude, correlation_latitude, humidity, sensorValue) specified under the `select` clause, from each event received.
- When selecting the attributes, it also renames the `sensorValue` attribute name as `value`.
- Emits those events as output events through the `sensor_value_projected_stream`.

Since we do not perform any processing apart from selecting a few attributes, this is considered as a pass-through query.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0101. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0101` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Open a terminal, go to `<CEP_HOME>/samples/cep/producers/wso2-event` and run the following command from there.

   ```
   ant -DstreamId=org.wso2.event.sensor.stream:1.0.0 -Dsn=0101
   ```

   It builds and runs the `wso2event` producer, which will send sensor data to the CEP server.

2. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.
For example, given below is a screenshot of the output of the consumer sending events from the producer:

![Screenshot of output](image.png)

**Sample 0102 - Projections, Transformations and Enrichment for events**

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to set up an execution plan with some basic queries that perform projections and transformations of attributes and enriching events with new attributes. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The execution plan used in this sample are as follows:
@Import('TempStream:1.0.0')
define stream TempStream (deviceID long, roomNo int, temp double);

@Export('TransformedRoomTempStream:1.0.0')
define stream TransformedRoomTempStream (uuid string, temp double, scale string);

@Export('EnrichedRoomTempStream:1.0.0')
define stream EnrichedRoomTempStream (roomNo int, temp double, scale string);

from TempStream
select roomNo, temp
insert into RoomTempStream;

from RoomTempStream
select roomNo, temp, 'C' as scale
insert into EnrichedRoomTempStream;

from TempStream
select str:concat(roomNo, '-', deviceID) as uuid, (temp * 1.8000 + 32.00) as temp, 'F' as scale
insert into TransformedRoomTempStream;

The first query,
- Processes the events received through the TempStream
- Selects the attributes (roomNo, temp) specified under the select clause, from each event received.
- Emits those events as output events through the RoomTemp stream.

The second query,
- Processes the events received through the RoomTempStream (which is the output of the previous query)
- Selects the attributes (roomNo, temp) specified under the select clause, from each event received. It also adds a new attribute named 'scale', with a constant value that it 'C'.
- Emits those events as output events through the EnrichedRoomTempStream stream.

The third query,
- Processes the events received through the TempStream.
- When selecting the attributes, concatenate roomNo and deviceld with the inbuilt concat function and names it as uuid using the 'as' keyword, converts the temp attributes to Farenheit scale (from the incoming Celcius scale), and then adds a new attribute named 'scalw' with the value 'F' under the select clause, from each event received.
- Emits those events as output events through the TransformedRoomTempStream.

Prerequisites
Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0102. For instructions, see the Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0102` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Log into the CEP management console which is located at `https://localhost:9443/carbon`.
2. Go to Tools -> Event Simulator. Under the 'Multiple Events' section, you can see the listed `events.csv` file which contains some sample data. Click Play to start sending sample events from the file. Here, you can perform other operations such as Pause, Stop and Resume if required.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

For example, given below is a screenshot of the output of the consumer sending events from the producer:

![Screenshot of CEP console output](image)

**Sample 0103 - Using filters for generating alerts**

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to set up an execution plan with a filter query that checks for some specific conditions and then outputs a custom alert event when the conditions are true. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample uses the Event Simulator for inputs and the logger publisher for logging the outputs to the CEP console.
The execution plan used in this sample is as follows:

```sql
from TempStream [ roomNo > 245 and roomNo <= 365 and temp > 40 ]
select roomNo, temp
insert into AlertServerRoomTempStream ;
```

The first query,

- Processes the events received through the TempStream
- Inside the filter, checks for the condition `roomNo > 245 and roomNo <= 365 and temp > 40`
- If the condition is true,
  - Selects the attributes (roomNo, temp) specified under the `select` clause, from each event received.
  - Emits those events as output events through the AlertServerRoomTempStream stream.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0103. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0103` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Log into the CEP management console which is located at https://localhost:9443/carbon.
2. Go to Tools -> Event Simulator. Under the 'Multiple Events' section, you can see the listed 'events.csv' file which contains some sample data. Click Play to start sending sample events from the file.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

For example, given below is a screenshot of the output of the consumer sending events from the producer:

Sample 0104 - Calculations over time using Windows

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to set up an execution plan to perform calculations over time by aggregating events.
The queries use time windows and time batch windows to aggregate event over time. This sample uses the Event Simulator for inputs and the logger publisher for logging the custom output events to the CEP console. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats.

The queries used in the `WindowBasedAvgTemp` execution plan used in this sample are as follows:

```sql
-- with time sliding window of 1 min
from TempStream#window.time(1 min)
select roomNo, avg(temp) as avgTemp
group by roomNo
insert all events into AvgRoomTempStream ;

-- with time batch (tumbling) window of 1 min
from TempStream#window.timeBatch(1 min)
select roomNo, avg(temp) as avgTemp
group by roomNo
insert all events into AvgRoomTempPerMinStream ;
```

The first query,

- Processes the events received through the TempStream
- Defines a sliding time window of 1 minute, that keeps each arriving event for 1 minute
- Selects the attributes roomNo, avg(temp) from the events stored in the time window. Due to the group by clause used here, the average is calculated per roomNo here. The average of the temp values is named as avgTemp here.
- The all events clause in the insert statement makes the query to be triggered by both current events and expired events (a current event is the incoming events to the window. an expired event is an event emitted by the window after keeping it for 1 minute in this case)
- Emits those events as output events through the AvgRoomTempStream stream.
- Mathematically, this query calculates the moving average of the room temperature for each room and gives instantaneous results upon arrival/expiration of each incoming event.

The second query,

- Processes the events received through the TempStream
- Defines a time batch window of 1 minute, that keeps all incoming events and emits events periodically every 1 minute.
- Selects the attributes roomNo, avg(temp) from the events stored in the time window. Due to the group by clause used here, the average is calculated per roomNo here. The average of the temp values is named as avgTemp here.
- The all events clause in the insert statement makes the query to be triggered by both current events and expired events (a current event is the incoming events to the window. an expired event is an event emitted by the window after keeping it for 1 minute in this case)
- Emits those events as output events through the AvgRoomTempPerMinStream stream.
- Similar to the first query, this also calculates a moving average of the temperature for each room, but emits them periodically every 1 minute.

There's another execution plan that continuously calculate the average temperature from the beginning which is as follows:
from TempStream
select roomNo, avg(temp) as avgTemp
insert into AvgTempFromStartStream;

The third query,

- Processes the events received through the TempStream.
- When selecting the attributes, concatenate roomNo and avg(temp) renamed as avgTemp which is the average of the temperature for each room from the start.
- Emits those events as output events through the AvgTempFromStartStream.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0104. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to <CEP_HOME>/samples/cep/artifacts/0104 (by default, the Axis2 repo is <CEP_HOME>/repository/deployment/server).

**Executing the sample**

1. Log into the CEP management console which is located at https://localhost:9443/carbon.

2. Go to Tools -> Event Simulator. Under the Multiple Events section, you can see the listed events.csv file that contains some sample data. Click Play to start sending sample events from the file.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console. Since this execution plan uses 1 minute time windows, observe the results for a few minutes to get all results from different queries.

For example, given below is a screenshot of the output of the consumer sending events from the producer:
Sample 0105 - Performing Joins with windows

Introduction
Prerequisites
Building the sample
Executing the sample

Introduction

This sample demonstrates how to set up an execution plan with time windows and length windows and perform a join (with a condition) to trigger an output. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The execution plan used in this sample are as follows:

```
from TempStream[temp > 30.0]#window.time(1 min) as T
    join RegulatorStream[isOn == false]#window.length(1) as R
    on T.roomNo == R.roomNo
select T.roomNo, T.temp, R.deviceID, 'start' as action
insert into RegulatorActionStream;
```

The first query,

- Processes the events received through the TempStream, filters events with temp > 30.0 and inserts the events that satisfy the filter condition into a 1-minute time window (renamed as T).
- Filters events coming through the RegulatorStream with the condition isOn == false, and inserts the events that satisfy the condition into a length window of 1 event (renamed as R).
- Performs a join between the above two windows, with the condition T.roomNo == R.roomNo, this will create tuples consisting of the events matching this condition as in SQL.
- Selects the roomNo from T events, temp from T events, deviceID from R, and also inserts and additional attribute named 'action' with the value 'start'.
- Emits those events as output events through the RegulatorActionStream stream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0105. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to <CEP_HOME>/samples/cep/artifacts/0105 (by default, the Axis2 repo is <CEP_HOME>/repository/deployment/server).

Executing the sample

1. Log into the CEP management console which is located at https://localhost:9443/carbon.
2. Go to Tools -> Event Simulator. The following files are displayed under the Send multiple events section.
   - events.csv: This file contains sample events to simulate the event flow of the event stream named TempStream:1.0.0.
   - regularEvents.csv: This file contains sample events to simulate the event flow of the event stream
Sample 0106 - Using in-memory event tables

Introduction

This sample demonstrates how to set up an execution plan to filter out credit card transactions that makes use of an in-memory event table to identify blacklisted transactions. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The execution plan used in this sample are as follows:

```
define table CardUserTable (name string, cardNum string, blacklisted bool) ;
```

Given above is the table definition,

- Defines a table named CardUserTable with the given attributes. This is by default an in-memory table.

```
from CardUserStream
select *
insert into CardUserTable;
```

The first query,

- Processes the events received through the CardUserStream.
- Selects all the attributes under the `select` clause, from each event received.
- Inserts it to the CardUserTable.

```
from BlackListStream
select cardNo as cardNum, true as blacklisted
update CardUserTable
on cardNum == CardUserTable.cardNum;
```
The second query,

- Processes the events received through the BlackListStream.
- Selects cardNo and renames it as cardNum, introduces a new attribute named blacklisted with the value 'true' under the select clause, for each event received.
- Updates the CardUserTable with the condition cardNum == CardUserTable.cardNum. Here the blacklisted attribute in the table will be updated with the new value.

```
from PurchaseStream#window.length(1) as p join CardUserTable as c
  on p.cardNo == c.cardNum and c.blacklisted == false
select p.cardNo as cardNo, c.name as name, p.price as price
insert into WhiteListPurchaseStream ;
```

The third query,

- Defines a length window that keeps 1 event of the input stream PurchaseStream.
- Joins it with the CardUserTable with the condition p.cardNo == c.cardNum and c.blacklisted == false. In this condition, the events with blacklisted == true in the table gets filtered out and then the remaining events will be joined based on the card number.
- Emits those events as output events through the WhiteListPurchaseStream.

```
from DeleteAllUsers
delete CardUserTable
  on deleteAll == true;
```

The last query is used to clean up the table from an external trigger event through DeleteAllUsers stream,

- It processes the events received through the DeleteAllUsers.
- Checks for the condition deleteAll == true and if it’s true, deletes all the records in the CardUserTable.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0106. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to <CEP_HOME>/samples/cep/artifacts/0106 (by default, the Axis2 repo is <CEP_HOME>/repository/deployment/server).

Executing the sample

1. Log into the CEP management console which is located at https://localhost:9443/carbon.

2. Go to Tools -> Event Simulator. Under the Multiple Events section, you can see 4 files listed there which contains some sample data as follows.
3. The userEvents.csv file contains sample data that is used to fill the in-memory CardUserTable. Click play to start sending events to fill the table.

4. The blacklistUserEvents.csv file contains sample data that is used to mark user entries in CardUserTable as blacklisted. Click Play to start sending blacklisted events and mark some table entries as blacklisted.

5. The purchaseEvents.csv contains credit card transactions data. Play it and send the transaction data.

6. After sending sample events from purchaseEvents.csv, you will be able to see the outputs as follows.

Sample 0107 - Using RDBMS event tables

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to set up an execution plan to filter out credit card transactions that makes use of an in-memory event table to identify blacklisted transactions. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The execution plan used in this sample are as follows:

```java
@from(eventtable = 'rdbms' , datasource.name = 'WSO2_CARBON_DB' ,
      table.name = 'CEPSample0107CardUserTable')
define table CardUserTable (name string, cardNum string, blacklisted bool)
```

Given above is the table definition,

- Defines a table named CardUserTable with the given attributes.
- The annotation @from is used to link the table to an RDBMS event table named CEPSample0107CardUserTable and the data source used to access the table as WSO2_CARBON_DB (this is the datasource of the...
default H2 database that is shipped with WSO2 CEP. If needed you can define a separate data source pointing to some other DB such as MySQL and use it here.)

```sql
from CardUserStream
select *
insert into CardUserTable;
```

The first query,

- Processes the events received through the CardUserStream
- Selects all the attributes under the select clause, from each event received.
- Inserts it to the CardUserTable.

```sql
from BlackListStream
select cardNo as cardNum, true as blacklisted
update CardUserTable
on cardNum == CardUserTable.cardNum;
```

The second query,

- Processes the events received through the BlackListStream.
- Selects cardNo and renames it as cardNum, introduces a new attribute named blacklisted with the value true under the select clause, for each event received.
- Updates the CardUserTable with the condition cardNum == CardUserTable.cardNum. Here the blacklisted attribute in the table will be updated with the new value.

```sql
from PurchaseStream#window.length(1) as p join CardUserTable as c
on p.cardNo == c.cardNum and c.blacklisted == false
select p.cardNo as cardNo, c.name as name, p.price as price
insert into WhiteListPurchaseStream ;
```

The third query,

- Defines a length window that keeps 1 event of the input stream PurchaseStream.
- Joins it with the CardUserTable with the condition p.cardNo == c.cardNum and c.blacklisted == false. In this condition, the events with blacklisted == true in the table gets filtered out and then the remaining events are joined based on the card number.
- Emits those events as output events through the TransformedRoomTempStream.

```sql
from DeleteAllUsers
delete CardUserTable
on deleteAll == true;
```

The last query is used to clean up the table from an external trigger event through DeleteAllUsers stream,

- It processes the events received through the DeleteAllUsers.
- Checks for the condition deleteAll == true and if its true, deletes all the records in the CardUserTable.
Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0107. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0107` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

Executing the sample

1. Log in to the CEP management console using the following URL and admin/admin credentials: `https://localhost:9443/carbon`.
2. Click Tools, and then click Event Simulator. You view four files listed under the Send multiple Events section, which contains some sample data as follows.

<table>
<thead>
<tr>
<th>File</th>
<th>Stream Configuration</th>
<th>Delay between events (ms)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteUserEvents.csv</td>
<td>DeleteAllUsers:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>purchaseEvents.csv</td>
<td>PurchaseStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>userEvents.csv</td>
<td>CardUserStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>blacklistUserEvents.csv</td>
<td>BlackListStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
</tbody>
</table>

   3. Click the corresponding Play button of the userEvents.csv file which contains sample data that is used to fill the in-memory CardUserTable. This starts sending events to fill the table.

   4. Click the corresponding Play button of the blacklistUserEvents.csv file which contains sample data that is used to mark user entries in the CardUserTable as blacklisted. This starts sending blacklisted events and marks some table entries as blacklisted.

   5. Click the corresponding Play button of the purchaseEvents.csv file which contains credit card transactions data. This sends the transaction data.

   6. Click the corresponding Play button of the deleteUserEvents.csv file to clean up the tables. You view the output logs in the CLI in which you run the products as shown below.

Sample 0108 - Using patterns to detect ATM transaction frauds

- Introduction
- Prerequisites
Building the sample

Executing the sample

Introduction

This sample demonstrates how to set up an execution plan with a pattern query that can be used to detect credit card/ATM transaction frauds from transaction events received. This sample uses the Event Simulator for inputs and the logger publisher for logging the custom output events to the CEP console. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats.

The query used in this sample is as follows:

```sql
from every a1 = atmStatsStream[amountWithdrawed < 100] -> b1 = atmStatsStream[amountWithdrawed > 10000 and a1.cardNo == b1.cardNo] within 1 day
select a1.cardNo as cardNo, a1.cardHolderName as cardHolderName, b1.amountWithdrawed as amountWithdrawed, b1.location as location, b1.cardHolderMobile as cardHolderMobile
insert into possibleFraudStream;
```

The above query is based on the assumption that a ATM card thief will first test the stolen card with a small transaction and then within a short period will try to make large payments with the card. The query:

- Processes the events received through the atmStatsStream.
- First looks for an event a1 where the amountWithdrawed is less than 100.
- Next it looks for another event b1 where the card number is the same as previous and amountWithdrawed is greater than 10000. The within keyword ensures that this condition should be satisfied within a day.
- When the pattern condition is satisfied, select the attribute and renames them as:
  a1.cardNo as cardNo, a1.cardHolderName as cardHolderName, b1.amountWithdrawed as amountWithdrawed, b1.location as location, b1.cardHolderMobile as cardHolderMobile
- Inserts the output events to the possibleFraudStream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0108. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0108` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the CEP management console which is located at `https://localhost:9443/carbon`.
2. Go to Tools -> Event Simulator. Under the Multiple Events section, you can see the `events.csv` file. This file contains some sample data. Click Play to start sending sample events from the file.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output.
For example, given below is a screenshot of the output of the consumer sending events from the producer:

Sample 0109 - Detecting trends with sequences

Introduction

This sample demonstrates how to set up an execution plan with a sequence-based query that can be used to detect trends from a stock trades stream. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The query used in this sample is as follows:

```
from every e1=FilteredStockStream[price>20],
  e2=FilteredStockStream[((e2[last].price is null) and price>=e1.price) or
  ((not (e2[last].price is null)) and price>=e2[last].price)]+,
  e3=FilteredStockStream[price<e2[last].price]
select e1.price as priceInitial, e2[last].price as pricePeak, e3.price as priceAfterPeak
insert into PeakStream ;
```

Above query:

- Processes the events received through the FilteredStockStream.
- First it looks for an event e1 with the condition price greater than 20.
- Then it looks for one or more events, e2 with a condition:
  - \((e2[last].price is null) and price>=e1.price\) is used to check the first event after e1, i.e. \((e2[last].price is null)\) returns true since there is no last event in e2. Then it checks for the condition where the current event price is greater than e1 price.
  - \((not (e2[last].price is null)) and price>=e2[last].price\) this part is for any subsequent events. In this case the last of e2 is not null, and we check whether the price of the current event is greater than the last event. i.e. we are looking for one or more events with continuous price increase from this whole condition.
- Then we look for another event with a price drop from the last e2 event from the condition e3=FilteredStockStream[price<e2[last].price].
- From the select clause we select the attributes e1.price as priceInitial, e2[last].price as pricePeak, e3.price as priceAfterPeak.
- Finally the event is output to the PeakStream.
**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0109. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0109` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Log into the CEP management console which is located at `https://localhost:9443/carbon`.
2. Go to Tools -> Event Simulator. Under the Multiple Events section, the `events.csv` file is listed. This file contains some sample data. Click Play to start sending sample events from the file.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

For example, given below is a screenshot of the output of the consumer sending events from the producer:

![Screenshot of output](image)

**Sample 0110 - Sequences with partitioning to detect trends**

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to set up an execution plan with a sequence-based query that can be used to detect trends from a stock trades stream. This sample uses the Event Simulator for inputs and the logger publisher for logging the custom output events to the CEP console. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats.

The query used in this sample is as follows:

```plaintext
partition with (symbol of StockStream)
begin
    from every e1=StockStream[price>20],
        e2=StockStream[((e2[last].price is null) and price>=e1.price) or ((not (e2[last].price is null)) and price>=e2[last].price)],
        e3=StockStream[price<e2[last].price]
    select e1.symbol as symbol, e1.price as priceInitial, e2[last].price as pricePeak, e3.price as priceAfterPeak
    insert into PeakStream;
end;
```
Above query:

- A partition is defined with the symbol attribute of the StockStream, which means that the processing will take place independently for events of each symbol.
- Processes the events received through the FilteredStockStream.
- First it looks for an event e1 with the condition price greater than 20.
- Then it looks for one or more events, e2 with a condition:
  - ((e2[last].price is null) and price>=e1.price) is used to check the first event after e1, i.e. (e2[last].price is null) returns true since there is no last event in e2. Then it checks for the condition where the current event price is greater than e1 price.

  - ((not (e2[last].price is null)) and price>=e2[last].price) this part is for any subsequent events. In this case the last of e2 is not null, and we check whether the price of the current event is greater than the last event. i.e. we are looking for one or more events with continuous price increase from this whole condition.

- Then we look for another event with a price drop from the last e2 event from the condition e3=FilteredStockStream[price<e2[last].price].
- From the select clause we select the attributes e1.price as priceInitial, e2[last].price as pricePeak, e3.price as priceAfterPeak.
- Finally the event is output to the PeakStream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0110. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to <CEP_HOME>/samples/cep/artifacts/0110 (by default, the Axis2 repo is <CEP_HOME>/repository/deployment/server).

Executing the sample

1. Log into the CEP management console which is located at https://localhost:9443/carbon.
2. Go to Tools -> Event Simulator. Under the Multiple Events section, the events.csv file is listed. This file contains some sample data. Click Play to start sending sample events from the file.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

For example, given below is a screenshot of the output of the consumer sending events from the producer:

Sample 0111 - Detecting non-occurrences with Patterns

Introduction
**Introduction**

This sample demonstrates how to set up an execution plan with a pattern matching siddhi query to detect non-occurrences. In this sample we use patterns to detect non-delivered packages of a courier service within a predefined time period (for demonstration purposes we assume the time limit to deliver a package is 2 minutes here).

The queries used in this sample is as follows.

```sql
from arrivals_stream#window.time(2 minutes)
select *
insert expired events into overdue_deliveries_stream;

from every arrivalEvent = arrivals_stream ->
  deliveryEvent = deliveries_stream[arrivalEvent.trackingId == trackingId]
  or overdue_delivery = overdue_deliveries_stream[arrivalEvent.trackingId == trackingId]
select arrivalEvent.trackingId as trackingId, arrivalEvent.customerName as customerName, arrivalEvent.telephoneNo as telephoneNo,
  deliveryEvent.trackingId as deliveryId
insert into filter_stream;

from filter_stream [ (deliveryId is null)]
select trackingId, customerName, telephoneNo
insert into alert_stream;
```

The first query keeps each arrival event for 2 minutes and then emits it to overdue_deliveries_stream which is then used in the second query.

The second query uses a pattern. It looks for a pattern where, there an arrival event (through the arrivals_stream) which is followed by a delivery event (through deliveries_stream) or else an overdue delivery event (this is an arrival event which is 2 minutes old, coming through the overdue_deliveries_stream)

Now, if the pattern detects an arrival event followed by an overdue delivery event with no delivery event, that means the package is not delivered within the 2 minute time limit.

To detect whether the delivery event is null, we use the third query (deliveryId is null) where the deliveryId will be null if the delivery event is null.

**Prerequisites**

- See Prerequisites in CEP Samples Setup page for generic prerequisites. (copy apache-axiom.jar to <CEP_HOME>/samples/cep/lib directory to send XML event types)

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0111. For instructions, see Starting sample
CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0111` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Open a new terminal, go to `<CEP_HOME>/samples/cep/consumers/generic-log-service` and run the command below. It builds the sample log service web app and deploys in the webapps repository that is relevant to the sample.

   ```shell
   ant -Dsn=0111
   ```

2. See the logs in CEP server when logger service is deploying. For example,

   After deployment, the Web app is able to receive messages sent from the CEP server.

3. Navigate to `<CEP_HOME>/samples/cep/producers/http` and execute the following command.

   ```shell
   ant -Durl=http://localhost:9763/endpoints/packageArrivalsHTTPReceiver
   -DfilePath=../../artifacts/0111/arrivalEvents.txt
   ```

   This reads the arrivalEvents.txt file which contains a few sample arrival events and then sends it to the url given.

4. Then execute the following command:

   ```shell
   ant -Durl=http://localhost:9763/endpoints/packageDeliveryHTTPReceiver
   -DfilePath=../../artifacts/0111/deliveryEvents.txt
   ```

   This reads the deliveryEvents.txt file which contains some sample delivery events and then sends it to the url given.

5. After 2 minutes, we can see the outputs on the console as follows.

   ![Event output](image)

Sample 0112 - Analyzing Twitter Feeds Using Partitions
Introduction

This sample demonstrates how to set up an execution plan with a Siddhi query that monitors Twitter feeds of organizations using Siddhi partitions. It uses the following Siddhi query to check whether the word count of the Twitter feeds of each company within the last minute is greater than 10, and publishes the details of the companies with a high frequency of Twitter feeds as the output. This query has one incoming custom event stream of the word counts of various company names taken from Twitter pages. Custom events are events with custom mappings that do not adhere to the default event formats. For more information on event formats, see Event Formats.

```
partition with (company of TwitterFeed)
begin @info(name = 'query1')
from TwitterFeed#window.time(1 min)
select company as company, sum(wordCount) as words
having words > 10
insert into HighFrequentTweets;
end ;
```

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0112. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- Two streams with the IDs HighFrequentTweets:1.0.0 and twitterFeed:1.0.0
- An event receiver named WSO2EventReceiver
- An event publisher named HighFrequentTweetsLogger to log the received messages
- An execution plan named HighFrequentTweetsExecutionPlan

Executing the sample

Navigate to the `<CEP_HOME>/samples/cep/producers/wso2-event` directory, and execute the following Ant command using another tab in the CLI:

```
ant -DstreamId=twitterFeed:1.0.0 -Dsn=0112
```

The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/wso2-event/build.xml` file.

This builds and runs the WSO2Event producer, which sends Twitter feed data to the CEP server. You view the details of the events that are sent as shown below.
You view the CEP server receiving the output events in the logs of it in the CLI as shown below.

Sample 0113 - Limiting the Output Rate of an Event Stream

Introduction

This sample demonstrates how to set up an execution plan with output rate limiting functionality. It uses the following Siddhi query to output custom events with the last IP addresses received from the `loginEvents` stream for every 5 events. This has the behavior of emitting a custom event per every 5 events. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats.

```
from loginEvents
select ipAddress as ip
output last every 5 events
insert into uniqueIps;
```

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0113. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- Two streams with the IDs `org.wso2.sample.login.info:1.0.0` and `org.wso2.sample.out.unique:1.0.0`
- An event receiver named `loginInfoReceiver`.
- An event publisher named `uniqueIpPublisher` to log the received messages.
- An execution plan named `UniqueLoginExecutionPlan`.

Executing the sample

Navigate to the `<CEP_HOME>/samples/cep/produces/http` directory, and execute the following Ant command using another tab in the CLI: ant -Durl=http://localhost:9763/endpoints/loginInfoReceiver -Dsn=0113
The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/http/build.xml` file.

This builds the HTTP client and publishes the events defined in the `<CEP_HOME>/samples/cep/artifacts/013/loginInfoReceiver.txt` file to the `loginInfoReceiver` endpoint. You view the details of the events that are sent as shown below.

You view the CEP server receiving the output events in the logs of it in the CLI as shown below.

Sample 0114 - Using External Time Windows
### Introduction

This sample demonstrates how to use external time windows for a fraud detection use-case. In this sample, we look for two or more transactions done within a very short period of time and send an alert immediately when such an occurrence is detected.

The query used in this sample is as follows:

```sql
from atm_transactions#window.externalTime(meta_timestamp, 60 sec)
select cardNumber, count(cardNumber) as transactionCount, sum(amount) as totalAmount
group by cardNumber
insert current events into transactions_per_card;

from transactions_per_card[transactionCount > 1]
select cardNumber, transactionCount, totalAmount
insert into alert_stream;
```

The first query uses a 60-second external time window, which keeps events based on the time of the meta_timestamp attribute. Upon arrival of each new event, it gets a count of the transactions so far (last 60 seconds), sum of the amount per each card and emits the results to an intermediate stream named transactions_per_card.

The second query looks for the condition where more than one transaction has taken place for a specific card and sends an alert.

### Prerequisites

See Prerequisites in CEP Samples Setup page.

### Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0114. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0114` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

### Executing the sample

1. Open a terminal, go to `<CEP_HOME>/samples/cep/producers/wso2-event` and run the following command:

   ```
   ant -DstreamId=org.wso2.sample.atm.transactions:1.0.0 -Dsn=0114
   ```

   It builds and runs the `wso2event` producer, which sends sample ATM transaction events to the CEP server.
2. From the terminal opened, see the details of the events sent.

To configure host, port, username, password use 
-Dhost=xxxx  
-Dport=xxxx  
-Dusername=xxxx  
-Dpassword=xxxx

3. After sending events, you can see the outputs from the CEP console (the outputs are logged by the logger which we use for this sample), given below is part of the console output of the logger when sending events from the producer.

Sample 0115 - Quartz scheduler based alerts

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to set up an execution plan with a quartz scheduler based cron window for generating periodic alerts on build failures.

The execution plan used in this sample is as follows:

```
from buildStatisticsStream[isBuildFailed == true]#window.cron("0 0 1 * *
?")
select project, productTeam, lastCommitter, count(timeStamp) as
totalBuildFailures

group by project
insert into buildFailureStream;
```

The first query,

- Receives events through the buildStatisticsStream.
- Adds events that satisfy the condition isBuildFailed == true to a cron window. The cron window is configured to output at 1 am every day using the cron notation.
- Selects the attributes project, productTeam, lastCommitter, count(timeStamp) as totalBuildFailures.
- The group by clause causes the count() to be calculated per project.
- Outputs events to the buildFailureStream.

Prerequisites

See Prerequisites in CEP Samples Setup page.
Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0115. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0115` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

Executing the sample

1. Open another terminal, go to `<CEP_HOME>/samples/cep/artifacts/0115` and run the following command:
   ```
   ```
   It builds the http client and publishes the events at `<CEP_HOME>/samples/cep/artifacts/0115/buildStatisticsEvents.txt` to the `buildStatisticsEventReceiver` http endpoint.

2. You can see the events getting received by CEP by the logs in its console.

The above query will output the processed events at "1 am" as defined in the execurion plan. You can edit the query in "BuildFailureStatisticsPlan.siddhiql" file which is located in the `<CEP_HOME>/samples/cep/artifacts/0115/executionplans` directory to reschedule as per requirement. Below is an example of a scheduler related query which outputs processed events on 5th minute each hour. For more information see Inbuilt Windows.

```siddhiql
from buildStatisticsStream[isBuildFailed == true]#window.cron("0 5 * * *")
select project, productTeam, lastCommitter, count(timeStamp) as totalBuildFailures
group by project
insert into buildFailureStream;
```

Sample 0116 - Performing Linear Regression

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to run Linear Regression using the Timeseries Toolbox. This sample uses Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The data used for the regression is from a baseball stats dataset. The dependent variable (predictor variable) is the salary of the Baseball player based on his performance statistics which are the independent variables – rbi, walks, strikeouts and errors.
The execution plan used in this sample is as follows:

```sql
from baseballData#timeseries:regress(2, 10000, 0.95, salary, rbi, walks, strikeouts, errors)
select *
insert into regResults;
```

The inputs to the regression function are as follows.

- Calculation Interval – 2
- Batch size – 10,000
- Confidence Interval – 0.95
- Y (dependent) variable – salary
- X (independent) variables – rbi, walks, strikeouts, errors

The output of the query will be the coefficients of the regression equation for the accumulated dataset at each 2nd event. The output attributes will include the input variable values, beta coefficients for each X variable, beta zero and the standard error.

For more detail on input and output parameters of regression see Regression.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0116. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to the `<CEP_HOME>/samples/cep/artifacts/0116` directory (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Log into the CEP management console which is located at `https://localhost:9443/carbon`.

2. Go to Tools -> Event Simulator. Under the Multiple Events section, the `BaseballData.csv` file is listed. This file contains the sample data. Click Play to start sending sample events from the file.

3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

For example, given below is a screenshot of the final regression output for this data.
Sample 0117 - Filtering and Outputting to Multiple Streams

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to filter WSO2 events and custom text events and output them to multiple streams. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats.

The execution plan used in this sample is as follows,

```sql
from ServerLogs[meta_tenantId != -1234]
select meta_timestamp, meta_tenantId, className, logType
insert into TenantLogs;
```
The first query,

- Receives events through the serverLogStream.
- Filter based on the tenant id and inserts tenant into TenantLogs.

The second query,

- Receives events through the serverLogStream.
- Filter based on logType and inserts only WARN logs into WarnTypeLogStream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0117. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to the `<CEP_HOME>/samples/cep/artifacts/0117` directory (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the CEP management console which is located at https://localhost:9443/carbon. Confirm that all the artifacts are deployed properly.
2. Log out and enter incorrect credentials to log into the CEP management console. This will generate a WARN log entry.
3. See the output events received from the CEP console. This sample uses the logger adaptor to log output events to the console.

Sample 0118 - Using Hazelcast Event Tables

Introduction

Prerequisites

Building the sample

Executing the sample

Introduction

This sample demonstrates how to set up an execution plan to filter out credit card transactions that makes use of a Hazelcast event table to identify blacklisted transactions. This sample uses Event simulator for inputs and the logger
publisher for logging the outputs to the CEP console.

The execution plan used in this sample are as follows:

```java
@from(eventtable = 'hazelcast', cluster.name = 'cluster_a',
      cluster.password = 'pass@cluster_a')
define table CardUserTable (name string, cardNum string, blacklisted bool) ;
```

Given above is the table definition,

- Defines a table named CardUserTable with the given attributes. This will create a event table backed by a new Hazelcast Instance in a new Hazelcast Cluster.

```java
from CardUserStream
select *
insert into CardUserTable;
```

The first query,

- Processes the events received through the CardUserStream.
- Selects all the attributes under the `select` clause, from each event received.
- Inserts it to the CardUserTable.

```java
from BlackListStream
select cardNo as cardNum, true as blacklisted
update CardUserTable
  on cardNum == CardUserTable.cardNum;
```

The second query,

- Processes the events received through the BlackListStream.
- Selects cardNo and renames it as cardNum, introduces a new attribute named blacklisted with the value 'true' under the `select` clause, for each event received.
- Updates the CardUserTable with the condition cardNum == CardUserTable.cardNum. Here the blacklisted attribute in the table is updated with the new value.

```java
from PurchaseStream#window.length(1) as p join CardUserTable as c
  on p.cardNo == c.cardNum and c.blacklisted == false
select p.cardNo as cardNo, c.name as name, p.price as price
insert into WhiteListPurchaseStream ;
```

The third query,

- Defines a length window that keeps 1 event of the input stream PurchaseStream.
• Joins it with the CardUserTable with the condition p.cardNo == c.cardNum and c.blacklisted == false. In this condition, the events with blacklisted == true in the table get filtered out and then the remaining events are joined based on the card number.
• Emits those events as output events through the WhiteListPurchaseStream.

```java
from DeleteAllUsers
delete CardUserTable
  on deleteAll == true;
```

The last query is used to clean up the table from an external trigger event through DeleteAllUsers stream,

• It processes the events received through the DeleteAllUsers.
• Checks for the condition deleteAll == true and if it exists, deletes all the records in the CardUserTable.

Prerequisites

Set up prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0118. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

• Points the default Axis2 repo to <CEP_HOME>/samples/cep/artifacts/0118 (by default, the Axis2 repo is <CEP_HOME>/repository/deployment/server).

Executing the sample

1. Log into the CEP management console which is located at https://localhost:9443/carbon.

2. Go to Tools -> Event Simulator. Under the ‘Multiple Events’ section, you can see 4 files listed there which contains some sample data as follows.

   Send multiple events

<table>
<thead>
<tr>
<th>File</th>
<th>Stream Configuration</th>
<th>Delay between events(ms)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteUserEvents.csv</td>
<td>DeleteAllUsers:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>purchaseEvents.csv</td>
<td>PurchaseStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>userEvents.csv</td>
<td>CardUserStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
<tr>
<td>blackListUserEvents.csv</td>
<td>BlackListStream:1.0.0</td>
<td>1000</td>
<td>Play, Configure, Delete</td>
</tr>
</tbody>
</table>

   3. The userEvents.csv file contains sample data that is used to fill the in-memory CardUserTable. Click play start sending events to fill the table.

   4. The blackListUserEvents.csv file contains sample data that is used to mark user entries in CardUserTable as blacklisted. Click Play to start sending blacklisted events and mark some table entries as blacklisted.

   5. The purchaseEvents.csv contains credit card transactions data. Play it and send the transaction data.

   6. After sending sample events from purchaseEvents.csv, you will be able to see the outputs as follows.
Sample 0119 - Trigger Events at Defined Time Intervals

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to set up an execution plan with triggers. This sample uses the Event simulator for inputs and the logger publisher for logging the outputs to the CEP console.

The execution plan used in this sample is as follows.

- The following query triggers an event in the `periodicalTriggerStream` event stream every 5 seconds. The purpose of this trigger is to generate events in the `periodicalTriggerStream` event stream in order to merge it with the `sensorStream` event stream.

  ```sql
  define trigger periodicalTriggerStream at every 5 sec ;
  ```

- The following query forwards a processed event as an output to the `cronTriggerStream` event stream every 10 seconds.

  ```sql
  define trigger cronTriggerStream at '*/10 * * * * ?' ;
  ```

- The following query merges the `periodicalTriggerStream` and `sensorStream` event streams. Only the values of the attributes specified in the query are merged. Attribute values are taken from the `sensorStream` event stream and inserted into the `periodicalTriggerStream` event stream.

  ```sql
  from periodicalTriggerStream join sensorStream#window.time(10 sec) 
  select meta_timestamp, meta_isPowerSaverEnabled, meta_sensorId, 
  meta_sensorName, correlation_longitude, correlation_latitude, 
  triggered_time, humidity, sensorValue 
  insert into periodicalTriggeredSensorStream;
  ```

- The following query merges the `cronTriggerStream` and `sensorStream` event streams. Only the values of the attributes specified in the query are merged. Attribute values are taken from the `sensorStream` event stream and inserted into the `cronTriggeredSensorStream` event stream.

  ```sql
  from cronTriggerStream join sensorStream 
  select meta_timestamp, meta_isPowerSaverEnabled, meta_sensorId, 
  meta_sensorName, correlation_longitude, correlation_latitude, 
  triggered_time, humidity, sensorValue 
  insert into cronTriggeredSensorStream;
  ```
from cronTriggerStream join sensorStream#window.time(10 sec)
select meta_timestamp, meta_isPowerSaverEnabled, meta_sensorId,
meta_sensorName, correlation_longitude, correlation_latitude,
triggered_time, humidity, sensorValue
insert into cronTriggeredSensorStream;

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0119. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Creates the following event streams.
  - org.wso2.event.sensor.stream:1.0.0
  - periodicalTriggeredSensorStream:1.0.0
  - cronTriggeredSensorStream:1.0.0
- Creates an execution plan named ExecutionPlan.
- Creates the following event publishers.
  - periodicalTriggeredSensorStreamLogger
  - cronTriggeredSensorStreamLogger

Executing the sample

Follow the steps below to execute the sample.

1. Log into the CEP Management Console.
2. Click on the Tools tab and then click Event Simulator to open the Event Stream Simulator page. A file named events.csv containing sample data is displayed in the Send multiple events section as follows. Click Play to start sending events to the CEP from this file.

The events triggered in the CEP are logged in your CLI as shown in the extract below.
The events that are triggered in the `periodicalTriggerStream` event stream every 5 seconds have `periodical` as the unique ID.

The events that are triggered in the `cronTriggerStream` event stream every 10 seconds have `cron` as the unique ID.

Sample 0120 - Using the Map Extension

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates the usage of map extension with a basic pass through query. It selects some attributes that are specific for the each incoming event and gives an output event with the selected attributes. This receives data via an http request. The received events are logged by the log event publisher.

The following three queries are performed in the given order.

1. The following query is performed on the received data from the `DataIn` event stream. Then specific objects are filtered from the received stream and published to the `innerStream`. 
The following query retrieves data from the `innerStream`. Then the temperature value that belongs to the specific attributes are sent to `innerStreamTwo` with the name of the event.

```sql
from innerStream
select name, map:get(specAttrObjMap, 'temperature') as temp
insert into innerStreamTwo;
```

3. If the temperature value is not `null` in the stream received from `innerStreamTwo`, the following query casts the temperature value into a string and publishes it to the `dataOut` stream with the event name.

```sql
from innerStreamTwo
select name, cast(temp, 'string') as temperature
insert into dataOut;
```

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0120. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0120` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

Open a new tab in your CLI and execute the following ant command from the `<CEP_HOME>/samples/cep/production/http` directory.

```bash
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0120
```

The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/production/http/build.xml` file.

This builds the HTTP client and publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0120/httpReceiver.txt` file to the `httpReceiver` endpoint. You can view the details of the events that are sent as shown in the log below.
Sample 0121 - Using Event Windows

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to set up an execution plan that makes use of an event window to analyze the sensor reading of a smart home environment with multiple sensors in every room. This sample uses the Event Simulator for inputs and the Logger Publisher for logging the outputs in the CEP console.

The execution plan used in this sample is as follows.

- The following query defined the sensor input stream.

```java
define stream SensorStream (sensorType string, value float, roomNo int, deviceID string);
```

- The following query defines an event window named `SensorWindow` to collect the events that arrive and to emit them every 5 seconds.
define window SensorWindow (sensorType string, value float, roomNo int, deviceID string) timeBatch(5 seconds);

• The following query inserts the events that are passed to the SensorStream stream to the sensorWindow window.

```sql
from SensorStream
insert into SensorWindow;
```

• The following query finds the maximum reading of each sensor in each room for each 5 second interval, and inserts them into the MaxSensorReadingPerRoomStream stream.

```sql
from SensorWindow
select sensorType, max(value) as maxValue, roomNo
group by sensorType, roomNo
insert into MaxSensorReadingPerRoomStream;
```

• The following query finds the average reading of each sensor in the smart home for each 5 second interval and inserts them into the OverallAverageSensorReadingStream stream.

```sql
from SensorWindow
select sensorType, avg(value) as avgValue
group by sensorType
insert into OverallAverageSensorReadingStream;
```

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0121. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

• Creates the following event streams.
  • SensorStream:1.0.0
  • MaxSensorReadingPerRoomStream:1.0.0
  • OverallAverageSensorReadingStream:1.0.0
  • OverallMaxSensorReadingStream:1.0.0

• Creates an execution plan named SmartHome.

• Creates the following event publishers.
  • maxSensorReadingPerRoomStreamPublisher
  • overallAverageSensorReadingStreamPublisher
1. Access the WSO2 CEP Management Console using the following URL, and log in using your credentials.
https://<CEP_HOST>:<CEP_PORT>/carbon/

2. In the Tools tab, click Event Stimulator to open the Event Stream Simulator page. A file named events.csv containing sample data is displayed in the Send multiple events section as follows. Click Play to start sending events to the CEP from this file.

The events triggered in the WSO2 CLI are logged in the CLI as shown below. The following events are printed for every 5 seconds.

- **MaxSensorReadingPerRoom**
  The maximum reading of each sensor in each room with a unique ID.

- **OverallMaxSensorReading**
  The maximum reading of every sensor in the smart home with a unique ID.

- **OverallAverageSensorReading**
  The average reading of each sensor in the smart home with a unique ID.
Sample 0301 - Network Intruder Detection with PMML Extension Predictions

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to set up an execution plan with a Siddhi query to use Siddhi PMML extension in order to detect network intruders. The input event stream is processed by the execution plan which uses the pmml predictive model to detect whether a particular user is an intruder to the network or not. The output stream contains the processed query results which includes the predicted response along with the feature values extracted from the input event stream. This sample uses CEP event simulator to input data and the logger publisher for logging the outputs to the CEP console.

The query used in this sample is as follows.
from IntruderInputStream#pmml:predict('samples/cep/artifacts/0301/decision-tree.pmml') select * insert into IntruderOutputStream;

Above query:

- Receives events through the IntruderInputStream.
- Use PMML extension to predict with the aid of the decision-tree.pmml model.
- Finally outputs results into the IntruderOutputStream.

**Prerequisites**

Set up the following prerequisites before starting the configurations.

1. Set up the prerequisites required for all samples.
2. Start the server in standard mode and install Siddhi PMML Extension as a feature. For more information see Install WSO2 GPL Features.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0301. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0301` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`).

**Executing the sample**

1. Log into the CEP management console which is located at `https://localhost:9443/carbon`.
2. Go to Tools -> Event Simulator. Under the ‘Multiple Events’ section, you can see the listed ‘intruder.csv’ file which contains some sample data. Click ‘play’ to start sending sample events from the file.
3. You view the values of the IntruderOutputStream from the CEP console as shown below.

**Sample 0501 - Processing a Simple Filter Query with Apache Storm Deployment**

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**
This sample demonstrates how to configure WSO2 CEP with Apache Storm in the distributed mode, and run the simple query below in a local Storm cluster.

```java
@name('query 1') @dist(parallel='4')
from analyticsStats[meta_ipAdd != '192.168.1.1']
select meta_ipAdd, meta_index, meta_timestamp, meta_nanoTime, userID
insert into filteredStatStream;
```

Above query filters events from `analyticsStats` stream, and inserts the results into the stream named `filteredStatStream`.

The `@dist(parallel='4')` annotation denotes that this query needs to be run in four Storm tasks. You can specify a query group ID to group queries together using the `execGroup` attribute of the `@dist` annotation.

**Prerequisites**

Follow the steps below to set up the prerequisites for this sample.

1. Set up the prerequisites required for all samples.
2. Download Apache Storm and set up a Storm cluster. For instructions, refer to documentation on setting up a Storm cluster.
3. Do the following changes in the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file, to enable Hazelcast clustering in WSO2 CEP.
4. Do the following changes in the `<CEP_HOME>/repository/conf/event-processor.xml` file to disable the HA processing mode, enable Distributed processing mode, and configure WSO2 CEP to run with Apache Storm.

```xml
<axisconfig name="AxisJava2.0">
    ...
    <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
        ...
        <parameter name="membershipScheme">wka</parameter>
        ...
        <!-- The host name or IP address of this member other than localhost/127.0.0.1 -->
        <parameter name="localMemberHost">127.0.0.1</parameter>
        ...
        <members>
            <member>
                <hostName>127.0.0.1</hostName>
                <port>4000</port>
            </member>
        </members>
    ...
</clustering>
</axisconfig>
```
<eventProcessorConfiguration>

<!-- HA Mode Config -->
<mode name="HA" enable="false">
  ...
</mode>

<!-- Distributed Mode Config -->
<mode name="Distributed" enable="true">
  <nodeType>
    <worker enable="true"/>
    <manager enable="true">
      <!-- The host name or IP address of this member -->
      <hostName>127.0.0.1</hostName>
      <port>8904</port>
    </manager>
    <presenter enable="true">
      <!-- The host name or IP address of this member -->
      <hostName>127.0.0.1</hostName>
      <port>11000</port>
    </presenter>
  </nodeType>
  <management>
    <managers>
      <manager>
        <hostName>127.0.0.1</hostName>
        <port>8904</port>
      </manager>
      ...
    </managers>
    ...
  </management>
</mode>
</eventProcessorConfiguration>

5. Integrate WSO2 CEP with Apache Storm. For instructions, see the distributed mode deployment in Clustered Deployment.

If you are using a clustered deployment, add configurations on Nimbus, Zookeeper, etc. in the `<CEP_HOME>/repository/conf/cep/storm/storm.yaml` file on all nodes, for WSO2 CEP to communicate with Apache Storm. However, these configurations are not necessary for this sample, if you run Apache Storm locally.

If you are executing this sample in WSO2 DAS, you need to place a built Storm jar in the `<DAS_HOME>/repository/conf/cep/storm` directory. This jar can be created by building the pom file in this location.

### Building the sample
Start the WSO2 CEP server with the sample configuration numbered 0501. For instructions, see Starting sample CEP configurations.

This sample configuration creates the following.

- Two streams with the iDs analytics_Statistics:1.3.0 and filteredStatStream:1.0.0
- An event receiver named WSO2EventReceiver
- An event publisher named WSO2EventPublisher
- An execution plan named PreprocessStats

**Executing the sample**

Follow the steps below to execute the sample.

1. Navigate to the `<CEP_HOME>/samples/cep/consumers/wso2-event` directory, and execute the following Ant command using another tab in the CLI: `ant -Dsn=0501`

   ```
   The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/consumers/wso2-event/build.xml` file.
   ```

   This builds the sample `wso2event` consumer and executes it.

   ```
   Do not close this terminal. It is required to keep the server running and receiving events.
   ```

2. Navigate to the `<CEP_HOME>/samples/cep/producers/wso2-event/` directory, and execute the following Ant command using another tab in the CLI: `ant -DstreamId=analytics_Statistics:1.3.0 -Dsn=0501`

   ```
   The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/producers/wso2-event/build.xml` file.
   ```

   This builds and runs the `wso2event` producer, which will send analytics statistics data to the CEP server. You view the details of the events that are sent as shown below.
You view the output events received by the consumer from WSO2 CEP via the terminal opened in step 2 above as shown below.

```

```

Sample 0502 - Processing a Window State Over Server Restart

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to persist WSO2 CEP states over server restart when Single Node is used. This uses the following window query to demonstrate how you can execute the previous state of a query, even after a server stops/crashes. The custom text output events are published using a logger event publisher. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see [Event Formats](#).
from loginEvents#window.length(50)
select count(ipAddress)  as ipAddress, sum(frequency) as totalFrequency
insert into loginCount;

Above query calculates the count of the IP addresses, and the sum of the frequencies of the last 50 events arrived.

**Prerequisites**

Follow the steps below to set up the prerequisites for this sample.

1. Set up the prerequisites required for all samples.
2. Set up a node (ignore the presenter node configurations for this sample) as mentioned in the configuring High Availability in CEP nodes section. Summarised steps are as follows.

   1. Do the following changes in the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file, to enable clustering.

```xml
<axisconfig name="AxisJava2.0">
  ...
  <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
    ...
    <parameter name="membershipScheme">wka</parameter>
    ...
    <!-- The host name or IP address of this member other than localhost/127.0.0.1 -->
    <parameter name="localMemberHost">127.0.0.1</parameter>
    ...
    <members>
      <member>
        <hostName>127.0.0.1</hostName>
        <port>4000</port>
      </member>
    </members>
  </clustering>
</axisconfig>
```

2. Do the following changes in the `<CEP_HOME>/repository/conf/event-processor.xml` file, to enable HA mode. Make sure that `<persistence/>` is enabled.
<eventProcessorConfiguration>

  <!-- HA Mode Config -->
  <mode name="HA" enable="true">
    <nodeType>
      <worker enable="true"/>
      <presenter enable="true"/>
    </nodeType>
    <persistence enable="true">
      <persistenceIntervalInMinutes>1</persistenceIntervalInMinutes>
      <persisterSchedulerPoolSize>10</persisterSchedulerPoolSize>
      <persister class="org.wso2.carbon.event.processor.core.internal.persistence.FileSystemPersistenceStore">
        <property key="persistenceLocation">cep_persistence</property>
        </persister>
    </persistence>
  </mode>

  <!-- Distributed Mode Config -->
  <mode name="Distributed" enable="false">
  ... 
  </mode>

</eventProcessorConfiguration>

With above changes, CEP will take snapshots every one minute, and stores them in the <CEP_HOME>/cep_persistence/ directory. Moreover, it uses FileSystemPersistenceStore, which is the default persistence store of WSO2 CEP is used to persist the states.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered **0502**. For instructions, see Starting sample CEP configurations.

This sample configuration creates the following.

- Two streams with the IDs `org.wso2.sample.login.info:1.0.0` and `org.wso2.sample.login.count:1.0.0`
- An event receiver named `loginInfoReceiver`
- An event publisher named `loginCountPublisher`
- An execution plan named `LoginCountExecutionPlan`

Executing the sample
Follow the steps below to execute the sample.

1. Navigate to the `<CEP_HOME>/samples/cep/producers/http/` directory, and execute the following Ant command using another tab in the CLI:
   ```command
   ant -Durl=http://localhost:9763/endpoints/loginInfoReceiver -Dsn=0502
   ```

   This builds the HTTP client, and publishes the events defined in the `<CEP_HOME>/samples/cep/artifacts/0502/loginInfoReceiver.txt` file to the `loginInfoReceiver` endpoint.

   Do not close this terminal. It is required to keep the server running and receiving events.

   You view the CEP server receiving the output events in the logs of it in the CLI as shown below.

   ![Event log output](image)

   In the above output of events, the `ipCount` is incremented by one for each new event, and the `totalCount` is increased by the frequency of each event.

2. Wait for CEP server to take a snapshot and persist.

   ![Snapshot log output](image)

3. Restart the server and repeat step 1.

   You view the CEP server receiving the output events in the logs of it in the CLI as shown below.

   ![Restart log output](image)

   Event processing continues from the previous state even after you restart the server. The `ipCount` o
Sample 0503 - Processing a Window Query in High Availability Mode

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to run WSO2 CEP in High Availability (HA) mode. This uses the following window query to demonstrate how a processing continues even after one server shuts down.

```sql
from pizzaOrder#window.length(20)
select count(orderNo) as totalOrders, sum(price) as sumPrice
insert into orderCount;
```

Above query calculates the order count and the sum of the prices of the last 20 orders.

**Prerequisites**

Follow the steps below to set up the prerequisites for this sample.

This sample uses two WSO2 CEP nodes. (Node1 as the active node and Node2 as the passive node.)

1. Set up the prerequisites required for all samples.
2. Set up the two nodes (ignore the presenter node and backup node configurations for this sample) as mentioned in the configuring High Availability in CEP nodes section. Summarised steps are as follows.

   1. Set the value of the `<Offset>` element of the `<CEP_HOME>/repository/conf/carbon.xml` file as shown below in Node2, to start it with a port offset of 1.

   2. In the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file, enable clustering in wka mode on both nodes.

   3. In the `<CEP_HOME>/repository/conf/event-processor.xml` file, enable HA mode and configure both nodes to have separate `<eventSync/>` and `<management/>` host and port configs.

   4. Share the registry by adding the Node1’s H2 database as the Node2’s database.

**Building the sample**

Start both WSO2 CEP nodes with the sample configuration numbered 0503. For instructions, see Starting sample CEP configurations.

This sample configuration creates the following.

- Two streams with the IDs `org.wso2.sample.pizza.orderStream:1.0.0` and `org.wso2.sample.order.count:1.0.0`
An event receiver named pizzaOrderReceiver
An event publisher named pizzaCountPublisher
An execution plan named PizzaOrdersExecutionPlan

 Executing the sample

Follow the steps below to execute the sample.

1. Navigate to the <CEP_HOME>/samples/cep/producers/http/ directory of any node, and execute the following Ant command using another tab in the CLI:
   
   ant -Durl=http://localhost:9763/endpoints/pizzaOrderReceiver -Dsn=0503

   This builds the HTTP client and publishes the events defined in the <CEP_HOME>/samples/cep/artifacts/0503/pizzaOrderReceiver.txt file to the pizzaOrderReceiver endpoint in Node1.

   You view **Node 1** (active node) receiving the output events in the logs of it in the CLI as shown below.

   ![Node 1 receiving output events](image)

   Value of totalOrders is incremented by one for each new event, and the sum of the prices is increased by the frequency of each event.

2. Repeat step 1 using the following Ant command: ant -Durl=http://localhost:9764/endpoints/pizzaOrderReceiver -Dsn=0503

   This builds the HTTP client and publishes the events defined in the <CEP_HOME>/samples/cep/artifacts/0503/pizzaOrderReceiver.txt file to the pizzaOrderReceiver endpoint in Node2.

   You view **Node 1** (active node) receiving the output events in the logs of it in the CLI as shown below.

   ![Node 1 receiving output events](image)

   Even if events are sent to one server, the two nodes coordinate in a way so that both receive all the events. Only the active node publishes the output events.

3. Shutdown the active node (**Node1**). Now, **Node2** becomes the active node.
4. Repeat step 2 and use the command to send events to the running node.
You view the running node receiving the output events in the logs of it in the CLI as shown below.

![Log output of running node](image)

Even though one node is down, the other carries on processing the events.

**Sample 0504 - Processing a Distributed Siddhi Query with Partitioning by integrating with Apache Storm**

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to configure WSO2 CEP with Apache Storm in the distributed mode, and run the sample query below in a local/distributed Storm cluster.

```siddhi
@name('query1')
@dist(parallel='4', execGroup='1')
from analyticsStats
select meta_timestamp, str:contains(userID, 'wso2') as isValidUserID,
    userID
insert into filteredAnalyticsStats;

@name('query2')
@dist(parallel='4', execGroup='1')
from filteredAnalyticsStats[isValidUserID == true]
select * insert into validAnalyticsStat;

@name('query3')
@dist(parallel='2', execGroup='2')
partition with (userID of validAnalyticsStat)
begin
    from validAnalyticsStat#window.lengthBatch(3)
    select userID, max(meta_timestamp) as latestTimestamp,
        min(meta_timestamp) as earliestTimestamp
    insert into processedAnalyticsStats;
end;

@name('query4')
@dist(parallel='3', execGroup='3')
from processedAnalyticsStats
select userID , (latestTimestamp - earliestTimestamp) as difference
insert into processedStream;
```

Above query filters events from `analyticsStats` stream, and inserts the results into the stream named `processedStream`.
Prerequisites

Follow the steps below to set up the prerequisites for this sample.

1. Set up the **prerequisites required for all samples**.
2. Download Apache Storm and set up a Storm cluster. For instructions, refer to documentation on **setting up a Storm cluster**.
3. Do the following changes in the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file, to enable Hazelcast clustering in WSO2 CEP.

```xml
<axisconfig name="AxisJava2.0">
  ...
  <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
    ...
    <parameter name="membershipScheme">wka</parameter>
    ...
    <!-- The host name or IP address of this member other then localhost/127.0.0.1 -->
    <parameter name="localMemberHost">127.0.0.1</parameter>
    ...
    <members>
      <member>
        <hostName>127.0.0.1</hostName>
        <port>4000</port>
      </member>
    </members>
  </clustering>
</axisconfig>
```

4. Do the following changes in the `<CEP_HOME>/repository/conf/event-processor.xml` file to disable the HA processing mode, enable Distributed processing mode, and configure WSO2 CEP to run with Apache Storm.

The `@dist(parallel='4')` annotation denotes that this query needs to be run in four Storm tasks. You can specify a query group ID to group queries together using the `execGroup` attribute of the `@dist` annotation.
<eventProcessorConfiguration>

<!-- HA Mode Config -->
<mode name="HA" enable="false">
...</mode>

<!-- Distributed Mode Config -->
<mode name="Distributed" enable="true">

<nodeType>

<worker enable="true"/>

<manager enable="true">

<!-- The host name or IP address of this member -->

<hostName>127.0.0.1</hostName>
<port>8904</port>
</manager>

<presenter enable="true">

<!-- The host name or IP address of this member -->

<hostName>127.0.0.1</hostName>
<port>11000</port>
</presenter>
</nodeType>

<management>

<managers>

<manager>

<hostName>127.0.0.1</hostName>
<port>8904</port>
</manager>

</managers>
...
</management>
...
</mode>

</eventProcessorConfiguration>

5. Integrate WSO2 CEP with Apache Storm. For instructions, see the distributed mode deployment in Clustered Deployment.

If you are using a clustered deployment, add configurations on Nimbus, Zookeeper, etc. in the `<CEP_HOME>/repository/conf/cep/storm/storm.yaml` file on all nodes, for WSO2 CEP to communicate with Apache Storm. However, these configurations are not necessary for this sample, if you run Apache Storm locally.

If you are executing this sample in WSO2 DAS, you need to place a built Storm jar in the `<DAS_HOME>/repository/conf/cep/storm` directory. This jar can be created by building the pom file in this location.

Building the sample
Start the WSO2 CEP server with the sample configuration numbered 0504. For instructions, see Starting sample CEP configurations.

This sample configuration creates the following:

- Two streams with the IDs `analytics_Statistics:1.3.0` and `processedStream:1.0.0`
- An event receiver named `WSO2EventReceiver`
- An event publisher named `LoggerPublisher`
- An execution plan named `StatExecutionPlan`

**Executing the sample**

Follow the steps below to execute the sample.

1. Navigate to the `<CEP_HOME>/samples/cep/producers/wso2-event/` directory, and execute the following Ant command using another tab in the CLI:
   ```
   ant -DstreamId=analytics_Statistics:1.3.0 -Dsn=0504
   ```
   The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/cep/samples/producers/wso2-event/build.xml` file.

2. This builds and runs the `wso2event` producer, which will send analytics statistics data to the CEP server. You view the details of the events that are sent as shown below.

3. You view the output events received by the consumer from WSO2 CEP via the terminal opened when running CEP in foreground.
Sample 1001 - WSO2 CEP Geo Dashboard

Introduction

This sample demonstrates how the geo dashboard of the WSO2 CEP can be used. Full details about setting up, configuring and running this sample is available at WSO2 CEP Geo Dashboard.

Predictive Analytics with WSO2 Machine Learner

- Siddhi syntax for the extension
- Prerequisites
- Installing required features in WSO2 CEP
- Creating the input stream
- Creating the output stream
- Creating the execution plan
- Creating an event publisher
- Simulating events

This extension facilitates you to use the machine learning models which you generate using WSO2 ML within WSO2 Complex Event Processor (CEP) for making predictions. Thereby, it integrates WSO2 ML with WSO2 CEP, to perform realtime predictions on an event stream by applying a model generated by WSO2 ML. An input event stream which is received by an event receiver of WSO2 CEP is processed by executing an execution plan within WSO2 CEP. This execution plan is written using Siddhi language. It processes the input event stream by applying the model generated using WSO2 ML. The output of this processing which includes the prediction will be published to an output stream through an event publisher of WSO2 CEP. For more information on WSO2 CEP, go to WSO2 CEP Documentation.

Siddhi syntax for the extension

There are two possible Siddhi query syntaxes to use the extension in an execution plan as follows.

1. `<double | float|long|int|string|boolean > predict(string> pathToMLModel, <string> dataType)`
   - **Extension Type:** StreamProcessor
   - **Description:** Returns an output event with the additional attribute with the response variable name of the model, set with the predicted value, using the feature values extracted from the input event.
   - **Parameter:** pathToMLModel: The file path or the registry path where ML model is located. If the model storage location is registry, the value of this this parameter should have the prefix “registry: ”
   - **Parameter:** dataType: Data type of the predicted value (double, float, long, integer/int, string, boolean/bool).
   - **Example:** predict(‘registry:/_system/governance/mlmodels/indian-diabetes-mode 1’)

2. `<double | float|long|int|string|boolean > predict(<string> pathToMLModel, <string> dataType, <double> input)`
   - **Extension Type:** StreamProcessor
   - **Description:** Returns an output event with the additional attribute with the response variable name of
the model, set with the predicted value, using the feature values extracted from the input event.

- **Parameter: pathToMLModel:** The file path or the registry path where ML model is located. If the model storage location is registry, the value of this parameter should have the prefix "registry:"
- **Parameter: dataType:** Data type of the predicted value (double, float, long, integer/int, string, boolean/bool).

- **Parameter: input:** A variable attribute value of the input stream which is sent to the ML model as feature values for predictions. Function does not accept any constant values as input parameters. You can have multiple input parameters.

- **Example:** `predict('registry:/_system/governance/mlmodels/indian-diabetes-model', NumPregnancies, TSFT, DPF, BMI, DBP, PG2, Age, SI2)

Siddhi query examples

A few Siddhi query examples that can be used in execution plans within WSO2 CEP based on the above syntaxes are as follows.

**Example 1**

```
@Import('InputStream:1.0.0')
define stream InputStream (NumPregnancies double, TSFT double, DPF double, BMI double, DBP double, PG2 double, Age double, SI2 double);
@Export('PredictionStream:1.0.0')
define stream PredictionStream (NumPregnancies double, TSFT double, DPF double, BMI double, DBP double, PG2 double, Age double, SI2 double, Class double);
from InputStream#ml:predict('registry:/_system/governance/ml/indian-diabetes-model', 'double')
select *
insert into PredictionStream;
```

**Example 2**

```
@Import('InputStream:1.0.0')
define stream InputStream (NumPregnancies double, TSFT double, DPF double, BMI double, DBP double, PG2 double, Age double, SI2 double);
@Export('PredictionStream:1.0.0')
define stream PredictionStream (NumPregnancies double, TSFT double, DPF double, BMI double, DBP double, PG2 double, Age double, SI2 double, Class double);
from InputStream#ml:predict('registry:/_system/governance/ml/indian-diabetes-model', 'double', NumPregnancies, TSFT, DPF, BMI, DBP, PG2, Age, SI2)
select *
insert into PredictionStream;
```

**Example 3**

```
```
@Import('InputStream:1.0.0')
define stream InputStream (NumPregnancies double, TSFT double, DPF double, 
BMI double, DBP double, PG2 double, Age double, SI2 double);
@Export('OutStream:1.0.0')
define stream OutStream (Class double);
from
InputStream#ml:predict('registry:/_system/governance/ml/indian-diabetes-model', 'double', NumPregnancies, TSFT, DPF, BMI, DBP, PG2, Age, SI2)
select Class
insert into OutStream;

### Prerequisites

Set up the following prerequisites before starting the configurations.

1. Download WSO2 ML, and start the server. For instructions, see [Getting Started](#).

Generate a model using WSO2 ML which you will use to make the predictions. For instructions on generating a model in WSO2 ML, see [Generating Models](#).

When following the instructions in [Generating Models](#), do the following changes.

- In step 9, select **RANDOM FOREST CLASSIFICATION** as the algorithm name and **Class** as the response variable.

- In step 11, enter **50** in the **Num Trees** parameter and leave the other parameter values unchanged.
1. Download WSO2 CEP, and start the server. For instructions, see Getting Started.

**Installing required features in WSO2 CEP**

1. Install the **WSO2 Machine Learner (ML) Features**. For detailed instructions to install WSO2 ML features, see Installing Machine Learner Features.

**Creating the input stream**

Follow the steps below to create the input stream in WSO2 CEP.

1. Log in to the CEP management console using the following URL, if you are not already logged in: https://<CEP_HOME>:9443/carbon/
2. Click **Main**, and then click **Event Streams** in the **Event Processor** menu.
3. Click **Add Event Stream**.
4. Enter details of the stream definition that you want to create as shown below.
5. Click **Add Event Stream**, to create the event stream in the system. You view the new input stream added to the list of all available event streams as shown below.

### Creating the output stream

Follow the steps below to create the output stream in WSO2 CEP.

1. Log in to the CEP management console using the following URL, if you are not already logged in: https://<CEP_HOME>:<CEP_PORT>/carbon/
2. Click **Main**, and then click **Event Streams** in the **Event Processor** menu.
3. Click **Add Event Stream**.
4. Enter details of the stream definition that you want to create as shown below.
5. Click **Add Event Stream**, to create the event stream in the system. You view the new output stream added to the list of all available event streams as shown below.

   **Available Event Streams**

<table>
<thead>
<tr>
<th>Event Stream Id</th>
<th>Event Stream Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PredictionStream:1.0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inputStream:1.0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Creating the execution plan**

Follow the steps below to create the execution plan in WSO2 CEP.

1. Log in to the CEP management console using the following URL, if you are not already logged in: https://<CEP_HOME>:<CEP_PORT>/carbon/
2. Click **Main**, and then click **Execution Plans** in the **Streaming Analytics** menu.
3. Click **Add Execution Plan**.
4. Select **InputStream:1.0.0** for **Import Stream** from the list, and click **Import**.
5. Enter the name of the output stream (**PredictionStream**) for **Value Of**, select **PredictionStream:1.0.0** from the **Stream ID** list, and click **Export** for **Export Stream**.

6. Enter the following Siddhi query as shown below.
Replace your file or registry path where to locate the downloaded ML model as the first argument and the response variable data-type as the second variable of the `predict()` function in the below execution plan as described in Siddhi syntax for the extension.

```siddhi
from InputStream#ml:predict('registry:/_system/governance/ml/indian-diabetes-model', 'double')
select *
insert into PredictionStream;
```

Click **Validate Query Expressions**, to validate the query you entered.

7. Click **Add Execution Plan** to save the execution plan in the system. You view the execution plan added to the list of all available execution plans as shown below.

---

**Creating an event publisher**

Follow the steps below to create an event publisher in WSO2 CEP.

1. Log in to the CEP management console using the following URL, if you are not already logged in: https://<CEP_HOME>:<CEP_PORT>/carbon/
2. Click **Main**, and then click **Execution Plans** in the **Event** menu.
3. Click **Add Event Publisher**.
4. Enter the details as shown below to create a logger type event publisher.
5. Click **Add Event Publisher**, to create the event publisher in the system. You view the new event publisher added to the list of available event publishers.

6. Click the corresponding **Enable Tracing** button as shown below.

---

**Simulating events**

Follow the steps below to simulate the sending of events in WSO2 CEP.

1. Log in to the CEP management console using the following URL, if you are not already logged in: `https://<CEP_HOME>:<CEP_PORT>/carbon/`
2. Click **Tools**, and then click **Event Simulator**.
3. Select **InputStream:1.0.0** for the **Event Stream Name**.
4. Enter the feature values to predict, in the **Stream Attributes** input fields as shown below.
5. Click **Send**, to send the events. You view the values of the output stream named **PredictionStream** logged by the event publisher in the back end console logs of WSO2 CEP as shown below.

```
```

**Sample 1501 - Viewing Real Time Analytics**

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to view and analyze real time statistics on the Analytics Dashboard.

**Prerequisites**

Set up the **prerequisites required for all samples**.

**Building the sample**

This sample configuration does the following.
- Creates an event stream named `realTimeStreamData`.
- Creates a UI publisher named `realTimeDataUiPublisher`.
- Creates a dashboard named `Real Time Analytics Dashboard`.
- Creates the following three gadgets.
  - `POPULATIONPERCOUNTRY`
  - `EVENTRECORDS`
  - `NUMOFVEHICLESPERCOUNTRY`

**Executing the sample**

Follow the steps below to execute this sample.

1. Execute one of the following commands to start the WSO2 CEP server.
   - On Windows: `<CEP_HOME>/bin\wso2server.bat --run`
   - On Linux/Solaris/Mac OS: `sh <CEP_HOME>/bin/wso2server.sh`

2. Copy the contents in the `<CEP_HOME>/samples/cep/artifacts/1501/carbonapps` directory to the `<CEP_HOME>/repository/deployment/server/carbonapps` directory.

   **If the carbonapps directory does not already exist in the `<CEP_HOME>/repository/deployment/server/carbonapps` directory, create it and copy the required content to it.**

3. Copy the contents in the `<CEP_HOME>/samples/cep/artifacts/1501/eventsimulatorfiles` directory to the `<CEP_HOME>/repository/deployment/server/eventsimulatorfiles` directory.

4. Log into the WSO2 CEP Management Console via the following URL.
   `https://<CEP_HOST>:<CEP_PORT/carbon/`

5. In the Tools tab click **Event Simulator** to open the Event Stream Simulator page.

6. Select `realTimeStreamData:1.0.0` for the **Event Stream Name** field and click **Play** for the `realTimeDashboardData.csv` file.

7. In the Main tab, click **Analytics Dashboard**, and log into the Analytics Dashboard using your credentials. The Real Time Analytics dashboard is displayed as follows.

8. Click **View**. Real time statistics generated are displayed as follows.
Samples on Publishing Events

- Sample 0051 - Publishing JSON Events via Logger Transport
- Sample 0052 - Publishing Custom JSON Events via Logger Transport
- Sample 0053 - Publishing XML Events via Logger Transport
- Sample 0054 - Publishing Custom XML Events via Logger Transport
- Sample 0055 - Publishing Text Events via Logger Transport
- Sample 0056 - Publishing Custom Text Events via Logger Transport
- Sample 0057 - Publishing WSO2 Events via WSO2Event Transport
- Sample 0058 - Publishing Custom WSO2 Events via WSO2Event Transport
- Sample 0059 - Publishing Map and Text Events via JMS Transport - ActiveMQ
- Sample 0060 - Publishing Custom Map and JSON Events via JMS Transport - Qpid
- Sample 0061 - Publishing Map and XML Events via JMS Transport - WSO2 MB
- Sample 0062 - Publishing JSON, JSON, and Custom Text Events via HTTP Transport
- Sample 0063 - Publishing XML Events via SOAP Transport
- Sample 0064 - Publishing Text Events via Email Transport
- Sample 0065 - Publishing JSON Events via SMS Transport
- Sample 0066 - Publishing JSON Events via MQTT Transport
- Sample 0067 - Publishing Map Events via Cassandra Transport
- Sample 0068 - Publishing XML Events via Kafka Transport
- Sample 0069 - Publishing JSON Events via WebSocket Transport
- Sample 0070 - Publishing JSON Events via Websocket-Local Output Event Adapter
- Sample 0071 - Publishing WSO2Event Events via UI Transport
- Sample 0072 - Publishing Map Events via RDBMS Transport

Sample 0051 - Publishing JSON Events via Logger Transport
Introduction
This sample demonstrates how to publish JSON events via logger transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites
Set up the prerequisites required for all samples.

Building the sample
Start the WSO2 CEP server with the sample configuration numbered 0051. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Creates a stream with the ID org.wso2.event.sensor.stream_1.0.0
- Creates an event publisher named logger
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0051/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample
Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click Tools, and then click Event Simulator.
3. Click Play on the corresponding event stream as shown below, to send the events in the <CEP_HOME>/samples/cep/artifacts/0051/eventsimulatorfiles/events.csv file to the publisher. Here, you can also pause/stop/resume sending events as well by using corresponding buttons.

You view the JSON events that are published to the CEP server in the logs of it in the CLI as shown below.

Sample 0052 - Publishing Custom JSON Events via Logger Transport
Introduction

This sample demonstrates how to publish custom JSON events via logger transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0052. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.sensor.stream_1.0.0
- Creates an event publisher named logger
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0052/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click Tools, and then click Event Simulator.
3. Click Play on the corresponding event stream as shown below, to send the events in the <CEP_HOME>/samples/cep/artifacts/0052/eventsimulatorfiles/events.csv file to the publisher.

You view the JSON events that are published to the CEP server in the logs of it in the CLI as shown below.
Sample 0053 - Publishing XML Events via Logger Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish XML events via logger transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0053. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.sensor.stream.1.0.0
- Creates an event publisher named logger
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0053/eventsimulatorfiles/events.csv file to the event simulator.
Executing the sample

Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click **Tools**, and then click **Event Simulator**.
3. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0053/eventsimulatorfiles/events.csv` file to the publisher.

You view the XML events that are published to the CEP server in the logs of it in the CLI as shown below.

Sample 0054 - Publishing Custom XML Events via Logger Transport

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to publish custom XML events via logger transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

**Prerequisites**

Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0054**. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID `org.wso2.event.sensor.stream_1.0.0`
- Creates an event publisher named `logger`
- Loads the events stored in the `<CEP_HOME>/samples/cep/artifacts/0054/eventsimulatorfiles/events.csv` file to the event simulator.

**Executing the sample**

Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click **Tools**, and then click **Event Simulator**.
3. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0054/eventsimulatorfiles/events.csv` file to the publisher.

Send multiple events

You view the XML events that are published to the CEP server in the logs of it in the CLI as shown below.

Sample 0055 - Publishing Text Events via Logger Transport
Introduction

This sample demonstrates how to publish text events via logger transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0055. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.sensor.stream_1.0.0
- Creates an event publisher named logger
- Loads the events stored in the <CEP_HOME>/samples/cep/artifacts/0055/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click Tools, and then click Event Simulator.
3. Click Play on the corresponding event stream as shown below, to send the events in the <CEP_HOME>/samples/cep/artifacts/0055/eventsimulatorfiles/events.csv file to the publisher.
You view the text events that are published to the CEP server in the logs of it in the CLI as shown below.

Sample 0056 - Publishing Custom Text Events via Logger Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish custom text events via logger transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0056. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.sensor.stream_1.0.0
- Creates an event publisher named logger
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0056/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample
Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click **Tools**, and then click **Event Simulator**.
3. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0056/eventsimulatorfiles/events.csv` file to the publisher.

You view the test events that are published to the CEP server in the logs of it in the CLI as shown below.

---

Sample 0057 - Publishing WSO2 Events via WSO2Event Transport
Introduction

This sample demonstrates how to publish WSO2 events via Thrift transport to a WSO2 Event client. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0057. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- A stream with the ID org.wso2.event.sensor.stream:1.0.0
- An event publisher named eventPublisher
- An events have been stored in events.csv which are loaded to event simulator.

Executing the sample

Navigate to the <CEP_HOME>/samples/cep/consumers/wso2-event/ directory, and execute the following Ant command using another tab in the CLI: ant -Dsn=0057

The other optional parameters that can be used in the above command are defined in the <CEP_HOME>/samples/cep/consumers/wso2-event/build.xml file.

This builds client which fetches the the published events in the <CEP_HOME>/samples/cep/artifacts/0057/eventsimulatorfiles/events.csv file from the emailPublisher endpoint.

Go to the event simulator and click play to send events contain in events.csv file

Received events will be shown in the Wso2Event client terminal console.
Sample 0058 - Publishing Custom WSO2 Events via WSO2Event Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish custom WSO2 events via WSO2Event transport. Custom events are
events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0058. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.sensor.stream.map:1.0.0
- Creates an event publisher named eventPublisher
- Loads the events stored in the <CEP_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample.

1. Navigate to the <CEP_HOME>/samples/cep/consumers/wso2-event/ directory, and execute the following Ant command using another tab in the CLI: ant -Dsn=0058

   The other optional parameters that can be used in the above command are defined in the <CEP_HOME>/samples/cep/consumers/wso2-event/build.xml file.

   This builds the WSO2Event client which fetches the published events in the <CEP_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv file from the eventPublisher endpoint as shown below.

   ![Command Output]

2. Log in to the management console.
3. Click Tools, and then click Event Simulator.
4. Click Play on the corresponding event stream as shown below, to send the events in the <CEP_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv file to the publisher.
You view the WSO2 events that are published to the CEP server in the logs of the consumer terminal as shown below.

```java
[Java]
Event{
    streamId='org.wso2.event.sensor.stream.map:1.0.0',
    timeStamp=0,
    metaData=[199008131345, false, 103, temperature],
    correlationData=[23.45656, 7.12324],
    payloadData=[100.34, 23.4545],
    arbitraryDataMap=null,
}
]
} for username admin
eventListSize=3 eventList [ ]
Event{
    streamId='org.wso2.event.sensor.stream.map:1.0.0',
    timeStamp=0,
    metaData=[199008131245, false, 100, temperature],
    correlationData=[23.45656, 7.12324],
    payloadData=[100.34, 23.4545],
    arbitraryDataMap=null,
}
]
},
Event{
    streamId='org.wso2.event.sensor.stream.map:1.0.0',
    timeStamp=0,
    metaData=[199008131285, true, 101, temperature],
    correlationData=[23.45656, 7.12324],
    payloadData=[100.34, 23.4545],
    arbitraryDataMap=null,
}
]
},
Event{
    streamId='org.wso2.event.sensor.stream.map:1.0.0',
    timeStamp=0,
    metaData=[199008131325, true, 102, temperature],
    correlationData=[23.45656, 7.12324],
    payloadData=[100.34, 23.4545],
    arbitraryDataMap=null,
}
]
} for username admin
eventListSize=1 eventList [ ]
Event{
    streamId='org.wso2.event.sensor.stream.map:1.0.0',
    timeStamp=0,
    metaData=[199008131345, false, 103, temperature],
    correlationData=[23.45656, 7.12324],
    payloadData=[100.34, 23.4545],
    arbitraryDataMap=null,
}
]
}
```

Sample 0059 - Publishing Map and Text Events via JMS Transport - ActiveMQ

- **Introduction**
- **Prerequisites**
Introduction

This sample demonstrates how to publish Map and Text events via JMS transport. You consume the published events in WSO2 CEP using a JMS client which is subscribed to ActiveMQ broker.

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install Apache ActiveMQ JMS event adapter.
   
   This guide uses ActiveMQ versions 5.7.0. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to [Apache ActiveMQ Documentation](#).
2. Issue the following command from the `<ACTIVEMQ_HOME>/bin` directory to start ActiveMQ.
   ```bash
   ./activemq console
   
   Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Configure sample client by adding relevant jars. See [setting up JMS for JMS sample clients](#).
4. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the `register some connection factories` section. And add the topics to be sent to the ActiveMQ broker

   ```
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/test?brokerlist='tcp://localhost:5672'
   topic.topicMap = topicMap
   topic.topicText = topicText
   ```

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0059. For instructions, see [Starting sample CEP configurations](#). This sample configuration creates the following:

- A stream with the ID `org.wso2.event.sensor.stream:1.0.0`
- Two event publishers named `jmsPublisherMap` and `jmsPublisherText`
- Loads the events stored in the `<CEP_HOME>/samples/cep/artifacts/0059/eventsimulatortfiles/events.csv` file to the event simulator.

Executing the sample

1. Open another terminal and navigate to the `<CEP_HOME>/samples/cep/consumers/jms` directory.

   and run one of the following commands according to the type of topic name of the messages suppose to consume
   ```
   ant -Dbroker=activemq -DtopicName=topicMap
   ant -Dbroker=activemq -DtopicName=topicText
   ```

   It subscribes to the ActiveMQ broker under the mentioned topic name.

2. Play the `event.csv` file deployed using the Event Simulator. This will send an event flow through the stream.
to the JMS publisher.

3. You can see the events getting published by CEP by the logs in the jms consumer console.

   for Map formatted events:

   ```java
   Received Map Message : 
   ```
   `{isPowerSaverEnabled=false, 
    timestamp=199088131245, 
    sensorValue=23.4545, 
    humidity=100.34, 
    sensorId=100, 
    longitude=23.45656, 
    latitude=7.12324, 
    sensorName=temperature}`

   Received Map Message :
   ```
   `{isPowerSaverEnabled=true, 
    timestamp=199088131245, 
    sensorValue=23.4545, 
    humidity=100.34, 
    sensorId=101, 
    longitude=23.45656, 
    latitude=7.12324, 
    sensorName=temperature}`

   Received Map Message :
   ```
   `{isPowerSaverEnabled=false, 
    timestamp=199088131245, 
    sensorValue=23.4545, 
    humidity=100.34, 
    sensorId=104, 
    longitude=23.45656, 
    latitude=7.12324, 
    sensorName=temperature}`

   ```java
   Received Text Message : meta_timestamp=199088131245,
   ```

   for Text formatted events:

   ```java
   Listening for messages
   ```

   ```java
   Received Text Message : meta_timestamp=199088131245,
   ```

Sample 0060 - Publishing Custom Map and JSON Events via JMS Transport - Qpid

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to publish custom Map and JSON events via JMS transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. You consume the published events in CEP using aJMS client which is subscribed to Qpid broker.

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install JMS-Qpid Broker and JMS-Qpid Client.
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.

3. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.

4. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the `register some connection factories` section. And add the topics to be sent to the Qpid broker.

```java
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/default?
brokerlist='tcp://localhost:5672'
topic.topicMap = topicMap
topic.topicJSON = topicJSON
```

5. Start Qpid Broker with `.qpid-server start` command by navigating to `<QPID-BROKER_HOME>/bin`.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered 0060. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/sample/cep/artifacts/0060`.
- Creates a stream with id `org.wso2.event.sensor.stream:1.0.0`.
- Create event publishers named `jmsPublisherMap` and `jmsPublisherText`.
- Hot deploys the `event.csv` file along with event configuration xml.

**Executing the sample**

1. Start WSO2 CEP with sample 0060 configurations. For more information, see Starting sample CEP configurations.

2. Open another terminal, go to `<CEP_HOME>/samples/cep/consumers/jms` and run one of the following commands according to the type of topic name of the messages suppose to consume:

```
ant -Dbroker=qpid -DtopicName=topicMap
ant -Dbroker=qpid -DtopicName=topicJSON
```

It subscribes to the Qpid broker under the mentioned topic name.

3. Play the `event.csv` file deployed using the Event Simulator. This will send an event flow through the stream to the JMS publishers.

4. You can see the events getting published by CEP by the logs in the jms consumer console.

For Map formatted events:

```
[java] Received Map Message : {timestamp=199080131245, humidity=100.34, sensorId=108, longitude=23.45656, powerSaving=false, latitude=7.12324, sensorName=temperature, temperature=23.4545}
[java] Received Map Message : {timestamp=199080131245, humidity=100.34, sensorId=101, longitude=23.45656, powerSaving=true, latitude=7.12324, sensorName=temperature, temperature=23.4545}
```

For Text formatted events:

```
```
Sample 0061 - Publishing Map and XML Events via JMS Transport - WSO2 MB

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish map and XML events via JMS transport. You consume the published events in CEP using a JMS client which is subscribed to WSO2 Message Broker. This sample does not do any processing on the outgoing event.

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Before you configure the WSO2 CEP:
   - Start WSO2 MB 3.0.0
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport
3. Configure sample client by adding relevant jars. See setting up JMS for JMS sample clients.
4. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the `register some connection factories` section. And add the topics to be sent to the Qpid broker.

   ```
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@localhost:5672
   topic.topicMap = topicMap
   topic.topicXML = topicXML
   ```

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0061 with an off-port since the WSO2 MB has started in the default port. For instructions, see Starting sample CEP configurations and append `-DportOffset=1` `-Dqpid.dest_syntax=BURL` to the command as follows.

   ```
   ./wso2cep-samples.sh -sn 0061 -DportOffset=1 -Dqpid.dest_syntax=BURL
   ```

   - Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/sample/artifacts/0061`
Executing the sample

1. Start WSO2 CEP with sample 0061 configurations. For more information, see Starting sample CEP configurations.
2. Open two separate terminals for `<CEP_HOME>/samples/cep/consumers/jms`, and run the following commands according to the type of topic name of the messages suppose to consume
   
   ```
   ant -Dbroker=mb -DtopicName=topicMap
   ant -Dbroker=mb -DtopicName=topicXML
   ```

   It subscribes to the WSO2 Message Broker under the mentioned topic name.

3. Play the `event.csv` file deployed using the Event Simulator. This will send an event flow through the stream to the JMS publishers.
4. You can see the events getting published by CEP by the logs in the jms consumer console.

   for Map formatted events:

   ```
   [java] Received Map Message : {isPowerSaverEnabled=false, timestamp=199808131245, sensorValue=23.4545, humidity=88.34, sensorId=100, longitude=7.12324, latitude=7.12324, sensorName=temperature}
   [java] Received Map Message : {isPowerSaverEnabled=true, timestamp=199808131245, sensorValue=23.4545, humidity=88.34, sensorId=101, longitude=7.12324, latitude=7.12324, sensorName=temperature}
   ```

   for XML formatted events:

   ```
   [java] Received Text Message : <event><eventMeta>timeStamp=199808131245</eventMeta><isPowerSaverEnabled=false><sensorId=100</sensorId><sensorName=temperature></sensorName></isPowerSaverEnabled><sensorValue=23.4545><humidity=88.34><correlationData=7.12324></correlationData></humidity></sensorValue><payloadData=7.12324></payloadData><event></event>
   ```

Sample 0062 - Publishing XML, JSON, and Custom Text Events via HTTP Transport

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to publish XML, JSON or custom Text events via HTTP transport. Custom events are events with custom mappings that does not adhere to the default event formats. For more information on event formats, see Event Formats. This sample does not do any processing on the outgoing event. Use log event publisher, and log the published event to verify if the messages have been properly published.
Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0062. For instructions, see Starting sample CEP configurations. This sample configuration does the following:

- Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0062`
- Creates three streams with the following ids:
  - org.wso2.event.transaction.stream:1.0.0
  - org.wso2.event.keyword.stream:1.0.0
  - org.wso2.event.message.stream:1.0.0
- Loads the events-xml.csv, events-json.csv, events-text.csv and related configuration files so that event simulator can be used to send events to the three different pre-configured event publishers.
- Creates the following three event publishers to log the received messages via an HTTP servlet receiver:
  - httpXml - Publishes events in default XML event format
  - httpJson - Publishes events in default JSON event format
  - httpText - Publishes the event as a customized text message

Executing the sample

1. Open another terminal, go to `<CEP_HOME>/samples/cep/consumers/generic-log-service` and run the following command:

   `ant -Dsn=0062`

   It builds the servlet web application that logs any HTTP traffic it receives. Events published by the http event publisher will be sent to this web application and logged.

2. Go to the management console of CEP and determine the required stream and select ‘Play’ to play the events.

   ![Management Console](image)

   The following streams are configured to publish the following event formats via the different pre-configured http event output adapters.
   - org.wso2.event.transaction.stream:1.0.0 - XML - via httpXml event publisher
   - org.wso2.event.keyword.stream:1.0.0 - JSON - via httpJson event publisher
   - org.wso2.event.message.stream:1.0.0 - Text - via httpText event publisher

3. You can see the events getting received by CEP by the logs printed in the CEP console by the generic-log-service web application.
Sample 0063 - Publishing XML Events via SOAP Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish XML events via SOAP transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0063. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream with the ID org.wso2.event.statistics.stream:1.0.0
- Creates an event publisher named soap.
- Loads the events stored in the <CEP_HOME>/samples/cep/artifacts/0063/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample.

1. Navigate to the <CEP_HOME>/samples/cep/consumers/axis2-log-service/ directory, and execute the following Ant command using another tab in the CLI: ant -Dsn=0063

   The other optional parameters that can be used in the above command are defined in the <CEP_HOME>/samples/cep/consumers/axis2-log-service/build.xml file.

This builds the WSO2Event client which fetches the published events in the <CEP_HOME>/samples/cep/artifacts/0063/eventsimulatorfiles/events.csv file from the soap endpoint as shown below.
2. Log in to the management console.
3. Click **Tools**, and then click **Event Simulator**.
4. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0063/eventsimulatorfiles/events.csv` file to the publisher.

   **Send multiple events**

   ![](https://example.com/image.png)

<table>
<thead>
<tr>
<th>File</th>
<th>Stream Configuration</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>events.csv</td>
<td>org.wso2.event.sensor.stream:1.0.0</td>
<td>Pay</td>
</tr>
</tbody>
</table>

   5. You view the WSO2 events that are published to the CEP server in the logs of the consumer terminal as shown below.

   ![](https://example.com/image.png)

   **Sample 0064 - Publishing Text Events via Email Transport**

   - **Introduction**
   - **Prerequisites**
   - **Building the sample**
   - **Executing the sample**

   **Introduction**

   This sample demonstrates how to publish outgoing text events via email transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

   **Prerequisites**

   Set up the prerequisites required for all samples.

   Edit the mail transport sender configuration in `<CEP_HOME>/repository/conf/output-event-adapters.xml` file as required to send events to email client.
<adapterConfig type="email">
  <property key="mail.smtp.from">abcd@gmail.com</property>
  <property key="mail.smtp.user">abcd</property>
  <property key="mail.smtp.password">xxxx</property>
  <property key="mail.smtp.host">smtp.gmail.com</property>
  <property key="mail.smtp.port">587</property>
  <property key="mail.smtp.starttls.enable">true</property>
  <property key="mail.smtp.auth">true</property>
</adapterConfig>

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0064. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- A stream named `org.wso2.event.sensor.stream:1.0.0`.
- An event publisher named `emailPublisher` to log the received messages

Executing the sample

This builds the publishes the events in the `<CEP_HOME>/samples/cep/artifacts/0064/eventsimulatorfiles/events.csv` file to the `emailPublisher` endpoint.

First go to the Event publishers and update existing `emailPublisher` and change email property for receiving mail address as shown below.
Then go to the event simulator and click play to send events contain in events.csv file

You view the receiver email should receive events as following screenshot.
WSO2 CEP - Sensor information

wso2cloudtest@gmail.com
to me

meta_timestamp:199008131285,
meta_isPowerSaverEnabled:true,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545

wso2cloudtest@gmail.com
to me

meta_timestamp:199008131345,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,

wso2cloudtest@gmail.com
to me

meta_timestamp:199008131325,
meta_isPowerSaverEnabled:true,
meta_sensorId:102,
Sample 0065 - Publishing JSON Events via SMS Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

**Introduction**

This sample demonstrates how to publish JSON events via SMS transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and an SMSC simulator.

**Prerequisites**

Follow the steps below to complete the prerequisites before starting the configuration.

1. Add the following configuration under Transport Outs section in the `<CEP_HOME>/repository/conf/axis2[axis2_client.xml` file, to enable SMS transport.
<axisconfigname="AxisJava2.0">
  ...
  <transportSender class="org.apache.axis2.transport.sms.SMSSender"
  name="sms">
    <parameter name="systemType"/>
    <parameter name="systemId">cep1</parameter>
    <parameter name="password">cep123</parameter>
    <parameter name="host">localhost</parameter>
    <parameter name="port">2775</parameter>
    <parameter name="phoneNumber">CEP1</parameter>
  </transportSender>
  ...
</axisconfig>

2. Configure WSO2 CEP by adding relevant jars to support SMS transport.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0065. For instructions, see Starting sample CEP configurations. This sample configuration creates the following.

- An event stream named org.wso2.event.sensor.stream:1.0.0.
- An event publisher named smsEventPublisher to fetch events from the configured receiver email address
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0065/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample. This sample uses Logica SMSC Simulator to receive messages. To start messaging simulator clone this repository and follow ReadMe to build it.

1. Navigate to build/jar directory in SMSC Simulator repository. The folder must contain following two files.
   
   
   
   smcsim.jar
   users.txt

2. Add the following name-value pairs to users.txt file.

   Enter the value of the systemId parameter defined in the above SMS transport sender configuration as the value of the name parameter in the below list. Please note that white spaces aren’t removed from neither attribute name and it’s value, i.e. "name=peter" and "name= peter" define two different users, "peter" and " peter".

   name=cep1
   password=cep123
   timeout=unlimited

3. Start SMSC Simulator by executing the following command: java -cp
   smcsim.jar:../../lib/smpp.jar com.logica.smcsim.Simulator

4. In the simulator console where the command runs:
4. Enter 1 for the prompt to start simulation.
5. Enter 2775 as the port number (this port is equal to the port defined in the SMS transport sender configuration.)

When the `Starting listener... started` log is displayed on the console, the SMSC simulator is ready to accept messages as shown below.

```
Copyright (c) 1996-2001 Logica Mobile Networks Limited
This product includes software developed by Logica by whom copyright
and know-how are retained, all rights reserved.
```

- 1 start simulation
- 2 stop simulation
- 3 list clients
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 8 exit

Enter port number> 2775
Starting listener... started.

5. Start the WSO2 CEP server with the sample configuration numbered 0065. For instructions, see Starting sample CEP configurations.

6. Click Tools, and then click Event Simulator.
7. Click Play on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv` file to the publisher.

```
<table>
<thead>
<tr>
<th>Input Data by File</th>
<th>switch to configure database for simulation</th>
</tr>
</thead>
</table>
```

You view the JSON events that are published by the CEP server in the logs of the simulator as shown below.

```
Sample 0066 - Publishing JSON Events via MQTT Transport
```
Introduction

This sample demonstrates how to publish JSON events via MQTT transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Follow the steps below before starting the output MQTT event publisher configuration.

1. Set up the prerequisites required for all samples.
2. Configure WSO2 CEP by adding relevant jars to support MQTT transport.
3. Configure sample client by adding relevant jars. See setting up MQTT for MQTT sample clients.
4. Start the MQTT-supported server. (E.g. Mosquitto)

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0066. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates a stream named org.wso2.event.sensor.stream:1.0.0.
- Creates an event publisher named mqttEventPublisher.
- Load the events stored in the <CEP_HOME>/samples/cep/artifacts/0066/eventsimulatorfiles/events.csv file to the event simulator.

Executing the sample

Follow the steps below to execute the sample.

1. Navigate to the <CEP_HOME>/samples/cep/consumers/mqtt/ directory, and execute the following Ant command using another tab in the CLI: ant -Durl=tcp://localhost:1883 -Dtopic=sensordata -Dsn=066

   The other optional parameters that can be used in the above command are defined in the <CEP_HOME>/samples/cep/consumers/mqtt/build.xml file.

   This builds the MQTT client which fetches the published events in the <CEP_HOME>/samples/cep/artifacts/0066/eventsimulatorfiles/events.csv file from the eventPublisher endpoint as shown below.

   ![ MQTT client executing ]

2. Log in to the management console.
3. Click Tools, and then click Event Simulator.
4. Click Play on the corresponding event stream as shown below, to send the events in the <CEP_HOME>/samples/cep/artifacts/0066/eventsimulatorfiles/events.csv file to the publisher.
Sample 0067 - Publishing Map Events via Cassandra Transport

- **Introduction**
  This sample demonstrates how to publish incoming events to Apache Cassandra using the Cassandra event publisher of CEP. This sample does not do any processing on the incoming event. Use the pre-packaged Cassandra-CLI tool of Apache Cassandra, to verify if the messages are properly published.

- **Prerequisites**
  Set up the prerequisites required for all samples.

- **Building the sample**
  Start the WSO2 CEP server with the sample configuration numbered 0067. For instructions, see Starting sample CEP configurations. This sample configuration does the following:
  - Changes the default Axis2 repo from `<CEP_HOME>/repository/deployment/server` to `<CEP_HOME>/samples/cep/artifacts/0067`.
  - Creates an event stream named `org.wso2.event.transaction.stream:1.0.0`.
  - Loads the `events.csv` and related configuration files so that event simulator can be used to send events to the pre-configured event publisher.
  - Creates an event publisher named `cassandra` to send the received messages to a Cassandra cluster. This event publisher will publish events in Map (key,value pair) event format.

- **Executing the sample**
  1. Setup a Apache Cassandra cluster. The sample is configured to send events to Cassandra cluster that is configured to listen on localhost:9160 port. (If the Cassandra cluster is running on a different node, please change the event publisher configuration as necessary). The Cassandra event publisher needs the following configurations to be valid in the Cassandra cluster.
1. Cluster Name: Test Cluster

2. Keyspace: CEP_KS

3. Column Family: CF_Transactions (Comparator should be of type 'UTF8Type' or 'AsciiType')

4. Connection Username: admin

5. Connection Password: admin

You can use the Cassandra CLI client that is packaged with Apache Cassandra to create the keyspace and column family as necessary. The following CLI commands have been tested with Apache Cassandra v1.2.19

   - Create a keyspace: create keyspace CEP_KS;
   - Use keyspace: use CEP_KS admin 'admin';
   - Create column family: create column family CF_Transactions with comparator = 'UTF8Type' and caching='ALL';

2. Go to the management console of CEP and select org.wso2.event.transaction.stream:1.0.0, and select 'Play' to play the events.

3. You can see the events getting received by CEP by executing the command 'list CF_Transactions;' in Cassandra CLI.
Sample 0068 - Publishing XML Events via Kafka Transport

- Introduction
- Prerequisites
- Building the sample
- Executing the sample

Introduction

This sample demonstrates how to publish XML events via Kafka transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

Prerequisites

Follow the steps below before starting the output Kafka event publisher configuration.

1. Set up the prerequisites required for all samples.
2. Configure WSO2 CEP by adding relevant jars to support Kafka transport.
3. Configure sample client by adding relevant jars. See Setting up Kafka for Kafka sample clients.
4. Start the Apache ZooKeeper server with the following command: `bin/zookeeper-server-start.sh config/zookeeper.properties`. You view the below logs. For more information, see Apache Kafka documentation.
Then start the Kafka server with the following command: `bin/kafka-server-start.sh config/server.properties`. You view the below logs.

5. Then start the Kafka server with the following command: `bin/kafka-server-start.sh config/server.properties`. You view the below logs.

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0068. For instructions, see Starting sample CEP configurations. This sample configuration creates the following:

- An event stream with the ID `org.wso2.event.sensor.stream:1.0.0`
- An event publisher named `kafkaEventPublisher` to fetch events from the configured receiver email address
- Loads the events stored in the `<CEP_HOME>/samples/cep/artifacts/0068/eventsimulatorfiles/events.csv` file to the event simulator.
**Executing the sample**

Follow the steps below to execute the sample.

1. Open another terminal, navigate to `<KAFKA_HOME>/` directory, and execute the following command to consume messages which are published from the CEP via the Kafka event publisher: `bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic sensorInfo --from-beginning`

2. Log in to the management console.

3. Click **Tools**, and then click **Event Simulator**.

4. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv` file to the publisher.

5. Execute the following command to view the output logs in the terminal of the Kafka server: `bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic sensorInfo --from-beginning`

You view the XML events that are published to the CEP server in the logs of the Kafka consumer terminal as shown below.

---

**Sample 0069 - Publishing JSON Events via WebSocket Transport**

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates how to publish custom JSON events via WebSocket transport. This sample does not do any processing on the outgoing event. Events are generated using an input event file and the event simulator.

**Prerequisites**

Follow the steps below before starting the output websocket event publisher configuration.

1. Set up the prerequisites required for all samples.

**Building the sample**

Start the WSO2 CEP server with the sample configuration numbered **0069**. For instructions, see **Starting sample CEP configurations**. This sample configuration creates the following.

- An event stream named `org.wso2.event.sensor.stream:1.0.0`.
- An event publisher named `wsPublisher` to fetch events from the configured receiver email address.
- Loads the events stored in the `<CEP_HOME>/samples/cep/artifacts/0069/eventsimulatorfiles/events.csv` file to the event simulator.

**Executing the sample**
Follow the steps below to execute the sample.

1. Navigate to the `<CEP_HOME>/samples/cep/consumers/websocket/` directory, and execute the following Ant command using another tab in the CLI:
   ```
   ant -Dport=9099 -Dsn=0069
   ```

   The other optional parameters that can be used in the above command are defined in the `<CEP_HOME>/samples/cep/consumers/websocket/build.xml` file.

   This builds the WebSocket client which fetches the published events in the `<CEP_HOME>/samples/cep/artifacts/0069/eventsimulatorfiles/events.csv` file from the `wsPublisher` endpoint as shown below.

2. Log in to the management console.

3. Click **Tools**, and then click **Event Simulator**.

4. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0069/eventsimulatorfiles/events.csv` file to the publisher.

   ![Event Simulator](image)

   You view the WSO2 events that are published to the CEP server in the logs of the consumer terminal as shown below.

   ![Log Output](image)

Sample 0070 - Publishing JSON Events via Websocket-Local Output Event Adapter

- **Introduction**
- **Prerequisites**
- **Starting sample CEP configurations**
- **Connect to CEP via a Websocket client (to receive events)**

**Introduction**

This sample demonstrates how the WSO2 CEP can publish events to a Websocket client.

Refer to below image to get an overview of the event flow.
Refer to the below steps, which we are going to do in this sample:

- Step 1: Start CEP server. This will start a Websocket server which is in-built to CEP.
- Step 2: Define an Event Stream in CEP.
- Step 3: Create an Output Websocket-local Event Adapter, so events of type defined in step 1 will be published via the CEP Websocket server.
- Step 4: Connect a Websocket client to the Websocket Server in CEP.
- Step 5: Send a few events to the CEP. To do this, we will use the Event Simulator, in-built to CEP. As the events are sent, Websocket client will receive the same.

**Prerequisites**

See Prerequisites in CEP Samples Setup page.

In addition to that, go to `<CEP_HOME>/samples/cep/utils/output-websocket-local-adapter` folder and execute the following command:

```
ant -Dsn=0070
```

This will copy outputwebsocket.war webapp to `<CEP_HOME>/sample/artifacts/0070/webapps` folder.

**Starting sample CEP configurations**

Start the WSO2 CEP server with the sample configuration numbered 0070 For instructions, see Starting sample CEP configurations.

This sample configuration points the default Axis2 repo to `<CEP_HOME>/samples/cep/artifacts/0070` (by default, the Axis2 repo is `<CEP_HOME>/repository/deployment/server`). As a result, the artifacts in `<CEP_HOME>/sample/artifacts/0070` folder will get deployed.

When these sample configurations are deployed:

- An event stream will be defined.
- The event simulator will be ready to play a set of events to CEP.
- An Output Websocket-local Event Adapter will be created.

In other words, all the first three steps listed in Introduction section will be done, when the server is started with sample configuration number 0070.

Now the remaining tasks are to execute steps 4 and 5: connect a Websocket client, and send a few events to the CEP.
CEP.

**Connect to CEP via a Websocket client (to receive events)**

In this sample, you will use your web browser as the Websocket client. However, any Websocket client can be used for this purpose.

**Step I:** Start a web browser and go to its Javascript console. In most browsers, such as Chrome and Firefox, you can load the Javascript console by simply pressing the keys Ctrl+Shift+J (or Cmd+Shift+J on a Mac).

Ensure that the page on which you open the Javascript Console is loaded over HTTP. For example, you can open the page [http://ws02.com](http://ws02.com) and load the Javascript console on that page.

**Step II:** On the browser console, copy and paste the following Javascript code block and press enter:

```javascript
var ws = new WebSocket("ws://localhost:9763/outputwebsocket/wsLocalOutputAdapter");
ws.onopen = function() {
  console.log("opened");
};
ws.onmessage = function (evt) {
  alert("Message: " + evt.data);
};
ws.onclose = function() {
  console.log("closed!");
};
ws.onerror = function(err) {
  console.log("Error: " + err);
};
```

Note: If you have started CEP server on a different host and a port, replace ‘localhost’ in the above command with that host, and ‘9763’ with that port, respectively. See Running the Product page for more details.

This step is shown in the screenshot below.
Upon successful connection, you will see an output in the same console as shown in the below screenshot.

```javascript
var ws = new WebSocket("ws://localhost:9763/outputwebsocket/wsLocalOutputAdapter");

ws.onopen = function() {
    console.log("opened");
};

ws.onmessage = function (evt) {
    alert("Message: " + evt.data);
};

ws.onclose = function() {
    console.log("closed!");
};

ws.onerror = function(err) {
    console.log("Error: " + err);
};
```

Upon successful connection, you will see an output in the same console as shown in the below screenshot.

```
09:34:51.313 var ws = new WebSocket("ws://localhost:9763/outputwebsocket/wsLocalOutputAdapter");
ws.onopen = function() {
    console.log("opened");
};
ws.onmessage = function (evt) {
    alert("Message: " + evt.data);
};
ws.onclose = function() {
    console.log("closed!");
};
ws.onerror = function(err) {
    console.log("Error: " + err);
};
```

```
09:34:51.317 [object Function]
09:34:51.365 GET http://localhost:9763/outputwebsocket/wsLocalOutputAdapter [HTTP/1.1 101 Switching Protocols 77ms]
09:34:51.400 "opened"
```
Now this browser-client is ready to receive events from the CEP.

**Step III:** Next and final step is to send four events to the CEP. We will send the following four events of type `org.wso2.event.sensor.stream` to the CEP.

```
199008131245,false,100,temperature,23.45656,7.12324,100.34,23.4545
199008131245,true,101,temperature,23.45656,7.12324,100.34,23.4545
199008131245,false,103,temperature,23.45656,7.12324,100.34,23.4545
199008131245,true,104,temperature,23.45656,7.12324,100.34,23.4545
```

For sending events, we will be using the CEP event simulator.

To use the event simulator, [log in to CEP management console](#), go to **Tools --> Event Simulator**.

In the Event Simulator page, under 'Send multiple events', you will see an input data file being ready to be played. Once we click 'play', the events included in this data file will be sent to the CEP.

Click 'play' button, as shown in the screenshot below.

![Send multiple events](image)

As the events are being played, you will see four alerts from your browser, indicating the receipt of events. One such alert is shown in the image below.

![JavaScript Application](image)

**Sample 0071 - Publishing WSO2Event Events via UI Transport**

- **Introduction**
- **Prerequisites**
- **Building the sample**
- **Executing the sample**

**Introduction**

This sample demonstrates the usage of the UI publisher using a client app. This sample does not have Siddhi queries because it does not not use any execution plans.
Prerequisites

Set up the prerequisites required for all samples

Building the sample

Start the WSO2 CEP server with the sample configuration numbered 0071. For instructions, see Starting sample CEP configurations. This sample configuration does the following.

- Creates an event stream named `org.wso2.event.sensor.stream:1.0.0`.
- Creates an event publisher named `uiPublisher`.
- Loads the events stored in the `<CEP_HOME>/samples/cep/artifacts/0071/eventsimulatorfiles/events.csv` file to the event simulator.
- Deploys a Jaggery application named `outputuitest`. This application functions as the client that consumes the events published by the UI publisher.

Executing the sample

Follow the steps below to execute the sample.

1. Access the Analytics Dashboard using the following URL.
   https://localhost:9443/portal/
2. Access the following page.
   https://localhost:9443/portal/outputuitest/index.jag

This opens the following form with the fields filled in with default values as shown below.

The fields in the form are described in the table below.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Name</td>
<td>The name of the event stream to which you want to subscribe. When events with values for the attributes defined in this stream are received by the Output Adapter UI test application.</td>
<td><code>org.wso2.event.sensor.stream</code></td>
</tr>
<tr>
<td>Version</td>
<td>The version of the event stream to which you want to subscribe.</td>
<td><code>1.0.0</code></td>
</tr>
</tbody>
</table>
2. **CEP Host**
   The host on which the CEP server (which contains the UI publisher) runs.
   
   **localhost**

3. **CEP Port**
   The port on which the CEP server (which contains the UI publisher) runs.
   
   **9443**

4. **Mode**
   UI publisher supports two types of subscriptions named Websocket and HTTP. If you select Websocket for this field, the client application receives events via the Websocket transport. If you select HTTP for this field, the client application receives events via the HTTP transport.

   When the **HTTP** subscription is used, the events are polled from the UI publisher at a constant interval, and this interval needs to be specified in the **Polling Interval** field.

5. **Polling Interval**
   The time interval at which events should be polled from the UI publisher. This time interval is specified in seconds.

   **30**

   This field value is only applicable is the value specified in the **Mode** field is **HTTP**.

3. Click **Connect**. Now the client app is ready to receive events. Next step is sending events so the UI publisher receives them and publishes them to the client application.

4. Log into the WSO2 CEP Management Console.

5. In the **Tools** tab, click **Event Simulator**.

6. Click **Play** on the corresponding event stream as shown below to send the events in the `<CEP_HOME>/samples/cep/artifacts/0071/eventsimulatorfiles/events.csv` file to the publisher.

   The output events published from WSO2 CEP and received by the client Jaggery application are logged as follows.
Sample 0072 - Publishing Map Events via RDBMS Transport

- Introduction
- Prerequisites
- Starting the server
- Executing the sample
- Output of the sample

Introduction

This sample demonstrates how incoming events are stored in a H2 database using output RDBMS event adapter in two execution modes, which are insert and update-insert. This sample uses the default H2 database, which is shipped with WSO2 CEP.

The `<CEP_HOME>/repository/conf/output-event-adapters.xml` file includes configuration details of the supported RDBMSs.

Prerequisites

Follow the steps below to set up the prerequisites before starting the configurations.

1. Set up the general prerequisites that are applicable to all WSO2 CEP Samples.
2. Enable the following H2 database configurations by uncommenting them in the `<CEP_HOME>/repository/conf/carbon.xml` file as follows, to browse through the database and see the changes.

Keep the other properties of the `H2DatabaseConfiguration` element uncommented.
Starting the server

Start the WSO2 CEP server with the sample configuration numbered 0072. For instructions on starting WSO2 CEP server with a sample configuration, see Starting sample CEP configurations. This sample configuration points the default Axis2 repository to `<CEP_HOME>/samples/cep/artifacts/0072/` directory. (By default, the Axis2 repository is pointed to `<CEP_HOME>/repository/deployment/server/` directory).

Executing the sample

Follow the steps below to execute the sample.

1. Log in to the management console.
2. Click **Tools**, and then click **Event Simulator**.
3. Click **Play** on the corresponding event stream as shown below, to send the events in the `<CEP_HOME>/samples/cep/artifacts/0072/eventsimulatorfiles/events.csv` file to the publisher.

4. Use the following URL to access the H2 database through your web browser: `https://localhost:8082`

5. Enter the following values to connect to the database.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Class</td>
<td></td>
<td><code>org.h2.Driver</code></td>
</tr>
</tbody>
</table>
Enter the URL of the database. For example: jdbc:h2:repository/database/WSO2CARBON_E_ON_EXIT=FALSE;LOCK_TIMEOUT=60000

User Name: wso2carbon
Password: wso2carbon

6. Click **Test Connection** to check if the connection is successful.

7. If the connection is successful, click **Connect**.

**Output of the sample**

After simulating events a new table (SENSORDATA) will be visible on the console as shown below.

The table corresponds to the following execution mode:

- SENSORDATA - Contains records of **insert** execution mode. Mode does not consider primary keys and keeps on inserting values as per the events.

You can observe the results of the created table as shown below.
User Guide

The following diagram summarizes how events are processed in a CEP event flow.

1. WSO2 CEP receives events from external event sources and data agents. These events have different formats such as XML, JSON, Map, Text, etc.
2. WSO2 CEP supports a range of event receiver types to receive events in different formats. Each event receiver receives the events in a specific format based on its type, and converts them to tuples which enable WSO2 CEP to process them.
3. The converted event tuples are forwarded to the event stream connected to the event receiver. Event tuples are associated with their respective event streams based on the event stream ID.
4. The event stream in turn forwards the events to the connected execution plan (i.e. the execution plan with queries that define the same event stream as the input stream). The events are also forwarded to the event publishers in order to be published in external systems.
5. An execution plan processes the events in the input event streams via Siddhi queries written in the Siddhi Query Language. The events produced via output streams after processing the Siddhi queries of the execution plan are forwarded back to the event streams.
6. WSO2 CEP supports a range of event publisher types to publish processed events in a range of interfaces which require different formats such as JMS, KAFKA, SOAP etc. Each publisher converts the events into a specific format depending on its type before publishing the events in the relevant interface. The interface in which events are published can be dashboards, external event sinks or data stores.

The components of the event flow described in the above summary are explained in detail in the following sections.

- Understanding Event Streams
- Collecting Data
- Analyzing Data
- Communicating Results
- Packaging Artifacts as C-App Archive
- Monitoring and Debugging
- Managing CEP Artifacts Via the Template Manager

Understanding Event Streams

Events are the lifeline of WSO2 CEP/DAS. They not only process data as events, but also interact with external
systems using events. Event is a unit of data, and an event stream is a sequence of events of a particular type. The type of events can be defined as an event stream definition. The following sections explain how to work with events in WSO2 CEP.

- Event streams
- Event formats
- Event Flow

Event streams

You can manage event streams through event stream definitions.

- Event stream definition
- Adding an event stream
- Using the source view
- Deleting an event stream
- Editing an event stream
- Creating sample events

Event stream definition

Definitions of the event streams are stored in the filesystem as deployable artifacts in the `<PRODUCT_HOME>/repository/deployment/server/eventstreams/` directory as `.json` files. These are hot deployable files and can be added/removed when the server is up and running. A sample event stream definition is as follows.

```json
{
    "streamId": "org.wso2.test:1.0.0",
    "name": "org.wso2.test",
    "version": "1.0.0",
    "nickName": "TestStream",
    "description": "Test Stream",
    "metaData": [
        {
            "name": "ip",
            "type": "STRING"
        }
    ],
    "correlationData": [
        {
            "name": "id",
            "type": "LONG"
        }
    ],
    "payloadData": [
        {
            "name": "testMessage",
            "type": "STRING"
        }
    ]
}
```

The properties of the above event stream definition are described below.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream Name</td>
<td>Name of the event stream.</td>
</tr>
<tr>
<td>Event Stream Version</td>
<td>Version of the event stream. (Default value is 1.0.0.)</td>
</tr>
<tr>
<td>Event Stream Description</td>
<td>Description of the events stream. (This is optional.)</td>
</tr>
<tr>
<td>Event Stream Nick-Name</td>
<td>Nick-names of an event streams separated by commas. (This is optional.)</td>
</tr>
<tr>
<td>Stream Attributes</td>
<td>Stream Attributes contains the data of the event. These are divided into three logical separations to give more usability and maintenance to the user. It is not mandatory to have attributes in all three sections, but there should be at least one section with at least one attribute defined. Also attribute names should be unique within each section.</td>
</tr>
</tbody>
</table>

- **Meta Data**: Contains the meta information of the events. (Referred to as `meta_<attribute name>`.)
- **Correlation Data**: Contains the correlation information of the events. (Referred to as `correlation_<attribute name>`.)
- **Payload Data**: Contains the actual data that the event intends to have. (Referred to as `<attribute name>`.)

It is recommended to have logistic separation but the internal system does not differentiate or give privilege based on above separations. Also please note if you edit and add another attribute to existing stream definition, other artifacts which are associated with the stream will be inactive. So that it is recommended to create a new stream definition with a version including the additional attribute.

For example consider following attributes which exist in a single event.

- `event_timestamp`, `request_IP_address`, `correlation_Id`, `price`, `symbol` with their corresponding data types.

We can logically separate the above attributes as following.

- **Meta Data**: `event_timestamp`, `request_IP_address` (These describes the events itself)
- **Correlation Data**: `correlation_Id` (These correlates events with other events from other streams and will be useful when you perform a join operations on a stream)
- **Payload Data**: `price`, `symbol` (These are actual information/details of the event)

### Adding an event stream

You can create an event stream by creating a new event stream definition using the design view or the source view.

### Using the design view

Follow the steps below to add an event stream using the design view.

1. Log in to the management console, and click **Main**.
2. Click Event Streams in the Event Processor menu, and then click Add Event Stream.
3. Enter details of the stream definition that you want to create as shown in the below example.

4. Click Add Event Stream, to create the Event Stream in the system. When you click OK in the pop-up message on successful addition of the stream definition, you view it in the Available Event Streams list as shown below.

Using the source view

Follow the steps below to add an event stream using the source view.

1. Log in to the management console, and click Main.
2. Click Event Streams in the Event Processor menu, and then click Add Event Stream.
3. Click switch to source view.

Click switch to design view to add the event stream using the design view.
4. Enter details of the stream definition that you want to create as shown in the below example.

```json
{
    "streamId": "org.wso2.test:1.0.0",
    "name": "org.wso2.test",
    "version": "1.0.0",
    "nickName": "TestStream",
    "description": "Test Stream",
    "metaData": {
        "name": "ip",
        "type": "STRING"
    },
    "correlationData": {
        "name": "id",
        "type": "LONG"
    },
    "payloadData": {
        "name": "testMessage",
        "type": "STRING"
    }
}
```

5. Click **Add Event Stream**, to create the Event Stream in the system. You view the new event stream in the **Available Event Streams** list as shown below.

---

**Deleting an event stream**

Follow the steps below to delete an event stream by deleting the corresponding event stream definition.

1. Log in to the management console, and click **Main**.
2. Click Event Streams in the Event Processor menu. You view the Available Event Streams list.
3. Click the Delete button of the corresponding event stream to delete it.

**Editing an event stream**

Follow the steps below to edit an event stream by editing the corresponding event stream definition.

1. Log in to the management console, and click Main.
2. Click Event Streams in the Event Processor menu. You view the Available Event Streams list.
3. Click the Edit button of the corresponding event stream to edit it.

Click the switch to source view link to edit an event stream using the source view.

**Creating sample events**

Follow the steps below to create sample events for a defined event stream.

1. Log in to the management console, and click Main.
2. Click Event Streams in the Event Processor menu. You view the Available Event Streams list.
3. Click the Event Stream Id of the corresponding event stream for which you want to create the sample event.
4. Select the event format type (i.e. xml, json, or text) from the drop down list, which you want to create the sample event in.
5. You view details of the event stream as shown below.

![Create Sample Event](image)

6. Click Generate Event to create the sample event.

**Event formats**

WSO2 CEP/DAS facilitates the following default and custom event formats.

- Default event formats
- Custom event formats

**Default event formats**

By default, WSO2 CEP/DAS represents an event as a WSO2Event object. Furthermore, WSO2 CEP/DAS supports events in XML, JSON, Text and Map formats. The default event formats of the XML, JSON, Text and Map representations for the following sample event stream definition are as follows.

**Sample event stream definition**
{  
    "streamId": "org.wso2.test:1.0.0",
    "name": "org.wso2.test",
    "version": "1.0.0",
    "nickName": "TestStream",
    "description": "Test Stream",
    "metaData": [  
      {  
        "name": "ip",
        "type": "STRING"
      }  
    ],
    "correlationData": [  
      {  
        "name": "id",
        "type": "LONG"
      }  
    ],
    "payloadData": [  
      {  
        "name": "testMessage",
        "type": "STRING"
      }  
    ]
}

- Default XML format
- Default JSON format
- Default text format
- Default map format

<events>
  <event>
    <metaData>
      <ip>data4</ip>
    </metaData>
    <correlationData>
      <id>56783</id>
    </correlationData>
    <payloadData>
      <testMessage>data1</testMessage>
    </payloadData>
  </event>
</events>
Custom event formats

If you receive and publish events with a different format than the default format, you need to provide appropriate mappings for the system to interpret the events.

**Custom formats for receiving events**

For information on the custom event receiver mappings, see [Input Mapping Types](#).

**Custom formats for publishing events**

For information on the custom event publisher mappings, see [Output Mapping Types](#).

**Event Flow**

Event flow visualizes the stream flow in WSO2 CEP/DAS to easily navigate to different WSO2 CEP/DAS components.

Follow the steps below to view the event flow.

1. Log in to the management console.
2. Click **Main**, and then click **Event Flow** in the **Event Processor** menu.
This demonstrates how all the active event receivers, event streams, event publishers, and execution plans are connected.

Collecting Data

The first step in complex event processing is to collect the data to be processed. Data is obtained from data agents. This data is captured from external applications as well as from WSO2 Enterprise Server Bus, WSO2 Application Server and other custom data publishers for which WSO2 CEP acts as a summarization server.

This section covers the following topics.

- Publishing Data to CEP
- Configuring CEP to Receive Data

Publishing Data to CEP

The following topics cover the different ways in which data is sent to WSO2 CEP in the form of events.

- Publishing Data Using Event Simulation
- Publishing Binary/Thrift Data using Java Client
- Publishing JSON/REST Data using Java Client
- Publishing Data Through Other Protocols

Publishing Data Using Event Simulation

Event simulator tool is used to simulate predefined event streams. These event stream definitions have stream attributes. You can use event simulator to create events by assigning values to the defined stream attributes and send them as events. This tool is useful for debugging and monitoring the event receivers and publishers, execution plans and event formatters. The events are sent to the component (e.g. to an event receiver, event publisher, execution plan etc.) that is defined in the event stream.
There are two ways of simulating an event flow as shown below.

- **Sending a Single Event by Entering Data**
- **Sending Multiple Events**
  - Sending Multiple Events Using a File
  - Sending Multiple Events Using a Datasource
  - Sending Multiple Events via XML/JSON

### Sending a Single Event by Entering Data

Follow the steps below to send a single event to a defined event stream by entering data via the event simulator.

1. Log in to the management console using the following URL: https://<PRODUCT_HOST>:<PRODUCT_PORT>/carbon/
2. Click **Tools**, and then click **Event Simulator**.
3. Select the **Event Stream Name** from the drop down list.
4. Enter the attribute values accordingly as shown in the below example.

Use either 'true' or 'false' as values for boolean type attributes that are defined in an event stream, when sending events via the event simulator.
5. Click **Send**, to send the events to the event stream.

### Sending Multiple Events

You can send multiple events to a defined event stream using different methods as described in the following sections.

- **Sending Multiple Events Using a File**
- **Sending Multiple Events Using a Datasource**
- **Sending Multiple Events via XML/JSON**

#### Sending Multiple Events Using a File

You can insert all the data for a particular event stream to a **CSV** file including values separated by an appropriate separator (e.g. comma, slash, etc). You need to enter a new dataset for a new event, after a newline character. For example, the following CSV file includes data for four events.

<table>
<thead>
<tr>
<th>TimeStamp</th>
<th>IsPowerSaved</th>
<th>Temperature</th>
<th>PowerUnit</th>
<th>Value</th>
<th>Radius</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>199008131245</td>
<td>false</td>
<td>100</td>
<td>temperature</td>
<td>23.45656</td>
<td>7.12324</td>
<td>100.34</td>
</tr>
<tr>
<td>199008131245</td>
<td>true</td>
<td>101</td>
<td>temperature</td>
<td>23.45656</td>
<td>7.12324</td>
<td>100.34</td>
</tr>
<tr>
<td>199008131245</td>
<td>false</td>
<td>103</td>
<td>temperature</td>
<td>23.45656</td>
<td>7.12324</td>
<td>100.34</td>
</tr>
<tr>
<td>199008131245</td>
<td>true</td>
<td>104</td>
<td>temperature</td>
<td>23.45656</td>
<td>7.12324</td>
<td>100.34</td>
</tr>
</tbody>
</table>

Use either 'true' or 'false' as values for boolean type attributes that are defined in an event stream, when sending events via the event simulator.

Follow the steps below to send multiple events to a defined event stream using a CSV file via the event simulator.
1. Log in to the product management console using the following URL: https://<PRODUCT_HOST>:<PRODUCT_PORT>/carbon/
2. Click **Tools**, and then click **Event Simulator**.
3. Select the **Event Stream Name** from the drop down list.
4. Browse and upload the CSV file. It will be hot deployed in the server. Refresh the page to view the uploaded file.
5. Click **Configure**, to configure the CSV file to specify the field delimiter before simulating the event flow as shown below.

   ![CSV File Configuration](image)

6. Configure the CSV file by entering the field delimiter as shown below.

   ![CSV File Configuration](image)

7. Click **Play**, to simulate the event flow.

   ![Event Simulation](image)

   Click the corresponding **Delete** button to, delete the uploaded CSV file along with its configurations. Other than that you can pause, stop and resume event publishing by using corresponding buttons as well.

---

**Sending Multiple Events Using a Datasource**

Follow the steps below to send multiple events to a defined event stream using a datasource via the event simulator.

1. Create a datasource. For instructions on creating a datasource, see [Configuring an RDBMS Datasource](#).
2. Log in to the product management console using the following URL: https://<PRODUCT_HOST>:<PRODUCT_PORT>/carbon/
3. Click **Tools**, and then click **Event Simulator**.
4. Select the **Event Stream Name** from the drop down list.
5. Click **switch to add configuration for simulate by database**.

   ![Datasource Configuration](image)
6. Enter the datasource information to be used when simulating the events, as shown below.

[Image: Screenshot of the datasource configuration interface]

Column types in the table of the datasource should match with the event stream attributes. Also, use either `true` or `false` as values for boolean type attributes that are defined in an event stream, when sending events via the event simulator.

7. Click **Test Connection**, and then click **Save**, to test the connection and save it. When the configuration file gets hot deployed, click **Play** to simulate sending events via the database as shown below.

[Image: Screenshot of the event simulator interface]

**Sending Multiple Events via XML/JSON**

Follow the procedure below to send multiple events to WSO2 CEP via XML/JSON.

1. Create an event receiver of any receiver type that supports XML/JSON mapping. The following is a list of such receiver types.
   - Email Event Receiver
   - HTTP Event Receiver
   - JMS Event Receiver
   - Kafka Event Receiver
   - MQTT Event Receiver
   - SOAP Event Receiver (supports only XML mapping)
   - WebSocket Event Receiver
   - WebSocket Local Event Receiver

2. Define input mapping for the event receiver you created.
   - If the message format you selected for the receiver is `xml`, specify an appropriate parent selector path to be considered the top XML element. As a result, if the events received have multiple sub elements within the element specified as the parent selector path, they are considered as multiple events.

   For detailed instructions to define XML mapping, see **Input Mapping Types - XML input mapping**.

   - If the message format you selected for the receiver is `json`, define JSON mapping for the receiver. As a result, when events received are in JSON arrays, the objects within the arrays are considered as
For detailed instructions to define JSON mapping, see Input Mapping Types - JSON input mapping.

**Publishing Binary/Thrift Data using Java Client**

- Introduction to data publisher
- Custom fields with data stream
- Dependencies
- Configuring the data agent
- Data publisher sample

**Introduction to data publisher**

A data publisher allows you to send data to a predefined set of data fields in a DAS/CEP server. The data structure with predefined fields is defined in an event stream. The data is converted to the format defined by the event stream and sent via the WSO2 data-bridge component. You can also send custom key-value pairs with data events.

**Custom fields with data stream**

The data bridge data agent has a map data structure that enables you to send an arbitrary number of string key-value pairs. The other data structures are the three object arrays corresponding to the key-value pairs of metadata, correlation data, and payload data of fixed stream definitions. You can change the key-value pairs in the map data structure from message to message, but they all should be of the string data type.

You can put the data types of these custom key-value pairs into three groups according to the transmission category.

1. When the key starts with `meta`: The data field is considered as a metadata custom field. It is sent with metadata and referred as `meta_key_prefix`.
2. When the key starts with `correlation`: The data field is considered as a correlation data custom field. It is sent with correlation data and referred as `correlation_key_prefix`.
3. When the key starts with `payload` or any other string: The data field is considered as a payload data custom field. It is sent with payload data and saved in referred as key prefix.

**Dependencies**

In order to publish data to WSO2 DAS/CEP through a custom data agent, you need to have the following dependencies. You can configure the dependencies either using the class path or using the POM file.

**Adding dependencies using class path**

Add the JAR files listed below to your class path. Note that `${carbon.analytics-commons.version}` refers to the version of the carbon analytics commons github repository - https://github.com/wso2/carbon-analytics-common/. It is always recommended to use the jar file from the latest released version.

- org.wso2.carbon.logging_4.3.0.jar
- commons-pool-1.5.6.wso2v1.jar
- google-collect_1.0.0.wso2v2.jar
- org.wso2.carbon.utils_4.3.0.jar
- org.wso2.carbon.base_4.3.0.jar
- axiom_1.2.11.wso2v5.jar
Adding dependencies using POM file

Alternatively, add the following Maven project dependency entries to your POM file. Note that `${carbon.analytics-common.version}` refers to the version of the carbon-analytics-common github repository - https://github.com/wso2/carbon-analytics-common/. It is always recommended to use the dependency entry from the latest released version.

- `httpclient-4.2.5.wso2v1.jar`
- `libthrift-0.7.0.wso2v2.jar`
- `slf4j.log4j12-1.6.1.jar`
- `slf4j.api-1.6.1.jar`
- `org.wso2.carbon.databridge.agent-${carbon.commons.version}.jar`
- `org.wso2.carbon.databridge.commons-${carbon.commons.version}.jar`
- `org.wso2.carbon.databridge.commons-${carbon.commons.version}.jar`
- `disruptor-2.10.4.wso2v2.jar`
<repositories>
  <repository>
    <id>wso2-nexus</id>
    <name>WSO2 internal Repository</name>
    <url>http://maven.wso2.org/nexus/content/groups/wso2-public/</url>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
      <checksumPolicy>ignore</checksumPolicy>
    </releases>
  </repository>
  <repository>
    <id>wso2.releases</id>
    <name>WSO2 internal Repository</name>
    <url>http://maven.wso2.org/nexus/content/repositories/releases/</url>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
      <checksumPolicy>ignore</checksumPolicy>
    </releases>
  </repository>
  <repository>
    <id>wso2.snapshots</id>
    <name>Apache Snapshot Repository</name>
    <url>http://maven.wso2.org/nexus/content/repositories/snapshots/</url>
    <snapshots>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
    </snapshots>
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>
Configuring the data agent

A data agent is a single controller for all types of data publishers created. Data publishers share resources such as client pool etc. with one data agent. Thrift data agent is available by default. You can also extend and write a new data agent such as a binary data agent.

Follow the steps below to configure a data agent.

1. Load the following sample configurations and properties to define the data agent in the JVM.
<DataAgentsConfiguration>
  <Agent>
    <Name>Thrift</Name>
    <DataEndpointClass>org.wso2.carbon.databridge.agent.internal.endpoint.thrift.ThriftDataEndpoint</DataEndpointClass>
    <TrustSore>src/main/resources/client-truststore.jks</TrustSore>
    <TrustSorePassword>ws02carbon</TrustSorePassword>
    <QueueSize>32768</QueueSize>
    <BatchSize>200</BatchSize>
    <CorePoolSize>5</CorePoolSize>
    <MaxPoolSize>10</MaxPoolSize>
    <KeepAliveTimeInPool>20</KeepAliveTimeInPool>
    <ReconnectionInterval>30</ReconnectionInterval>
    <MaxTransportPoolSize>250</MaxTransportPoolSize>
    <MaxIdleConnections>250</MaxIdleConnections>
    <EvictionTimePeriod>5500</EvictionTimePeriod>
    <MinIdleTimeInPool>5000</MinIdleTimeInPool>
    <SecureMaxTransportPoolSize>250</SecureMaxTransportPoolSize>
    <SecureMaxIdleConnections>250</SecureMaxIdleConnections>
    <SecureEvictionTimePeriod>5500</SecureEvictionTimePeriod>
    <SecureMinIdleTimeInPool>5000</SecureMinIdleTimeInPool>
  </Agent>

  <Agent>
    <Name>Binary</Name>
    <DataEndpointClass>org.wso2.carbon.databridge.agent.internal.endpoint.binary.BinaryDataEndpoint</DataEndpointClass>
    <TrustSore>src/main/resources/client-truststore.jks</TrustSore>
    <TrustSorePassword>ws02carbon</TrustSorePassword>
    <QueueSize>32768</QueueSize>
    <BatchSize>200</BatchSize>
    <CorePoolSize>5</CorePoolSize>
    <MaxPoolSize>10</MaxPoolSize>
    <KeepAliveTimeInPool>20</KeepAliveTimeInPool>
    <ReconnectionInterval>30</ReconnectionInterval>
    <MaxTransportPoolSize>250</MaxTransportPoolSize>
    <MaxIdleConnections>250</MaxIdleConnections>
    <EvictionTimePeriod>5500</EvictionTimePeriod>
    <MinIdleTimeInPool>5000</MinIdleTimeInPool>
    <SecureMaxTransportPoolSize>250</SecureMaxTransportPoolSize>
    <SecureMaxIdleConnections>250</SecureMaxIdleConnections>
    <SecureEvictionTimePeriod>5500</SecureEvictionTimePeriod>
    <SecureMinIdleTimeInPool>5000</SecureMinIdleTimeInPool>
  </Agent>
</DataAgentsConfiguration>
2. Instantiate the data publisher as follows: `AgentHolder.setConfigPath("/path/to/data/agent/conf.xml")`

3. Set trustStore

   ```java
   File securityFile = new File(<CEP_HOME> + "repository" + File.separator + "resources" + File.separator + "security");
   String trustStore = securityFile.getAbsolutePath();
   System.setProperty("javax.net.ssl.trustStore", trustStore + File.separator + "client-truststore.jks");
   System.setProperty("javax.net.ssl.trustStorePassword", "wso2carbon");
   ```

   The key also the can be found at [https://github.com/wso2/carbon-kernel/tree/master/distribution/kernel/carbon-home/repository/resources/security](https://github.com/wso2/carbon-kernel/tree/master/distribution/kernel/carbon-home/repository/resources/security)

4. Instantiate and use the data publisher using one of the following configurations:

   ```java
   DataPublisher dataPublisher = new DataPublisher(url, username, password);
   DataPublisher dataPublisher = new DataPublisher(receiverURLSet, username, password);
   DataPublisher dataPublisher = new DataPublisher(receiverURLSet, authURLSet, username, password);
   ```

   For information on the `receiverURLSet` and `authURLSet` parameters of the above configuration, see [Setting up Multi Receiver and Load Balancing Data Agent](https://wso2.com/docs/ecommerce-cloud/start-dev/setting-up-multi-receiver-and-load-balancing-data-agent/). And similarly if you are passing an `receiverURLSet` as tcp://localhost:7611|tcp://localhost:7612|tcp://localhost:7613, then the corresponding authentication URL set will be ssl://localhost:7711|ssl://localhost:7712|ssl://localhost:7713.

In all the above methods, the default data agent (which is configured as first Agent element in the above configuration) will be used to create the data publishers. If you have configured only the Thrift data agent in the `<CEP_HOME>/repository/conf/data-bridge/data-agent-conf.xml` file, then this will provide you a Thrift-based data publisher instance.

However, if you have configured more types of data agents in the `<CEP_HOME>/repository/conf/data-bridge/data-agent-conf.xml` file (Eg: Binary Agent in the above sample data-agent-conf.xml), then you can pass an additional property named `type`, which denotes the type of data publisher that needs to be created. For example, if you have a binary data publisher, then you can pass `binary` as the type to get the binary data publisher Instance as shown below.
Data publisher sample

As a prerequisite for this sample, you need to define the event streams and a WSO2Event receiver in the server (WSO2 DAS/CEP).

Follow the procedure below to use the data publisher.

1. Initialize the data publisher as follows.

   ```java
   AgentHolder.setConfigPath(getDataAgentConfigPath());
   DataPublisher dataPublisher = new DataPublisher(url, username, password);
   ```

2. Generate the stream ID for the stream from which you are going to publish the event as follows.

   ```java
   String streamId =
   DataBridgeCommonsUtils.generateStreamId(HTTPD_LOG_STREAM, VERSION);
   ```

3. Publish the events using any of the following methods.
   - In the following configuration, the published event is blocked being called until the event is put into a disruptor. If the disruptor is full it will wait until there is a free space.
     ```java
     Event event = new Event(streamId, System.currentTimeMillis(), new
     Object[]{"external"}, null,
     new Object[]{aLog});
     dataPublisher.publish(event);
     ```
   - Try publish as shown in the following configuration, is a non-blocking publishing. If there is a space available in the disruptor, it will try to insert the event. However, if the disruptor is full, the event is returned back immediately without waiting.
     ```java
     Event event = new Event(streamId, System.currentTimeMillis(), new
     Object[]{"external"}, null,
     new Object[]{aLog});
     dataPublisher.tryPublish(event);
     ```
   - Try publish as shown in the following configuration, is a non-blocking publishing with timeout in mili
seconds. If there is a space available in the disruptor it will try to insert the event, but if the disruptor is full it will wait for the specified amount of time, and if the timeout is reached the event is returned back.

```java
Event event = new Event(streamId, System.currentTimeMillis(), new Object[]{"external"}, null, new Object[]{aLog});
dataPublisher.tryPublish(event, 100);
```

For more information on the usage of data publishers, see the sample in the `<CEP_HOME>/samples/cep/producers/wso2-event/` directory.

### Setting up Multi Receiver and Load Balancing Data Agent

You can send events to multiple DAS/CEP receivers, either by sending the same event to many DAS/CEP receivers or by load balancing events among a set of servers. This handles the fail-over problem. When events are load balanced within a set of servers and if one receiver cannot be reached, events are automatically sent to the other available and active DAS/CEP receivers.

The following scenarios are covered in this section.

All the scenarios described below are different ways to use Data Agents with multiple receivers using the load balancing functionality. Each approach has its own advantages. Select the most appropriate scenario depending on your requirements.

- **Load balancing configurations**
  - Load balancing events to a set of servers
  - Load balancing events to sets of servers
- **Sending all the events to several receivers**
- **Failover configuration**

### Load balancing configurations

The load balancing configurations that can be used when sending events to multiple DAS/CEP receivers are as follows.

Load balancing events to a set of servers
This setup shows load balancing the event to publish it to all three DAS/CEP receivers. The load balanced publishing is done in a Round Robin manner, sending each event to each receiver in a circular order without any priority. It also handles fail-over cases such as, if DAS/CEP Receiver-1 is marked as down, then the Data Agent will send the data only to DAS/CEP Receiver-2 and DAS/CEP Receiver-3 in a round robin manner. When DAS/CEP Receiver-1 becomes active after some time, the Data Agent automatically detects it, adds it to the operation, and again starts to load balance between all three receivers. This functionality significantly reduces the loss of data and provides more concurrency.

For this functionality, include the server URL in the Data Agent as a general DAS/CEP receiver URL. The URL should be entered in a comma separated format as shown below.

```
```

In the above format, <DAS/CEP Receiver - 1, 2, 3> can be either host names or IP addresses, and <port> is the port of the corresponding DAS/CEP receiver.

e.g., tcp://10.100.2.32:7611, tcp://10.100.2.33:7611, tcp://10.100.2.34:7611

Load balancing events to sets of servers
In this setup there are two sets of servers that are referred to as set-A and set-B. You can send events to both the sets. You can also carry out load balancing for both sets as mentioned in load balancing between a set of servers. This scenario is a combination of load balancing between a set of servers and sending an event to several receivers. An event is sent to both set-A and set-B. Within set-A, it will be sent either to DAS/CEP ReceiverA-1 or DAS/CEP ReceiverA-2. Similarly within set-B, it will be sent either to DAS/CEP ReceiverB-1 or DAS/CEP ReceiverB-2. In the setup, you can have any number of sets and any number of servers as required by mentioning them accurately in the server URL.

Similar to the other scenarios, you can describe this as a receiver URL. The sets should be mentioned within curly braces separated by commas. Further more, each receiver that belongs to the set, should be within the curly braces and with the receiver URLs in a comma separated format. The receiver URL format is given below.

```
(tcp://<DAS/CEP Receiver -B-1>:<port>, tcp://<DAS/CEP Receiver -B-2>:<port>)
```

`<DAS/CEP Receiver- (A-1, 2)>` and `<DAS/CEP Receiver-B-(1, 2)>` can be host name or IP addresses, and `<port>` is the port of the corresponding DAS/CEP receiver.

e.g., (tcp://10.100.2.32:7611, tcp://10.100.2.33:7611), (tcp://10.100.2.34:7611, tcp://10.100.2.35:7611)
**Sending all the events to several receivers**

This setup involves sending all the events to more than one DAS/CEP receiver. This approach is mainly followed when you use other servers to analyze events together with DAS/CEP servers. For example, you can use the same Data Agents to publish the events to WSO2 DAS. You can use this functionality to publish the same event to both DAS and CEP servers at the same time. This will be useful to perform real time analytics with CEP and to persist the data, and also to perform complex analysis with DAS in nearly real time with the same data.

Similar to load balancing between a set of servers, in this scenario you need to modify the Data Agent URL. You should include all DAS/CEP receiver URLs within curly braces ({}), separated with commas as shown below.

```
```

\(<\text{DAS/CEP Receiver - 1, 2, 3}>\) can be either host name or IP addresses, and <port> is the port of the corresponding DAS/CEP receiver.

*e.g.,* \{tcp://10.100.2.32:7611\}, \{tcp://10.100.2.33:7611\}, \{tcp://10.100.2.34:7611\}

**Failover configuration**
When using the failover configuration in publishing events to DAS/CEP, events are sent to multiple DAS/CEP receivers in a sequential order based on priority. You can specify multiple DAS/CEP receivers so that events can be sent to the next server in the sequence in a situation where they were not successfully sent to the first server. In the scenario depicted in the above image, first events are sent to DAS/CEP Receiver-1. If it is unavailable, then events will be sent to DAS/CEP Receiver-2. Further, if that is also available, then events will be sent to DAS/CEP Receiver-3.

For this functionality, include the server URLs in the Data Agent, separated by the vertical bar (|) symbols as follows.

```
tcp://localhost:7611|tcp://localhost:7612
```

In the above format, <DAS/CEP Receiver - 1, 2, 3> can be either host names or IP addresses, and <port> is the port of the corresponding DAS/CEP receiver.

```
e.g., tcp://10.100.2.32:7611| tcp://10.100.2.33:7611| tcp://10.100.2.34:7611
```

### Publishing JSON/REST Data using Java Client

#### Introduction to HTTP REST publisher

HTTP publisher allows you to send data as a JSON object to a predefined set of data fields in a DAS/CEP server. The data structure with predefined fields is defined in an event stream. The data is converted to the format defined by the event stream and sent via the WSO2 data-bridge component. You can also send custom key-value pairs with data events.

#### Dependencies
In order to publish data to WSO2 DAS/CEP through a custom data agent, you need to have the following dependencies. You can configure the dependencies either using the class path or using the POM file.

**Adding dependencies using class path**

Add the JAR files listed below to your class path. Note that \${carbon.commons.version} refers to the version of the carbon-commons github repository - [https://github.com/wso2/carbon-commons/](https://github.com/wso2/carbon-commons/). It is always recommended to use the jar file from the latest released version.

- [httpclient-4.2.5.wso2v1.jar](httpclient-4.2.5.wso2v1.jar)
- [httpcore-4.3.3.wso2v1.jar](httpcore-4.3.3.wso2v1.jar)
- [gson-2.1.jar](gson-2.1.jar)

**Adding dependencies using POM file**

Alternatively, add the following Maven project dependency entries to your POM file.
Maven repository

<repositories>
  <repository>
    <id>wso2-nexus</id>
    <name>WSO2 internal Repository</name>
    <url>http://maven.wso2.org/nexus/content/groups/wso2-public/</url>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
      <checksumPolicy>ignore</checksumPolicy>
    </releases>
  </repository>
  <repository>
    <id>wso2.releases</id>
    <name>WSO2 internal Repository</name>
    <url>http://maven.wso2.org/nexus/content/repositories/releases/</url>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
      <checksumPolicy>ignore</checksumPolicy>
    </releases>
  </repository>
  <repository>
    <id>wso2.snapshots</id>
    <name>Apache Snapshot Repository</name>
    <url>http://maven.wso2.org/nexus/content/repositories/snapshots/</url>
    <snapshots>
      <enabled>true</enabled>
      <updatePolicy>daily</updatePolicy>
    </snapshots>
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>
HTTP REST publisher

As a prerequisite for this sample, you need to define the event streams and a WSO2Event receiver in the server (WSO2 DAS/CEP).

Follow the procedure below to use the data publisher.

1. Initialize the HTTP Client and HTTP POST method as follows.

   ```java
   HttpClient httpClient = new SystemDefaultHttpClient();
   HttpPost method = new HttpPost(url);
   ```

2. JSON event format should be as follows whereas event object is consist of metaData, correlationData and payloadData objects. Each object has set of properties and values.
2. 

{ 
  "event": { 
    "metaData": { 
      "timestamp": 1442921557056, 
      "isPowerSaverEnabled": false, 
      "sensorId": 100, 
      "sensorName": "temperature" 
    }, 
    "correlationData": { 
      "longitude": 2332.424, 
      "latitude": 2323.23232 
    }, 
    "payloadData": { 
      "humidity": 2.3, 
      "sensorValue": 23423.234 
    } 
  } 
}

3. You can use above event as a string or else you can even build JSON object by using a parser like gson and convert it as a string.

```java
JsonObject event = new JsonObject();
JsonObject metaData = new JsonObject();
JsonObject correlationData = new JsonObject();
JsonObject payLoadData = new JsonObject();

metaData.addProperty("timestamp", System.currentTimeMillis());
metaData.addProperty("isPowerSaverEnabled", false);
metaData.addProperty("sensorId", count);
metaData.addProperty("sensorName", "temperature");

correlationData.addProperty("longitude", 2332.424);
correlationData.addProperty("latitude", 2323.23232);

payLoadData.addProperty("humidity", 2.3f);
payLoadData.addProperty("sensorValue", 23423.234);

event.add("metaData", metaData);
event.add("correlationData", correlationData);
event.add("payloadData", payLoadData);

String eventString = "{"event": " + event + "}";
```

4. Once JSON String is created, it will be converted as a StringEntity by enclosing as event object. StringEntity is a class provided by apache HTTP library. This entity will be set to HTTP method.
StringEntity entity = new StringEntity(eventString);
method.setEntity(entity);

5. If URL endpoint is https, we need to add Authorization Basic header by encoding username and password. Finally we will use HTTP client to execute defined HTTP Method.

if (url.startsWith("https")) {
    method.setHeader("Authorization", "Basic " + Base64.encode((username + ":" + password).getBytes()));
} httpClient.execute(method).getEntity().getContent().close();

For more information on the usage of data publishers, see the sample producer in the `<CEP_HOME>/samples/cep/producing/ftp/` directory.

### Publishing Data Through Other Protocols

Data agents are used to collect large amounts of data about services, mediators etc. from various data collection points such as ESB, application servers, and custom data publishers, and pump them to data analysis and summarization servers such as WSO2 DAS and WSO2 Complex Event Processor.

The data publishing functionality is provided by the following feature in the WSO2 feature repository:

**Name**: WSO2 Carbon - Data Bridge - Data Publisher Aggregate Feature

**Identifier**: org.wso2.carbon.databridge.datapublisher.feature.group

If the above feature is not bundled in your product by default, you can install it using the instructions given in section Feature Management.

When using data publisher API to publish data in a periodic manner to WSO2 DAS/CEP, the eviction time and eviction idle time for the connections should be higher than the periodic interval. This is required to re-use the created socket connections from the pool, avoiding closure of it and creation of new connections. The default eviction period is 5.5 seconds (5500 milliseconds). If you are publishing events in a periodic interval as more than 5.5s, you need to tune the `<secureEvictionTimePeriod>` parameter accordingly, in the `<CEP_HOME>/repository/conf/data-bridge/thrift-agent-config.xml` file of the agent in the client side, by increasing this default value.

This section provides the following information:

- **Setting up the JMX Agent**

**Setting up the JMX Agent**

The JMX agent of WSO2 DAS monitors JMX attributes of a required JMX-enabled server (e.g. Carbon-based servers), and stores monitored data in WSO2 DAS. It uses the Thrift API of WSO2 DAS to send monitored data to DAS server. You can create a JMX monitoring profile to monitor a set of attributes from a single JMX server.
• Adding the default JMX profile
• Adding a JMX server profile
• Uploading the C-App
• Viewing the output

Adding the default JMX profile

You can set up the default JMX toolbox which is shipped with WSO2 DAS to monitor system resources of a WSO2 server (CPU/memory/OS) running on Linux. Follow the steps below to setup this default JMX toolbox.

1. Log in to the management console using admin/admin credentials and the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon/
2. Click Configure, and then click JMX Agent.
3. Click Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux (CPU/Memory/OS). This adds a pre-configured server profile to monitor the JMX attributes of WSO2 DAS itself as shown below.

   Home > Configure > JMX Agent

   JMX Monitoring Profiles

<table>
<thead>
<tr>
<th>Profile</th>
<th>Version</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolbox</td>
<td>1.0.0</td>
<td>![Enable], ![Edit], ![Delete]</td>
</tr>
</tbody>
</table>

   4. Click Enable in the Actions column, to enable the added server profile, and then click Edit.
   5. Change the pre-configured details of the server profile as required.

   You can change the Server URL, to monitor any WSO2 server accordingly. Change all occurrences of the host and port (if you have set a port offset on the server) accordingly in the JMX server URL.

   6. Click Add More to monitor more attributes by the server profile if required in the below screen.
7. Click an MBean on the list that loads to view its attributes list.
8. Select the attributes that you require to monitor by this profile as shown below. You can set an alias to easily identify the data in the Data Access Layer of WSO2 DAS.

9. Click **Save** to save the changes.
10. Click **No** in the below pop-up message, to add the changes to the existing version of the server profile.
Else, click **Yes** in the above pop-up message, to save the changes by incrementing the version of the server profile as shown below.

**Adding a JMX server profile**

Follow the steps below to set up a JMX server profile in WSO2 DAS to monitor JMX attributes.

1. Log in to the management console using admin/admin credentials and the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Configure**, and then click **JMX Agent**.
3. Click **Add JMX Server Profile**, to add a new monitoring profile.
4. Enter the required details as shown below.

**New Profile**

- **All the fields marked with * are mandatory**
- **Enter basic JMX profile information.**
  - **Name:**
  - **Schedule:** `once every 2 seconds (0/2 * * ? *)`
- **JMX Server Information**
  - **Server URL:** `service:jmx:rmi://localhost:11111/jndi/rmi`
  - **User Name:** `admin`
  - **Password:**`******`

The details you enter in the above screen are described below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>

**JMX Monitoring Profiles**

- **Add JMX Server Profile**
- **Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux (CPU/Memory/OS)**
Name | Unique name of the server profile. | testProfile
Schedule | CRON expression defining how often the attributes should be monitored. | once every 2 seconds (0/2**?**)
User Name | The username of the JMX server. | admin
Password | The password of the JMX server. | admin

5. Click **Load MBeans**. You see the loaded MBeans of the JMX server as shown below.

6. Click an MBean on the list that loads to view its attributes list.
7. Select the attributes that you require to monitor by this profile as shown below. You can set an alias to easily identify the data in the Data Access Layer of WSO2 DAS.
7. Click **Save**. You view the new server profile added to the list of existing profiles as shown below.

You can enable/disable monitoring of JMX attributes, and also edit or delete the monitoring profiles using the options provided in this screen.

### JMX Monitoring Profiles

**Add JMX Server Profile**

**Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux (CPU/Memory/OS)**

<table>
<thead>
<tr>
<th>Profile</th>
<th>Version</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>testProfile</td>
<td>1.0.0</td>
<td><img src="false" alt="Disable" /> <img src="false" alt="Edit" /> <img src="false" alt="Delete" /></td>
</tr>
</tbody>
</table>

After you edit the server profile, click **No** in the below pop-up message, to add the changes to the existing version of the server profile, or click **Yes** to save the changes by incrementing the version of the server profile as shown below.

![Do you want to increment the version?](false)

**Yes**  **No**

### JMX Monitoring Profiles

**Add JMX Server Profile**

**Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux (CPU/Memory/OS)**

<table>
<thead>
<tr>
<th>Profile</th>
<th>Version</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>testProfile</td>
<td>2.0.0</td>
<td><img src="false" alt="Enable" /> <img src="false" alt="Edit" /> <img src="false" alt="Delete" /></td>
</tr>
<tr>
<td>toolBox</td>
<td>1.0.0</td>
<td><img src="false" alt="Enable" /> <img src="false" alt="Edit" /> <img src="false" alt="Delete" /></td>
</tr>
</tbody>
</table>

**Uploading the C-App**

WSO2 DAS is shipped with a sample C-App for the JMX Agent. This includes all the artifacts which you need to publish data to the JMX agent which you enabled above, and to persist that data. Follow the steps below to upload this sample Carbon Application (c-App) file to the DAS. For more information, see [Carbon Application Deployment](false)
for DAS.

1. Log in to the DAS management console using the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon/.
2. Click Main, and then click Add in the Carbon Applications menu.
3. Click Choose File, and upload the <DAS_HOME>/samples/capps/JMX_Agent.car file as shown below.

   ![Add Carbon Applications](image)

4. Click Main, then click Carbon Applications, and then click List view, to see the uploaded Carbon application as shown below.

   ![Carbon Applications List](image)

**Viewing the output**

You may use the Data Explorer of the WSO2 DAS Management Console to browse published events.

Using the Data Explorer

Follow the steps below to use the Data Explorer to view the output.

1. Log in to the DAS management console if you are not already logged in.
2. Click Main, and then click Data Explorer in the Interactive Analytics menu.
3. Select JMX_AGENT_TOOLBOX for the Table Name as shown below.

   ![Data Explorer](image)

4. Click Search. You view the published data as shown below.
Configuring CEP to Receive Data

The following topics cover the different ways in which WSO2 CEP can be configured to receive data in the form of events.

- Configuring Event Receivers
- Input Mapping Types
- Building Custom Event Receivers

Configuring Event Receivers

Events are received by WSO2 CEP/DAS server using event receivers, which manage the event retrieval process. Event receiver configurations are stored in the file system as deployable artifacts. WSO2 CEP/DAS receives events via multiple transports in JSON, XML, Map, Text, and WSO2Event formats, and converts them into streams of canonical WSO2Events to be processed by the server.

- Event receiver types
- Event receiver configuration
- Creating event receivers
- Enabling statistics for event receivers
- Enabling tracing for event receivers
- Deleting event receivers
- Editing event receivers

Event receiver types

WSO2 CEP/DAS has the capability of receiving events from event receivers via various transport protocols. Following are the event receivers that come with WSO2 CEP/DAS by default. You can write extensions to support other transport.

- Email Event Receiver
- File-tail Event Receiver
- HTTP Event Receiver
- JMS Event Receiver
- Kafka Event Receiver
- MQTT Event Receiver
Event receiver configuration

An event receiver configuration has four main sections as shown in the example below.

Create a New Event Receiver

Event receiver configurations are stored in the file system as hot deployable artifacts in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as shown in the example below.

```
<eventReceiver name="WSO2EventEventReceiver" statistics="disable"
trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="wso2event">
    <property name="events.duplicated.in.cluster">false</property>
  </from>
  <mapping customMapping="disable" type="wso2event"/>
  <to streamName="testEventStream" version="1.0.0"/>
</eventReceiver>
```

The above sections of an event receiver configuration are described below.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>An input event adapter (transport) configuration via which the event receiver receives events.</td>
</tr>
<tr>
<td>Adapter properties</td>
<td>Specific properties of the selected input event adapter. For information on configuring adapter properties of various transport types, see Event Receiver Types.</td>
</tr>
<tr>
<td>To</td>
<td>The event stream from which the event receiver will fetch the events for processing.</td>
</tr>
<tr>
<td>Mapping configuration</td>
<td>The format of the message that is received. You can configure custom mappings on the selected format via advanced settings. For information on configuring custom mappings, see Input Mapping Types.</td>
</tr>
</tbody>
</table>

Creating event receivers

You can create event receivers either using the management console or using a configuration file as explained below.

Creating receivers using the management console

Follow the steps below to create an event receiver using the management console of WSO2 CEP/DAS.

1. Log in to the management console, and click Main.
2. Click Receivers in the Event menu, and then click Add Event Receiver.
3. Enter a name for Event Receiver Name. (Do not use spaces between the words in the name of the event receiver.)
4. Select the input transport from which you want to receive events for the Input Event Adapter Type, and enter the Adapter Properties accordingly. For instructions on the adapter properties of input transport types, see Event Receiver Types.
5. Select the Event Stream, to which you want to map the received events.
6. Select the Message Format which you want to apply on the receiving events. WSO2 servers allow users to configure events in XML, JSON, Text, Map, and WSO2Event event formats.
7. Click Advanced to define custom input mappings based on the message format you selected, if you are sending events that do not adhere to the default event formats. For more information on custom input mapping types, see Input Mapping Types.
8. Click Add Event Receiver, to create the event receiver in the system. When you click OK in the pop-up message on successful addition of the event receiver, you view it in the Available Event Receivers list as shown below.

Creating receivers using a configuration file

Follow the steps below to create an event receiver using a configuration file.

1. Create an XML file with the following event receiver configurations. An event receiver implementation must start with <eventReceiver> as the root element.
the receiver within the \texttt{<from>} element. For the respective adapter properties of the event receiver configuration based on the transport type, see \textit{Event Receiver Types}.

```xml
<eventReceiver name="EVENT-RECEIVER-NAME" statistics="disable" 
trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="EVENT-ADAPTER-TYPE">
    ..................................
  </from>

  <mapping customMapping="disable" type="xml"/>
  <to streamName="Test Stream" version="1.0.0"/>
</eventReceiver>
```

The properties of the above configuration are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the event receiver</td>
</tr>
<tr>
<td>statistics</td>
<td>Whether monitoring event statistics is enabled for the receiver</td>
</tr>
<tr>
<td>trace</td>
<td>Whether tracing events is enabled for the receiver</td>
</tr>
<tr>
<td>xmlns</td>
<td>XML namespace for event receivers</td>
</tr>
<tr>
<td>eventAdapterType</td>
<td>Type of the event adapter.</td>
</tr>
<tr>
<td>customMapping</td>
<td>Whether a custom mapping is enabled on the receiver.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the enabled custom mapping.</td>
</tr>
<tr>
<td>streamName</td>
<td>Name of the event stream to which the receiver is mapped.</td>
</tr>
</tbody>
</table>

2. Add the XML file to the \texttt{<PRODUCT_HOME>/repository/deployment/server/eventreceivers/} directory. Since hot deployment is supported in the product, you can simply add/remove event receiver configuration files to deploy/undeploy event receivers to/from the server.

First define the stream to which the receiver is publishing data to activate the receiver. When receiving WSO2Events, the incoming stream definition that you select in the advanced input mappings must also be defined, to activate the event receiver. When you click \textbf{Inactive Event Receivers} in the \textbf{Available Event Receivers} screen, if an event receiver is in the inactive state due to some issue in the configurations, you view a short message specifying the reason why the event receiver is inactive as shown below. A similar message is also printed on the CLI.
After a receiver is successfully added, it gets added to the list of receivers displayed under **Event** in the **Main** menu of the product's management console. Click **Edit** to change its configuration and redeploy it. This opens an XML-based editor allowing you to edit the event receiver configurations from the UI. Do your modifications and click **Update**. You can also delete it, enable/disable statistics or enable/disable tracing on it using the provided options in the UI as described below.

**Enabling statistics for event receivers**

Follow the steps below to enable monitoring statistics of events received by an existing event receiver.

1. Log in to the management console, and click **Main**.
2. Click **Receivers** in the **Event** menu. You view the **Available Event Receivers** list.
3. Click the **Enable Statistics** button of the corresponding event receiver to enable monitoring event statistics for it.

**Enabling tracing for event receivers**

Follow the steps below to enable tracing on events received by an existing event receiver.

1. Log in to the management console, and click **Main**.
2. Click **Receivers** in the **Event** menu. You view the **Available Event Receivers** list.
3. Click the **Enable Tracing** button of the corresponding event receiver to enable event tracing for it.

**Deleting event receivers**

Follow the steps below to delete an existing event receiver.

1. Log in to the management console, and click **Main**.
2. Click **Receivers** in the **Event** menu. You view the **Available Event Receivers** list.
3. Click the **Delete** button of the corresponding event receiver to delete it.

**Editing event receivers**

Follow the steps below to edit an existing event receiver.

1. Log in to the management console, and click **Main**.
2. Click **Receivers** in the **Event** menu. You view the **Available Event Receivers** list.
3. Click the **Edit** button of the corresponding event receiver to edit it. This opens **Edit Event Receiver Configurations** XML editor.
4. After editing, click **Update**, to save the configuration, or click **Reset** to reset the configuration to its original state.

**Email Event Receiver**
Email event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with XML, text, and JSON input mapping types.

- Prerequisites
- Creating an email event receiver
- Related samples

Prerequisites

Follow the steps below to complete the prerequisites before starting the input receiver configurations.

1. Remove any rich text formatting from the email body. It must contain only plain text.

Creating an email event receiver

For instructions on creating an email event receiver, see Configuring Event Receivers.

Configuring global properties

The following global property can be set for Email event receiver type in the <CEP_HOME>/repository/conf/input-event-adapters.xml file. This property applies to all the receivers of the Email type. If this property is removed from the file, its default value still applies.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveToFolderName</td>
<td>The name of the folder in which the events received should be saved. If a</td>
<td>String</td>
<td>readMails</td>
</tr>
<tr>
<td></td>
<td>folder with the given name does not already exist in the email server, it</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>will be automatically created when events are received by email-receivers.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Configuring adapter properties

Specify the Adapter Properties, when creating an email event receiver using the Management Console as shown below.
Create a New Event Receiver

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving Mail Address</td>
<td>A valid mail address from which this service should fetch incoming mails.</td>
<td>transport.mail.Address</td>
<td><a href="mailto:receiver@gmail.com">receiver@gmail.com</a></td>
</tr>
<tr>
<td>User Name</td>
<td>Username of the receiver email account.</td>
<td>mail.protocol.user</td>
<td>test-user</td>
</tr>
<tr>
<td>Password</td>
<td>Password of the receiver email account.</td>
<td>mail.protocol.password</td>
<td>test-password</td>
</tr>
<tr>
<td>Subject</td>
<td>Only the mails with this subject mentioned will be processed. (The mail should be in plain text format.)</td>
<td>email.in.subject</td>
<td>Email Input Event Adapter Test</td>
</tr>
<tr>
<td>Mail Protocol Host</td>
<td>Host address of the mail receiver.</td>
<td>mail.protocol.host</td>
<td>pop.gmail.com/imap.gmail.com</td>
</tr>
<tr>
<td>Mail Protocol Port</td>
<td>Port of the mail receiver.</td>
<td>mail.protocol.port</td>
<td>995/993</td>
</tr>
<tr>
<td>Mail Protocol</td>
<td>The mail protocol to be used to receive messages. The default value is imap. For imap, make sure it is enabled in your email account settings.</td>
<td>transport.mail.Protocol</td>
<td>pop3/imap</td>
</tr>
</tbody>
</table>
Poll Interval (in seconds) | A positive integer which denotes the time limit, in which the product needs to check for new mails. | transport.PollInterval | 10

**Related samples**

For more information on Email event receiver type, see the following sample.

- Sample 0015 - Receiving Text Events via Email Transport

**File-tail Event Receiver**

File-tail event receiver reads the tail of a given file and feeds that to the product engine. It only supports **text** input mapping type.

- Creating a file-tail event receiver
- Related samples

**Creating a file-tail event receiver**

For instructions on creating a file-tail event receiver, see Configuring Event Receivers.

**Configuring global properties**

The following global property can be set for the file-tail event receiver type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. This property applies to all the receivers of the file-tail type. If this property is removed from the file, its default value still applies.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>events.duplicated.in.cluster</td>
<td>If this property is set to true, events received by file-tail receivers are re-created in every node in the cluster.</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

**Configuring adapter properties**

Specify the **Adapter Properties**, when creating a file-tail event receiver using the management console as shown below.
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventReceiver name="FileTailInputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="file-tail">
        <property name="filepath">/User/Desktop/abc.txt</property>
        <property name="startFromEnd">true</property>
        <property name="delayInMillis">10</property>
    </from>
</eventReceiver>
```

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
</table>

After entering the above adapter properties, select the Event Stream to which you want to map the incoming events, and the Message Format which you want to apply on the receiving events. Also, click Advanced to define custom input mappings based on the Message Format you selected. For more information on custom input mapping types, see Input Mapping Types.
**File path**

Absolute path of the text file to read the data from.

| filepath | /Users/User/Desktop/abc.txt |

**Delay**

The delay between checks for new content on file in milliseconds.

| delayInMillis | 10 |

**Start From End**

Set to true to tail from the end of the file, false to tail from the beginning of the file.

| startFromEnd | true |

**Related samples**

For more information on file-tail event receiver type, see the following samples.

- Sample 0017 - Receiving Custom Text Events via File Tail Transport
- Sample 0022 - Receiving Custom RegEx Text Events via File Tail
- Sample 0117 - Filtering and Outputting to Multiple Streams

**HTTP Event Receiver**

HTTP event receiver is an internal event receiver that comes with WSO2 products which is used to receive events in XML, JSON or Text formats via HTTP, HTTPS, and local transports.

- Creating a HTTP event receiver
- Related samples

**Creating a HTTP event receiver**

For instructions on creating a HTTP event receiver, see Configuring Event Receivers.

Configuring global properties

The following global properties can be set for the HTTP event receiver type in the `<CEP_HOME>/repository/con f/input-event-adapters.xml` file. These properties apply to all the receivers of the HTTP type. If a global property available by default is removed, the default value of the property is considered.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minOccurs</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td>keepAliveTimeInMillis</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td>jobQueueSize</td>
<td>The size of the queue that is used to hold events before they are forwarded to the event stream.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

Configuring adapter properties

Specify the **Adapter Properties**, when creating a HTTP event receiver using the management console as shown...
Create a New Event Receiver

Enter Event Receiver Details

**Event Receiver Name**: httpReceiver

- Enter a unique name to identify Event Receiver

**From**

- http

**Input Event Adapter Type**

- Select the type of Adapter to receive events

Following url formats are used to receive events
For super tenants:
- http://localhost:9764/endpoints/<event_receiver_name>
- https://localhost:9444/endpoints/<event_receiver_name>

For other tenants:
- http://localhost:9764/endpoints/t/<tenant_domain>/event_receiver_name>
- https://localhost:9444/endpoints/t/<tenant_domain>/event_receiver_name>

Usage Tips

**Adapter Properties**

**Transport(s)**

- http

**To**

**Event Stream**

- TestStream:1.0.0

- The event stream that will be generated by the received events

**Mapping Configuration**

**Message Format**

- xml

- Select the input message format

+ Advanced

Add Event Receiver

After entering the transport type in adapter properties, select the Event Stream to which you want to map the incoming events, and the Message Format which you want to apply on the receiving events. Also, click Advanced to define custom input mappings based on the Message Format you selected. For more information on custom input mapping types, see Input Mapping Types.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport(s)</td>
<td>Transport type via which the events are received.</td>
<td>transports</td>
<td>https</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on http event receiver type, see the following samples.

- Sample 0001 - Receiving JSON Events via HTTP Transport
- Sample 0002 - Receiving Custom JSON Events via HTTP Transport
- Sample 0003 - Receiving XML Events via HTTP Transport
- Sample 0004 - Receiving Custom XML Events via HTTP Transport
- Sample 0005 - Receiving Text Events via HTTP Transport
- Sample 0006 - Receiving Custom Text Events via HTTP Transport
- Sample 0111 - Detecting non-occurrences with Patterns
- Sample 0113 - Limiting the Output Rate of an Event Stream
- Sample 0115 - Quartz scheduler based alerts
- Sample 0502 - Processing a Window State Over Server Restart
- Sample 0503 - Processing a Window Query in High Availability Mode

**JMS Event Receiver**

JMS event receiver are used to receive events in XML, JSON, map, and text formats via a JMS transport. You can configure any type of JMS event receiver to run with WSO2 CEP/DAS.

The following global properties can be set for the jms receiver type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the receivers of the jms type. If a property available by default is removed from the file, the default value of that property is still considered.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
</table>

The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.

<table>
<thead>
<tr>
<th>keepAliveTimeInMillis</th>
<th>Integer 20000</th>
</tr>
</thead>
</table>

The size of the queue that is used to hold events before they are forwarded to the event stream.

<table>
<thead>
<tr>
<th>jobQueueSize</th>
<th>Integer 10000</th>
</tr>
</thead>
</table>

The following sections describe how to configure a few common JMS event receiver types.

- **ActiveMQ Event Receiver**
- **IBM WebSphere MQ JMS Event Receiver**
- **Oップ JMS Event Receiver**
- **TIBCO JMS Event Receiver**
- **WSO2 Message Broker JMS Event Receiver**

### ActiveMQ Event Receiver

ActiveMQ JMS event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with **XML**, **map**, **JSON**, and **text** input mapping types.

- Prerequisites
- Creating an ActiveMQ JMS event receiver
- Related samples

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install Apache ActiveMQ JMS.

   This guide uses ActiveMQ versions 5.7.0 or below. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to [Apache ActiveMQ Documentation](https://activemq.apache.org).

2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.

3. Start the ActiveMQ JMS server.

**Creating an ActiveMQ JMS event receiver**

For instructions on creating an ActiveMQ JMS event receiver, see [Configuring Event Receivers](https://wso2.com).

Configuring adapter properties

Specify the **Adapter Properties**, when creating an ActiveMQ JMS event receiver using the management console as shown below.
Create a New Event Receiver

<table>
<thead>
<tr>
<th>Enter Event Receiver Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event Receiver Name</strong></td>
</tr>
<tr>
<td><a href="#">ActiveMQJMSInputEvent Adapter</a></td>
</tr>
<tr>
<td><strong>From</strong></td>
</tr>
<tr>
<td>JMS</td>
</tr>
<tr>
<td>Select the type of Adapter to receive events</td>
</tr>
<tr>
<td><strong>Input Event Adapter Type</strong></td>
</tr>
</tbody>
</table>

**Adapter Properties**

<table>
<thead>
<tr>
<th>Topic/Queue Name*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Topic</td>
</tr>
<tr>
<td>Topic/Queue name of the input stream</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JNDI Initial Context Factory Class*</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.activemq.jndi.ActiveMQInitialContextFactory</td>
</tr>
<tr>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JNDI Provider URL*</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp://localhost:61616</td>
</tr>
<tr>
<td>URL of the JNDI provider.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The JMS connection password</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>The JMS connection username</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Connection Factory JNDI Name*</th>
</tr>
</thead>
<tbody>
<tr>
<td>The JNDI name of the connection factory.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination Type*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
</tr>
<tr>
<td>Type of the destination</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enable Durable Subscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
</tr>
<tr>
<td>Whether the subscription is durable or not.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Durable Subscriber Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the durable subscriber (if any value added, Durable subscription will be enabled).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JMS Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis2 JMS properties, e.g. “property1: value1, property2: value2”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream*</td>
</tr>
<tr>
<td>Test Stream: 1.0.0</td>
</tr>
<tr>
<td>The event stream that will be generated by the received events</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Format*</td>
</tr>
<tr>
<td>xml</td>
</tr>
<tr>
<td>Select the input message format</td>
</tr>
</tbody>
</table>

Advanced

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.
<eventReceiver name="ActiveMQJMSInputEventAdapter" statistics="disable"
trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="jms">
    <property name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</property>
    <property name="transport.jms.UserName">jms-user</property>
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.SubscriptionDurable">false</property>
    <property name="transport.jms.DurableSubscriberName">subscriber</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
  </from>
</eventReceiver>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic/Queue Name</td>
<td>A string of characters to denote a valid name of a JMS topic to subscribe to, or named queue to use when WSO2 CEP/DAS sends and receives messages.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td>The JMS connection password</td>
<td>A valid password for the JMS connection.</td>
<td>transport.jms.Password</td>
</tr>
<tr>
<td>The JMS connection username</td>
<td>A valid username for the JMS connection.</td>
<td>transport.jms.UserName</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Connection Factory JNDI Name</td>
<td>The JNDI name of the connection factory.</td>
<td>transport.jms.ConnectionFactoryJNDI</td>
</tr>
<tr>
<td>Destination Type</td>
<td>The sort order for messages that arrive on a specific destination.</td>
<td>transport.jms.DestinationType</td>
</tr>
<tr>
<td>Enable Durable Subscription</td>
<td>Whether the subscription is durable or not.</td>
<td>transport.jms.SubscriptionDurable</td>
</tr>
<tr>
<td>Durable Subscriber Name</td>
<td>A string of characters to denote a valid name of the durable subscriber. (It enables durable subscription if you add any value here).</td>
<td>transport.jms.DurableSubscriberName</td>
</tr>
<tr>
<td>JMS Properties</td>
<td>Valid property and value pairs to denote Axis2 JMS properties (e.g. &quot;property1: value1, property2: value2&quot;) For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation.</td>
<td>jms.properties</td>
</tr>
<tr>
<td>JMS Secured Properties</td>
<td>Properties that needs to be encrypted</td>
<td>jms.secured.properties</td>
</tr>
</tbody>
</table>

Related samples

For more information on ActiveMQ event receiver type, see the following samples.

- Sample 0009 - Receiving Map Events via JMS Transport - ActiveMQ
- Sample 0010 - Receiving Custom Map Events via JMS Transport - ActiveMQ
- Sample 0011 - Receiving JSON, Text, XML Events via JMS Transport - ActiveMQ
- Sample 0021 - Receiving Map Events via JMS Transport - ActiveMQ (For Queue)

**IBM WebSphere MQ JMS Event Receiver**

IBM WebSphere MQ JMS event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with XML, map, JSON, and text input mapping types.

- Prerequisites
- Creating an IBM WebSphere MQ JMS event receiver

Prerequisites

Follow the instructions below to complete the prerequisites before starting the event receiver configurations.

Configuring WebSphere MQ
Follow the steps below to configure WebSphere MQ.

### Configuring JMSAdmin.conf file

1. Download and install WebSphere MQ pack with the latest fixes. For more information on installing, go to the IBM documentation.
2. Start the IBM WebSphere MQ JMS server.
3. Open the `<WebSphere_MQ_HOME>/java/bin/JMSAdmin.config` file in a text editor.
4. Comment the existing INITIAL_CONTEXT_FACTORY, and add an INITIAL_CONTEXT_FACTORY named `com.sun.jndi.fscontext.RefFSContextFactory` as follows:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   ```

5. Comment the default PROVIDER_URL, and use a directory path instead. Ensure the directory is created in the file system (e.g., `C:/JNDI-Directory`).

   ```
   If there are .bindings files of earlier versions already existing in this directory, delete them. It should typically be an empty folder.
   ```

Your JMSAdmin.config file should now look similar to this:

```
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WMQInitialContextFactory
#
# The following line specifies the URL of the service provider's initial context. It currently refers to an LDAP root context. Examples of a file system URL and WebSphere's JNDI namespace are also shown, commented out.
#
#PROVIDER_URL=ldap://polaris/o=ibm,c=us
PROVIDER_URL=file:/C:/JNDI-Directory
#PROVIDER_URL=iiop://localhost/
#PROVIDER_URL=localhost:1414/SYSTEM.DEF.SVRCONN
......
```

6. Restart the WebSphere MQ service.

### Creating the Queue in WebSphere MQ

1. Start IBM WebSphere MQ Explorer, and create a new queue manager. Make sure you select **make this the default queue manager** check box and leave the default values on the other fields as shown below.
2. Select the options to **Start Queue Manager**, **Autostart Queue Manager**, and **Create server connection channel**, and then click **Next**.

3. Select the option to create a listener configuration for TCP/IP, and provide a port number (e.g., 1415).

4. Select the created queue manager and expand its navigation tree. Click **Queues** in the tree and create a new local queue (e.g., localq) as shown below.

5. Keep the default configurations, and click **Finish**.

6. Click **Topics** in the tree view, and create a new local topic (e.g., localt).

7. Right-click **Channels** under **Advanced**, and click **New Server-connection Channel**. Provide a name for the channel (e.g., myChannel), and click **Next**.
8. Set the transmission protocol as TCP, and click Finish. A listener is created and is running on the given port (e.g., 1415). You should be able to view it by clicking the listeners icon.

Generating the .bindings file

1. Navigate to the `<WebSphere_MQ_HOME>/java/bin/` directory, and invoke the IVT app by running the following command:

   ```
   IVTRun.bat -nojndi -client -m mymgr -host localhost -channel mychannel
   ```

2. Create the default set of JNDI bindings by running the following command on the command prompt:

   ```
   IVTSetup.bat
   ```

3. Execute the IVTRun tool as follows.

   ```
   ```

You have now enabled and verified JNDI support.

4. Navigate to C:/JNDI-Directory to view the .bindings file there.
5. Start the JMSAdmin tool by running the `jmsadmin.bat` file.
6. Modify the JNDI bindings by executing the following commands:

   For queues:

   ```
   ALTER QCF(ivtQCF) TRANSPORT(CLIENT)
   ALTER QCF(ivtQCF) QMGR(mymgr)
   ```

   For topics:
7. In IBM WebSphere MQ Explorer, select **JMS Administered Objects** from the tree view on the left, and then select **Add initial context**.

8. Select **File system**, and enter the JNDI directory path. This brings up all created queues and topics. You have now set up and configured IBM WebSphere MQ in your environment.

**Configuring WSO2 CEP**

Follow the instructions below to configure WSO2 CEP.

1. If you set up WSO2 CEP on a different machine from WebSphere MQ, copy `C:/JNDI-Directory/` to that machine. The bindings file allows you to access WebSphere queues from any machine in the network.

2. Copy the following JAR files from the `<WebSphere_MQ_HOME>/java/lib/` directory to the `<CEP_HOME>/repository/components/lib/` directory.
   - `com.ibm.mq.jms.jar`
   - `fscontext.jar`
   - `providerutil.jar`
   - `com.ibm.mq.jmqi.jar`
   - `dhubcore.jar`

3. If you are using WebSphere MQ version 6.0 instead of version 7.0, add the following two JAR files. You might not find `com.ibm.mq.jmqi.jar` in version 6.0.
   - `com.ibm.mq.jar`
   - `connector.jar`

   Optionally, you might have to add the following JAR files as well.
   - `jms.jar`
   - `jndi.jar`
   - `jta.jar`
   - `ldap.jar`

4. If you are using WebSphere MQ version 7.1 or later, add the following JAR files to the `<CEP_HOME>/repository/components/dropins/` directory.
   - `com.ibm.mq_2.0.0.jar`
   - `fscontext_1.0.0.jar`

5. Add the following files to the `<CEP_HOME>/repository/components/lib` directory.
   - `jms.jar`
   - `jta.jar`

6. Log in to the JMSAdmin tool, and create a queue named **bogusq** by running the following commands in JMSAdmin shell.

   ```
   DEFINE Q(bogusq) QMGR(mymgr)
   ALTER Q(bogusq) QUEUE(localq)
   ```

   `localq` is the queue you created earlier. You use two queues for the queue scenario. The queue named **bogusq** is the default destination since you need the default queue (ivtQ) for your proxy service only. If you use **ivtQ** here, all the services deployed in WSO2 CEP (XKMS, echo, wso2carbon-sts etc.) will start listening on the same queue.
7. Repeat these steps for the topic scenarios. For example:

```
DEFINE T(bogust)
ALTER T(bogust) TOPIC(localt)
```

`localt` is the topic you created earlier.

8. Configure the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file as follows:

```
<transportReceiver name="jms"
    class="org.apache.axis2.transport.jms.JMSListener">
    <parameter name="myTopicConnectionFactory" locked="false">
        <parameter name="java.naming.factory.initial" locked="false">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
        <parameter name="java.naming.provider.url" locked="false">file:/C:/JNDI-Directory</parameter>
        <parameter name="transport.jms.ConnectionFactoryJNDIName" locked="false">ivtQCF</parameter>
        <parameter name="transport.jms.ConnectionFactoryType" locked="false">topic</parameter>
    </parameter>

    <!--parameter name="SQProxyCF" locked="false">
        <parameter name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
        <parameter name="java.naming.provider.url">file:/C:/JNDI-Directory</parameter>
        <parameter name="transport.jms.ConnectionFactoryJNDIName">ivtQCF</parameter>
        <parameter name="transport.jms.ConnectionFactoryType">queue</parameter>
    </parameter-->

    <parameter name="default" locked="false">
        <parameter name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
        <parameter name="java.naming.provider.url">file:/C:/JNDI-Directory</parameter>
        <parameter name="transport.jms.ConnectionFactoryJNDIName">ivtQCF</parameter>
        <parameter name="transport.jms.ConnectionFactoryType">queue</parameter>
    </parameter>
</transportReceiver>
```

Comment and uncomment the non-default connection factories depending on which scenario you are
running, as described in the next section.

For details on the JMS configuration parameters used in the code segments, see JMS Connection Factory Parameters.

Creating an IBM WebSphere MQ JMS event receiver

For instructions on creating an IBM WebSphere JMS event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the Adapter Properties, when creating an IBM WebSphere JMS event receiver using the management console as shown below.
Create a New Event Receiver

### Enter Event Receiver Details

- **Event Receiver Name**: Enter a unique name to identify Event Receiver
- **From**: jms
- **Input Event Adapter Type**: Select the type of Adapter to receive events

### Adapter Properties

- **Topic/Queue Name**: test_topic
- **JNDI Initial Context Factory Class**: com.sun.jndi.fscontext.RefFSContextFactory
- **JNDI Provider URL**: file:/C:/JNDI-Directory
- **The JMS connection password**: 
- **The JMS connection username**: 
- **Connection Factory JNDI Name**: intJCF
- **Destination Type**: topic
- **Enable Durable Subscription**: false
- **Durable Subscriber Name**: 
- **JMS Properties**: Axis2 JMS Properties, e.g. "property1: value1, property2: value2"

### To

- **Event Stream**: Test Stream:1.0.0
- **Mapping Configuration**: wml
- **Message Format**: Select the input message format

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.
<eventReceiver name="iBMWebSphereJMSInputEventAdapter"
    statistics="disable" trace="disable"
    xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="jms">
        <property
            name="java.naming.factory.initial">com.sun.jndi.fscontext.ReffFSContextFactory</property>
        <property
            name="java.naming.provider.url">file:/C:/JNDI-Directory</property>
        <property
            name="transport.jms.DestinationType">topic</property>
        <property
            name="transport.jms.SubscriptionDurable">false</property>
        <property
            name="transport.jms.Destination">test_topic</property>
        <property
            name="transport.jms.DurableSubscriberName">subscriber</property>
        <property
            name="transport.jms.UserName">jms-user</property>
        <property
            name="transport.jms.Password">P2ve4G8+qF7JXkiGnP9/Ew5GXALEWTu7znEwxZGYa/MQMaQBRfsXiP094fn9U+0rntdBMitXU9o7h5uV3m5h97Po8WTJRpnnF8BV5YCGZEO+ELSg6twY3386MipwFhFMrbUKamI2sXksDrCogojWKtoHNmODnt8Ud1dhbOLK5zagc=</property>
        <property
            name="transport.jms.DestinationFactoryJNDIName">ivtQCF</property>
        <property
            name="transport.jms.ConnectionFactoryJNDIName">ivtQCF</property>
    </from>
</eventReceiver>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topic/Queue Name</strong></td>
<td>A valid name for the JMS topic/queue. WSO2 CEP/DAS sends and receives messages by subscribing to a topic or using named queues.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td><strong>JNDI Initial Context Factory Class</strong></td>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td><strong>JNDI Provider URL</strong></td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td><strong>The JMS connection password</strong></td>
<td>A valid password for the JMS connection</td>
<td>transport.jms.Password</td>
</tr>
</tbody>
</table>
**Qpid JMS Event Receiver**

Qpid JMS event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with `map`, `XML`, `JSON`, and `text` input mapping types.

- Prerequisites
- Creating a Qpid JMS event receiver
- Related samples

Prerequisites

Follow the steps below to set up the prerequisites before starting the event receiver configurations.

1. Install JMS-Qpid Broker and JMS-Qpid Client.
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory by entering relevant details in the register some connection factories section. Below is a sample for a connection factory named `TopicConnectionFactory` (the "default" is the name of the virtually hosted node in Qpid).

   ```
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/default?brokerlist='tcp://localhost:5672'
   ```

4. Start the Qpid JMS server.
Creating a Qpid JMS event receiver

For instructions on creating a Qpid JMS event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the **Adapter Properties**, when creating a Qpid JMS event receiver using the management console as shown below.

### Create a New Event Receiver

<table>
<thead>
<tr>
<th>Enter Event Receiver Details</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Receiver Name*</td>
<td>QpidMSInputEventAdapter</td>
</tr>
<tr>
<td>From</td>
<td>jms</td>
</tr>
<tr>
<td>Input Event Adapter Type*</td>
<td>Select the type of Adapter to receive events</td>
</tr>
</tbody>
</table>

**Adapter Properties**

<table>
<thead>
<tr>
<th><strong>Topic/Queue Name</strong></th>
<th>test-topic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JNDI Initial Context Factory Class</strong></td>
<td>org.apache.qpid.jms.PropertiesFileInitialContextFactory</td>
</tr>
<tr>
<td><strong>JNDI Initial Class Factory Class</strong></td>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface</td>
</tr>
<tr>
<td><strong>JNDI Provider URL</strong></td>
<td>repository/infra/jndi.properties</td>
</tr>
<tr>
<td><strong>The JMS connection password</strong></td>
<td></td>
</tr>
<tr>
<td><strong>The JMS connection username</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Connection Factory JNDI Name</strong></td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td><strong>Destination Type</strong></td>
<td>topic</td>
</tr>
<tr>
<td><strong>Enable Durable Subscription</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Durable Subscriber Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>JMS Properties</strong></td>
<td>Axis2 JMS Properties, e.g. &quot;property1: value1, property2: value2&quot;</td>
</tr>
</tbody>
</table>

**To**

| **Event Stream** | Test Stream:1.0.0 |
| **Mapping Configuration** |  |
| **Message Format** | xml |
| **Advanced** | Select the input message format |

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventReceiver name="QpidJMSInputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="jms">
    <property name="java.naming.factory.initial">org.apache.qpid.jndi.PropertiesFileInitialContextFactory</property>
    <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.SubscriptionDurable">false</property>
    <property name="transport.jms.Destination">test-topic</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    <property encrypted="true" name="transport.jms.Password">P2ve4G8+qF7JXkiGnP9/Ew5GXALEWTu7znEwx2GYa/MQMaQBRfsXiP094fn9U+0rntdBMitUX907h5uV3m5h97Po8WTJRpnFBV5YCGZEO+ELs96twY3386MipwFhMRbUMKamI2sXksDrqgojWKtoH7nObt8Ud1dh0LK59zgec=</property>
    <property name="transport.jms.UserName">jms-user</property>
    <property name="transport.jms.DurableSubscriberName">subscriber</property>
    <property name="jms.properties">SessionTransacted:false</property>
  </from>
</eventReceiver>
```

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic/Queue Name</td>
<td>Valid name for the JMS topic. WSO2 CEP/DAS sends and receives messages by subscribing to a topic or using named queues.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
</tbody>
</table>

more information on custom input mapping types, see [Input Mapping Types](#).

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic/Queue Name</td>
<td>Valid name for the JMS topic. WSO2 CEP/DAS sends and receives messages by subscribing to a topic or using named queues.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td><strong>The JMS connection password</strong></td>
<td>A valid password for the JMS connection</td>
<td><code>transport.jms.Password</code></td>
</tr>
<tr>
<td><strong>The JMS connection username</strong></td>
<td>A valid username for the JMS connection</td>
<td><code>transport.jms.UserName</code></td>
</tr>
<tr>
<td><strong>Connection Factory JNDI Name</strong></td>
<td>The JNDI name of the connection factory.</td>
<td><code>transport.jms.ConnectionFactoryJNDI</code></td>
</tr>
<tr>
<td><strong>Destination Type</strong></td>
<td>Define the sort order for messages that arrive on a specific destination.</td>
<td><code>transport.jms.DestinationType</code></td>
</tr>
<tr>
<td><strong>Enable Durable Subscription</strong></td>
<td>Whether the subscription is durable or not.</td>
<td><code>transport.jms.SubscriptionDurable</code></td>
</tr>
<tr>
<td><strong>Durable Subscriber Name</strong></td>
<td>A valid name of a durable subscriber (Durable subscription gets enabled if you add a value here.)</td>
<td><code>transport.jms.DurableSubscriberName</code></td>
</tr>
<tr>
<td><strong>JMS Properties</strong></td>
<td>A valid &quot;property:value&quot; pair of Axis2 JMS properties. (E.g. &quot;property1: value1, property2: value2&quot;) For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation.</td>
<td><code>jms.properties</code></td>
</tr>
</tbody>
</table>

### Related samples

For more information on Qpid event receiver type, see the following sample.

- [Sample 0012 - Receiving Map, XML Events via JMS Transport - Qpid](#)

### TIBCO JMS Event Receiver

TIBCO JMS event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with XML, JSON, and text input mapping types.

- **Prerequisites**
  - Creating a TIBCO JMS event receiver

#### Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Download and install TIBCO Enterprise Message Service. For more information on installing, go to TIBCO documentation.
2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.
3. Start the TIBCO EMS Server.

#### Creating a TIBCO JMS event receiver

For instructions on creating an TIBCO JMS event receiver, see Configuring Event Receivers
Configuring adapter properties

Specify the **Adapter Properties**, when creating a TIBCO JMS event receiver using the management console as shown below.

**Event Receiver Details**

![Event Receiver Details](image)

When configuring WSO2 CEP for fail over connections, specify JNDI Provider URLs as a comma-separated list of URLs. For example:

```
tibjmsnaming://localhost:7222, tibjmsnaming://localhost:7224
```

WSO2 CEP will attempt to connect to each URL in the ordered list. If a connection to one URL fails, the WSO2 CEP will try the next URL in the list.

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventReceiver name="TibcoJmsReceiver" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="jms">
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.DurableSubscriberName">subscriber</property>
    <property name="transport.jms.Destination">Test Topic</property>
    <property>
      name="java.naming.factory.initial">com.tibco.tibjms.naming.TibjmsInitialContextFactory</property>
    <property name="java.naming.provider.url">tibjmsnaming://localhost:7222</property>
    <property name="transport.jms.SubscriptionDurable">true</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    <property encrypted="true" name="transport.jms.Password">JP4yDiEh6HogOEjZzQQwHaJFIWZlnJTzaERl4eYrwukNeypm36R+odMkaN9b2q4H9jBQsR+mhCTlwQVnBpEzn4a+SuFuLKh3NhDEgwwe6Rl7ztZVo8p1D6TUKvSHXyEpwS0gKrk0mdaFEOQjfdhfk3HrnjKz/MYPYQknrL5MIY=</property>
    <property name="transport.jms.UserName">jms-user</property>
  </from>
  ............
</eventReceiver>
```

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic/Queue Name</td>
<td>A string of characters to denote a valid name of a JMS topic to subscribe to, or named queue to use when WSO2 CEP/DAS sends and receives messages.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td>The JMS connection password</td>
<td>A valid password for the JMS connection.</td>
<td>transport.jms.Password</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>The JMS connection username</td>
<td>A valid username for the JMS connection.</td>
<td>transport.jms.UserName</td>
</tr>
<tr>
<td>Connection Factory JNDI Name</td>
<td>The JNDI name of the connection factory.</td>
<td>transport.jms.ConnectionFactoryJNDI</td>
</tr>
<tr>
<td>Destination Type</td>
<td>The sort order for messages that arrive on a specific destination.</td>
<td>transport.jms.DestinationType</td>
</tr>
<tr>
<td>Enable Durable Subscription</td>
<td>Whether the subscription is durable or not.</td>
<td>transport.jms.SubscriptionDurable</td>
</tr>
<tr>
<td>Durable Subscriber Name</td>
<td>A string of characters to denote a valid name of the durable subscriber. (It enables durable subscription if you add any value here).</td>
<td>transport.jms.DurableSubscriberName</td>
</tr>
<tr>
<td>JMS Properties</td>
<td>Valid property and value pairs to denote Axis2 JMS properties (e.g. &quot;property1: value1, property2: value2&quot;)</td>
<td>jms.properties</td>
</tr>
</tbody>
</table>

For more information on Axis2 JMS properties, go to [Apache AXIS2 Transports Documentation](https://axis.apache.org/axis2/docs/).  

---

**WSO2 Message Broker JMS Event Receiver**

WSO2 Message Broker (MB) JMS event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with **map**, **XML**, **JSON**, and **text** input mapping types.

- **Prerequisites**
- Creating a WSO2 MB JMS event receiver
- Related samples

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configurations.

1. Download and install WSO2 Message Broker (MB). For instructions on WSO2 MB, go to [Message Broker documentation](https://wso2.com/docs/).

2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.

3. Open the `<CEP_HOME>/repository/conf/jndi.properties` file and register a connection factory by...
entering relevant details in the register some connection factories section. Below is a sample for a connection factory named TopicConnectionFactory (the "carbon" is the name of the virtually hosted node in WSO2 MB).

```
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/carbon?brokerlist='tcp://localhost:5672'
```

### Creating a WSO2 MB JMS event receiver

For instructions on creating a WSO2 MB JMS event receiver, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the **Adapter Properties**, when creating a WSO2 MB JMS event receiver using the management console as shown below.
Create a New Event Receiver

### Enter Event Receiver Details

<table>
<thead>
<tr>
<th>Event Receiver Name*</th>
<th>WSO2NBJMSInputEventAdapter</th>
</tr>
</thead>
</table>

#### From

<table>
<thead>
<tr>
<th>Input Event Adapter Type*</th>
<th>Select the type of Adapter to receive events</th>
</tr>
</thead>
</table>

#### Adapter Properties

<table>
<thead>
<tr>
<th>Topic/Queue Name*</th>
<th>&quot;test topic&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JNDI Initial Context Factory Class</strong></td>
<td>org.wso2.carbon.jndi.PropertiesFileInitialContextFactory</td>
</tr>
<tr>
<td><strong>JNDI Provider URL</strong></td>
<td>repository/conf/jndi.properties</td>
</tr>
<tr>
<td>The JMS connection password</td>
<td></td>
</tr>
<tr>
<td>The JMS connection username</td>
<td></td>
</tr>
<tr>
<td>Connection Factory JNDI Name*</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td><strong>Destination Type</strong></td>
<td>&quot;topic&quot;</td>
</tr>
<tr>
<td>Enable Durable Subscription</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>Durable Subscriber Name</td>
<td>Name of the durable subscriber (if any value added, durable subscription will be enabled)</td>
</tr>
<tr>
<td><strong>JMS Properties</strong></td>
<td>Axis2 JMS Properties, e.g. &quot;property1: value1, property2: value2&quot;</td>
</tr>
</tbody>
</table>

#### To

<table>
<thead>
<tr>
<th>Event Stream*</th>
<th>Test Stream:1.0.0</th>
</tr>
</thead>
</table>

### Mapping Configuration

<table>
<thead>
<tr>
<th>Message Format*</th>
<th>Select the input message format</th>
</tr>
</thead>
</table>

---

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.
<eventReceiver name="WSO2MBJMSInputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="jms">
    <property name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
    <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.SubscriptionDurable">false</property>
    <property name="transport.jms.Destination">test_topic</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    <property encrypted="true" name="transport.jms.Password">P2ve4G8+qF7JXkiGnP9/Ew5GXALEWTu7znEwxBGYa/MQMaQBRfsXlP094fn9U+0rntdBMItXU9o7h5uV3m5h97Po8WTJRpnFBV5YCGZEO+ELs96twY3386MipwFHMrbUMXam12sXxkDRcogojWktoHNM0dnt8Ud1dh0LK5zqc=</property>
    <property name="transport.jms.UserName">jms-user</property>
    <property name="transport.jms.DurableSubscriberName">subscriber</property>
    <property name="jms.properties">SessionTransacted:false</property>
  </from>
  ....................
</eventReceiver>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic/Queue Name</td>
<td>Valid name for the JMS topic. WSO2 CEP/DAS sends and receives messages by subscribing to a topic or using named queues.</td>
<td>transport.jms.Destination</td>
</tr>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider.</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td>The JMS connection password</td>
<td>A valid password for the JMS connection.</td>
<td>transport.jms.Password</td>
</tr>
<tr>
<td>The JMS connection username</td>
<td>A valid username for the JMS connection.</td>
<td>transport.jms.UserName</td>
</tr>
</tbody>
</table>
### Connection Factory JNDI Name
The JNDI name of the connection factory.

```
transport.jms.ConnectionFactoryJNDIName
```

### Destination Type
Define the sort order for messages that arrive on a specific destination.

```
transport.jms.DestinationType
```

### Enable Durable Subscription
Whether the subscription is durable or not.

```
transport.jms.SubscriptionDurable
```

### Durable Subscriber Name
A valid name of the durable subscriber (If any value added, Durable subscription will be enabled).

```
transport.jms.DurableSubscriberName
```

### JMS Properties
Valid property and value pairs to denote Axis2 JMS properties (e.g. "property1: value1, property2: value2")

For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation.

```
jms.properties
```

**Related samples**

For more information on WSO2 MB event receiver type, see the following sample.

- [Sample 0013 - Receiving Map, Text Events via JMS Transport - WSO2 MB](#)

### Kafka Event Receiver

The Apache Kafka event receiver reads the tail of a given file and inputs that to the WSO2 product engine. This feature is donated by Andres Gomez Ferrer. For more information, go to Apache Kafka documentation.

- [Prerequisites](#)
- [Creating an Kafka event receiver](#)
- [Related samples](#)

#### Prerequisites

Set up the below prerequisites to start configuring an Apache Kafka event receiver.

1. Download Apache Kafka server.
2. Configure WSO2 CEP by adding relevant jars to support Kafka transport.
3. Start the Apache Kafka server. For more information, see Apache Kafka documentation.

#### Creating an Kafka event receiver

For instructions on creating an Apache Kafka event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the **Adapter Properties**, when creating an Apache Kafka event receiver using the management console as shown below.
## Create a New Event Receiver

<table>
<thead>
<tr>
<th>Enter Event Receiver Details</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Receiver Name*</td>
<td>KafkaInputEventAdapter</td>
</tr>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>Input Event Adapter Type*</td>
<td>kafka</td>
</tr>
</tbody>
</table>

### Adapter Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Zookeeper IP*</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>Group ID Kafka*</td>
<td></td>
</tr>
<tr>
<td>Threads*</td>
<td>4</td>
</tr>
<tr>
<td>Optional Configuration Properties</td>
<td>Define optional configuration properties (eg property1.property_value1,property2.property_value2)</td>
</tr>
</tbody>
</table>

### To

| Event Stream*             | Test Stream 1.0.0 |

### Mapping Configuration

| Message Format*           | xml |

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see [Input Mapping Types](#).
<eventReceiver name="KafkaInputEventAdpater" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="kafka">
    <property name="topic">test_topic</property>
    <property name="zookeeper.connect">127.0.0.1</property>
    <property name="threads">4</property>
    <property name="optional.configuration">zk.sessiontimeout.ms:6000</property>
    <property name="group.id">groupid</property>
    <property name="events.duplicated.in.cluster">false</property>
  </from>
... 
</eventReceiver>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server</strong></td>
<td>IP address of the Zookeeper Server</td>
<td>zookeeper.connect</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>Zookeeper IP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Group ID</strong></td>
<td>Kafka consumer group id which uniquely identifies a set of consumers within the same consumer group</td>
<td>group.id</td>
<td>groupid</td>
</tr>
<tr>
<td><strong>Kafka</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threads</strong></td>
<td>Number of consumer threads</td>
<td>threads</td>
<td>4</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>Valid property and value pairs to denote optional configuration properties for Apache Kafka. (E.g. &quot;property1: value1, property2: value2&quot;)</td>
<td>optional.configuration</td>
<td>zk.sessiontimeout.ms:6000</td>
</tr>
<tr>
<td><strong>Configuration Properties</strong></td>
<td>For more information on Kafka properties, go to Apache Kafka Documentation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Topic Kafka</strong></td>
<td>Name of the Kafka topic to which, input messages are published</td>
<td>topic</td>
<td>test_topic</td>
</tr>
<tr>
<td><strong>Is events duplicated in cluster</strong></td>
<td>In a cluster whether the same event can reach two receiver nodes</td>
<td>events.duplicated.in.cluster</td>
<td>true/false</td>
</tr>
</tbody>
</table>

**Related samples**
For more information on kafka event receiver type, see the following sample.

- Sample 0018 - Receiving JSON Events via Kafka Transport

**MQTT Event Receiver**

MQTT event receiver is an internal event receiver that comes with WSO2 products. You can configure it with XML, JSON, and text input mapping types.

- Prerequisites
- Creating a MQTT event receiver
- Related samples

**Prerequisites**

Follow the steps below before starting the MQTT event receiver configurations.

1. Configure WSO2 CEP by adding relevant jars to support MQTT transport.
2. Start the MQTT-supported server.

**Creating a MQTT event receiver**

For instructions on creating a MQTT event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the Adapter Properties, when creating a MQTT event receiver using the management console as shown below.
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventReceiver name="mqttEventReceiver" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="mqtt">
    <property name="topic">sensordata</property>
    <property name="clientId">CEP-CONSUMER</property>
    <property name="url">tcp://localhost:1883</property>
    <property name="username">mqtt-user</property>
    <property name="password">mqtt-password</property>
    <property name="cleanSession">true</property>
  </from>
</eventReceiver>
```

The above adapter properties are described below.
### Adapter Property | Description | Configuration file property | Example
---|---|---|---
**Topic** | A valid name for the MQTT broker topic which is used to receive messages on the MQTT input event adapter. | topic | MQTTInputEventAdapter
**Broker Url** | MQTT broker URL. You can use the same URL for WSO2 MB (when offset is zero). | url | tcp://localhost:1883
**Username** | A valid username for the broker connection. | username | mqtt-user
**Password** | A valid password for the broker connection. | password | mqtt-password
**Clean Session** | Persist topic subscriptions and acknowledge positions across client sessions. | cleanSession | true/false
**ClientId** | Unique client ID used by the server to identify a client when it reconnects. Used for durable subscriptions or reliable delivery of messages. | clientId | clientid

#### Related samples

For more information on MQTT event receiver type, see the following sample.

- [Sample 0016 - Receiving JSON Events via MQTT Transport](#)

**SOAP Event Receiver**

SOAP event receiver is used to receive events in **XML** format via HTTP, HTTPS, and local transports.

- [Creating a SOAP event receiver](#)
- [Related samples](#)

#### Creating a SOAP event receiver

For instructions on creating a SOAP event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the Adapter Properties, when creating a SOAP event receiver using the management console as shown below.
After entering the transport type in adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport(s)</td>
<td>Transport type via which the events are received.</td>
<td>transports</td>
<td>http</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on soap event receiver type, see the following sample.

- Sample 0014 - Receiving XML Events via Soap Transport

**WebSocket Event Receiver**

WebSocket event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with XML, JSON, and text input mapping types.

The websocket event receiver should be used when the event source is a websocket server to which the CEP server need to connect in order to receive events. However, if the event source is a websocket client, the event source should connect to the inbuilt web socket server of WSO2 CEP. In such scenarios, use the WebSocket Local Event Receiver.

- Prerequisites
- Creating a WebSocket event receiver
- Related samples

**Prerequisites**

Start the WebSocket server, before starting the event receiver configurations.

**Creating a WebSocket event receiver**

For instructions on creating a WebSocket event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the **Adapter Properties**, when creating a WebSocket event receiver using the management console as shown below.
After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventReceiver name="WebsocketInputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="websocket">
    <property name="websocket.server.url">ws://localhost:9099</property>
    <property name="events.duplicated.in.cluster">false</property>
  </from>

  ..................
</eventReceiver>
```
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Socket Server URL</td>
<td>URL of the WebSocket server to which you want to connect to</td>
<td>websocket.server.url</td>
<td>ws://localhost:9099</td>
</tr>
<tr>
<td>Is events duplicated in cluster</td>
<td>In a cluster whether the same event can reach two receiver nodes</td>
<td>events.duplicated.in.cluster</td>
<td>true/false</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on websocket event receiver type, see the following sample.

- Sample 0019 - Receiving JSON Events via WebSocket Transport

**WebSocket Local Event Receiver**

WebSocket local event receiver is an internal event receiver that comes with WSO2 products by default. You can configure it with XML, text, and JSON input mapping types.

The websocket-local event receiver should be used if the event source is a web socket client. Such event sources need to connect to the inbuilt websocket server of WSO2 CEP for the CEP to receive events. However, if the event source is a web socket server, the CEP should connect to it in order to receive events. In such scenarios, use the WebSocket Event Receiver.

- Creating a WebSocket local event receiver
- Related samples

**Creating a WebSocket local event receiver**

For instructions on creating a WebSocket local event receiver, see Configuring Event Receivers.

Configuring adapter properties

There are not any adapter-specific properties for the WebSocket local event receiver as shown below.
When multi-tenancy is used, the URL formats used to receive events are as follows.

<table>
<thead>
<tr>
<th>Super-tenant/Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super-tenant</td>
<td>ws://localhost&quot;:&lt;CEP_Server_Port&gt;&quot;/inputwebsocket/&lt;receiver_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost&quot;:&lt;CEP_SSL_Server_Port&gt;&quot;/inputwebsocket/&lt;receiver_name&gt;</td>
</tr>
<tr>
<td>Tenant</td>
<td>ws://localhost&quot;:&lt;CEP_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/inputwebsocket/&lt;receiver_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost&quot;:&lt;CEP_SSL_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/inputwebsocket/&lt;receiver_name&gt;</td>
</tr>
</tbody>
</table>

E.g., if the receiver name is WebSocketLocalInputEventReceiver and the tenant domain is mycompany.com, the URL would be as follows when you use the default server ports.

<table>
<thead>
<tr>
<th>Super-tenant/Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
</table>
| Super-tenant | ws://localhost:";9763"/inputwebsocket/WebSocketLocalInputEventReceiver  
| Tenant       | ws://localhost:";9763"/t/mycompany.com/inputwebsocket/WebSocketLocalInputEventReceiver |
|             | wss://localhost:";9443"/inputwebsocket/WebSocketLocalInputEventReceiver |

**Related samples**

For more information on **websocket-local** event receiver type, see the following sample.

- **Sample 0020 - Simple JSON Pass-through with Websocket-Local Input Event Adapter**

**WSO2Event Event Receiver**

WSO2Event event receiver is used to receive events in the WSO2Event format via Thrift or binary protocols. By default it uses the following ports to retrieve events.

- **For Thrift:**
  - TCP port:7611
  - SSL port:7711
- **For Binary:**
  - TCP port:9611
  - SSL port:9711

Use the `tcp://<HOSTNAME>:<PORT>` and `ssl://<HOSTNAME>:<PORT>` URLs to send events to the server as follows.

- Use the following format for load-balancing:
  
    `{tcp://<HOSTNAME>:<PORT>, tcp://<hostname>:<PORT>, ...}`

- Use the following format for failover:
  
    `{tcp://<HOSTNAME>:<PORT> | tcp://<hostname>:<PORT> | ...}`

- Use the following format to send messages to more than one cluster of endpoints (cluster is defined using "[]"):
  
    `{tcp://<HOSTNAME>:<PORT> | tcp://<hostname>:<PORT> | ...}, {tcp://<hostname>:<PORT>}`

In the above format, the event is delivered to one endpoint on the first cluster of endpoints in a failover manner. Also, the same message is delivered to the endpoint defined in the second cluster.

**Creating a WSO2Event event receiver**

For instructions on creating a WSO2Event event receiver, see Configuring Event Receivers.

Configuring adapter properties

Specify the **Adapter Properties**, when creating a WSO2Event event receiver using the management console as shown below.
Create a New Event Receiver

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers` directory as follows.
<eventReceiver name="WSO2EventReceiver" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
  <from eventAdapterType="wso2event">
    <property name="events.duplicated.in.cluster">false</property>
  </from>
</eventReceiver>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is events duplicated in cluster</td>
<td>In a cluster whether the same event can reach two receiver nodes</td>
<td>events.duplicated.in.cluster</td>
<td>true/false</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on wso2event event receiver type, see the following samples.

- Sample 0007 - Receiving WSO2 Events Via WSO2Event Receiver
- Sample 0008 - Receiving Custom WSO2 Events via WSO2Event Receiver
- Sample 0101 - Pass-Through/Projection Query in an Execution Plan
- Sample 0112 - Analyzing Twitter Feeds Using Partitions
- Sample 0114 - Using External Time Windows
- Sample 0501 - Processing a Simple Filter Query with Apache Storm Deployment
- Sample 0504 - Processing a Distributed Siddhi Query with Partitioning by integrating with Apache Storm
- Sample 0072 - Publishing Map Events via RDBMS Transport

**Input Mapping Types**

By default, event receivers process incoming messages in the XML, JSON, Text, Map (Key-value pairs), and WSO2Event formats. If the incoming events adhere to a default format, select the supported default format for `Message Format` property under `Mapping Configuration` when creating event receivers.

However, if the incoming events do not adhere to a default format, when creating event receivers select the supported format for `Message Format`, click the **Advanced** section, and provide input mappings to convert the message to a canonical format for the server to understand the message and process it.

This section covers the following types of input event receiver mappings that WSO2 CEP/DAS supports and how to configure them.

- WSO2Event input mapping
- XML input mapping
- JSON input mapping
- Text input mapping
Map input mapping

WSO2Event input mapping

WSO2Event input mapping allows you to convert events from one WSO2Event format to another. You need to define both the incoming event stream and the mapped event stream for it. A sample mapping configuration is shown below.

### Mapping Configuration

<table>
<thead>
<tr>
<th>Message Format*</th>
<th>wso2event:</th>
</tr>
</thead>
</table>

#### Advanced

**WSO2Event Mapping**

<table>
<thead>
<tr>
<th>Input Event Stream*</th>
<th>sensor.stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Event Version*</td>
<td>1.0.6</td>
</tr>
</tbody>
</table>

**Meta Data**

<table>
<thead>
<tr>
<th>Input Attribute Name</th>
<th>Mapped To</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>meta_timestamp</td>
<td>long</td>
</tr>
<tr>
<td>meta_isPowerServed</td>
<td>meta_isPowerSaverEnable</td>
<td>bool</td>
</tr>
<tr>
<td>id</td>
<td>meta_sensorId</td>
<td>int</td>
</tr>
<tr>
<td>name</td>
<td>meta_sensorName</td>
<td>string</td>
</tr>
</tbody>
</table>

**Correlation Data**

<table>
<thead>
<tr>
<th>Input Attribute Name</th>
<th>Mapped To</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation_longitude</td>
<td>correlation_longitude</td>
<td>double</td>
</tr>
<tr>
<td>correlation_latitude</td>
<td>correlation_latitude</td>
<td>double</td>
</tr>
</tbody>
</table>

**Payload Data**

<table>
<thead>
<tr>
<th>Input Attribute Name</th>
<th>Mapped To</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>humid</td>
<td>humidity</td>
<td>float</td>
</tr>
<tr>
<td>value</td>
<td>sensorValue</td>
<td>double</td>
</tr>
</tbody>
</table>

The configuration XML file of the above sample mapping is as follows.
<eventReceiver ... xmlns="http://wso2.org/carbon/eventreceiver">
  <from ... />
  <mapping customMapping="enable" type="wso2event">
    <from streamName="sensor.stream" version="1.0.6"/>
    <property>
      <from dataType="meta" name="time"/>
      <to name="meta_timestamp" type="long"/>
    </property>
    <property>
      <from dataType="meta" name="meta_ispowerServed"/>
      <to name="meta_isPowerSaverEnabled" type="bool"/>
    </property>
    <property>
      <from dataType="meta" name="id"/>
      <to name="meta_sensorId" type="int"/>
    </property>
    <property>
      <from dataType="meta" name="name"/>
      <to name="meta_sensorName" type="string"/>
    </property>
    <property>
      <from dataType="correlation" name="correlation_longitude"/>
      <to name="correlation_longitude" type="double"/>
    </property>
    <property>
      <from dataType="correlation" name="correlation_latitude"/>
      <to name="correlation_latitude" type="double"/>
    </property>
    <property>
      <from dataType="payload" name="humid"/>
      <to name="humidity" type="float"/>
    </property>
    <property>
      <from dataType="payload" name="value"/>
      <to name="sensorValue" type="double"/>
    </property>
  </mapping>
  <to ... />
</eventReceiver>

---

Events with null or empty attributes

In case of an event is received with null or empty attributes the behaviour for WSO2Event Input mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - `null` - CEP will throw an error and the event will be dropped
  - Empty Attribute - Error will be thrown and event will be dropped

- **String Attributes**
null - It will be processed as a null value
Empty Attribute - It will be processed as an empty string

When events are sent in the wso2event format, it is possible to include arbitrary attributes. Arbitrary attributes are attributes that are not defined in the event stream to which the events are sent.

The following is a sample configuration of an event in the wso2event format with arbitrary attributes.

```java
Map<String, String> arbitraryData = new HashMap<>();
arbitrary.put("meterId", "m0011");
arbitrary.put("switchId", "s0023");
arbitrary.put("phaseId", "p0097");
Event event = new Event(streamDefinition.getStreamId(),
System.currentTimeMillis(), metaData, correlationData, payloadData,
arbitraryData);
```

The values for arbitrary attributes in events can be accessed for processing by an execution plan. This is done by selecting the Include Arbitrary check box to the following in the attribute list of the import stream.

arbitraryDataMap Object

For more information, see Creating a Standalone Execution Plan.

XML input mapping

XML input mapping allows you to convert events of any XML format to the server's canonical event format (WSO2Event) for processing. If the XML message comprises of more than one message, then you can specify a Parent Selector XPath Expression pointing to an XML tag where all its child elements will be considered as separate XML messages. A sample mapping configuration is shown below.
The configuration XML file of the above sample mapping is as follows.
Events with null or empty attributes

In case of an event is received with null or empty attributes the behaviour for XML Input mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - **null** - CEP will throw an error and the event will be dropped
  - Empty Attribute - Error will be thrown and event will be dropped
JSON input mapping

JSON input mapping allows you to convert events of any JSON format to the server's canonical event format (WSO2Event) for processing. If the JSON message is an array, all the array elements are considered as separate messages when mapping the event. A sample mapping configuration is shown below.

The configuration XML file of the above sample mapping is as follows.
<eventReceiver ... xmlns="http://wso2.org/carbon/eventreceiver">
  <from ... />
  <mapping customMapping="enable" type="json">
    <property>
      <from jsonPath="$.sensorData.time"/>
      <to name="meta_timestamp" type="long"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.powerSaving"/>
      <to default="true" name="meta_isPowerSaverEnabled" type="bool"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.id"/>
      <to name="meta_sensorId" type="int"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.name"/>
      <to default="---" name="meta_sensorName" type="string"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.long"/>
      <to name="correlation_longitude" type="double"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.lat"/>
      <to name="correlation_latitude" type="double"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.humidity"/>
      <to name="humidity" type="float"/>
    </property>
    <property>
      <from jsonPath="$.sensorData.value"/>
      <to name="sensorValue" type="double"/>
    </property>
  </mapping>
  <to ... />
</eventReceiver>

---

**Events with null or empty attributes**

In case of an event is received with null or empty attributes the behaviour for JSON Input mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - **null** - CEP will throw an error and the event will be dropped
  - **Empty Attribute** - Error will be thrown and event will be dropped
- **String Attributes**
Text input mapping

Text input mapping allows you to convert events of any text format to the server’s canonical event format (WSO2Event) for processing. Data from the text message are expected to use regular expression patterns. A sample mapping configuration is shown below.

The configuration XML file of the above sample mapping is as follows.
Events with null or empty attributes

In case of an event is received with null or empty attributes the behaviour for Text Input mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - `null` - CEP will throw an error and the event will be dropped
  - Empty Attribute - Error will be thrown and event will be dropped

- **String Attributes**
  - `null` - It will be processed as a `null` value
  - Empty Attribute - It will be processed as an empty string

Map input mapping

Map input mapping allows you to convert events of any Map (Key-value pairs) format to the server's canonical event format (WSO2Event) for processing. A sample mapping configuration is shown below.
The configuration XML file of the above sample mapping is as follows.

```xml
<map>
  <timestamp>meta_timestamp</timestamp>
  <isPowerSaverEnabled>meta_isPowerSaverEnabled</isPowerSaverEnabled>
  <id>meta_sensorid</id>
  <name>meta_sensorName</name>
  <long>correlation_longitude</long>
  <lat>correlation_latitude</lat>
  <humidity>humidity</humidity>
  <sensorValue>sensorValue</sensorValue>
</map>
```
Events with null or empty attributes

In case of an event is received with null or empty attributes the behaviour for Map Input mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - **null** - CEP will throw an error and the event will be dropped
  - Empty Attribute - Error will be thrown and event will be dropped

- **String Attributes**
  - **null** - It will be processed as a null value
Building Custom Event Receivers

In addition to the default receiver types, you can define your own custom receiver, which gives more flexibility to receive events that are sent to WSO2 products. Since each event receiver implementation is an OSGI bundle, you can deploy/undeploy it easily on the WSO2 product. To create a custom event receiver, import `org.wso2.carbon.event.input.adaptor.core` package with the provided skeleton classes/interfaces required by a custom receiver implementation.

- Implementing InputEventAdapter Interface
- Implementing InputEventAdapterFactory Class
- Exposing Custom Event Receiver as an OSGI Service
- Deploying Custom Event Receiver

Implementing InputEventAdapter Interface

`org.wso2.carbon.event.input.adaptor.core.InputEventAdapter` interface contains the event receiver logic that will be used to receive events. You should override the below methods when implementing your own custom receiver.

1. `void init(InputEventAdapterListener eventAdaptorListener) throws InputEventAdapterException`
   
   This method is called when initiating event receiver bundle. Relevant code segments which are needed when loading OSGI bundle can be included in this method.

2. `void testConnect() throws TestConnectionNotSupportedException, InputEventAdapterRuntimeException, ConnectionUnavailableException`

   This method checks whether the receiving server is available.

3. `void connect() throws InputEventAdapterRuntimeException, ConnectionUnavailableException`

   Method connect() will be called after calling the `init()` method. Intention is to connect to a receiving end and if it is not available "ConnectionUnavailableException" will be thrown.

4. `void disconnect()`

   `disconnect()` method can be called when it is needed to disconnect from the connected receiving server.

5. `void destroy()`

   The method can be called when removing an event receiver. The cleanups that has to be done when removing the receiver can be done over here.

6. `boolean isEventDuplicatedInCluster()`

   Returns a boolean output stating whether an event is duplicated in a cluster or not. This can be used in clustered deployment.

7. `boolean isPolling()`

   Checks whether events get accumulated at the adapter and clients connect to it to collect events.

Below is a sample File Tail Receiver implementation of the above described methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init</code></td>
<td>Called when initiating event receiver bundle.</td>
</tr>
<tr>
<td><code>testConnect</code></td>
<td>Checks whether the receiving server is available.</td>
</tr>
<tr>
<td><code>connect</code></td>
<td>Called after <code>init</code> method. Intended to connect to the receiving end.</td>
</tr>
<tr>
<td><code>disconnect</code></td>
<td>Called when disconnecting from the receiving server.</td>
</tr>
<tr>
<td><code>destroy</code></td>
<td>Called when removing an event receiver.</td>
</tr>
<tr>
<td><code>isEventDuplicatedInCluster</code></td>
<td>Checks event duplication in a cluster.</td>
</tr>
<tr>
<td><code>isPolling</code></td>
<td>Checks if events are polled.</td>
</tr>
</tbody>
</table>
public class FileTailEventAdapter implements InputEventAdapter {

    @Override
    public void init(InputEventAdapterListener eventAdapterListener) throws InputEventAdapterException {
        validateInputEventAdapterConfigurations();
        this.eventAdapterListener = eventAdapterListener;
    }

    @Override
    public void testConnect() throws TestConnectionNotSupportedException {
        throw new TestConnectionNotSupportedException("not-supported");
    }

    @Override
    public void connect() {
        createFileAdapterListener();
    }

    @Override
    public void disconnect() {
        if (fileTailerManager != null) {
            fileTailerManager.getTailer().stop();
        }
    }

    @Override
    public void destroy() {
    }

    @Override
    public boolean isEventDuplicatedInCluster() {
        return Boolean.parseBoolean(globalProperties.get(EventAdapterConstants.EVENTS_DUP_LICATED_IN_CLUSTER));
    }

    @Override
    public boolean isPolling() {
        return true;
    }

    private void validateInputEventAdapterConfigurations() throws InputEventAdapterException {
        String delayInMillisProperty =
                eventAdapterConfiguration.getProperties().get(FileTailEventAdapterConstants.EVENT_ADAPTER_DELAY_MILLIS);
        try{
            Integer.parseInt(delayInMillisProperty);
        } catch (NumberFormatException e){
            throw new InputEventAdapterException("Invalid value set for property Delay: " + delayInMillisProperty, e);
        }
    }
}
private void createFileAdapterListener() {
    if (log.isDebugEnabled()) {
        log.debug("New subscriber added for " +
                eventAdapterConfiguration.getName());
    }
    String delayInMillisProperty =
            eventAdapterConfiguration.getProperties().get(FileTailEventAdapterConstants.EVENT_ADAPTER_DELAY_MILLIS);
    int delayInMillis =
            FileTailEventAdapterConstants.DEFAULT_DELAY_MILLIS;
    if (delayInMillisProperty != null &&
            (!delayInMillisProperty.trim().isEmpty())) {
        delayInMillis = Integer.parseInt(delayInMillisProperty);
    }
    boolean startFromEnd = false;
    String startFromEndProperty =
            eventAdapterConfiguration.getProperties().get(FileTailEventAdapterConstants.EVENT_ADAPTER_START_FROM_END);
    if (startFromEndProperty != null &&
            (!startFromEndProperty.trim().isEmpty())) {
        startFromEnd = Boolean.parseBoolean(startFromEndProperty);
    }
    String filePath = eventAdapterConfiguration.getProperties().get(FileTailEventAdapterConstants.EVENT_ADAPTER_CONF_FILEPATH);
    FileTailerListener listener = new FileTailerListener(new
            File(filePath).getName(), eventAdapterListener);
    Tailer tailer = new Tailer(new File(filePath), listener,
            delayInMillis, startFromEnd);
    fileTailerManager = new FileTailerManager(tailer, listener);
Implementing InputEventAdapterFactory Class

`org.wso2.carbon.event.input.adapter.core. InputEventAdapterFactory` class can be used as the factory to create your appropriate event receiver type. You should override the below methods when extending your own custom receiver.

1. **public String getType()**
   
   This method returns the receiver type as a String.

2. **public List<String> getSupportedMessageFormats()**
   
   Specify supported message formats for the created receiver type.

3. **public List<Property> getPropertyList()**
   
   Here the properties have to be defined for the receiver. When defining properties you can implement to configure property values from the management console.

4. **public String getUsageTips()**
   
   Specify any hints to be displayed in the management console.

5. **public InputEventAdapter createEventAdapter(InputEventAdapterConfiguration eventAdapterConfiguration, Map<String, String> globalProperties)**
   
   This method creates the receiver by specifying event adapter configuration and global properties which are common to every adapter type.

Below is a sample File Tail Receiver implementation of the InputEventAdapterFactory class:

```java
public class FileTailEventAdapterFactory extends InputEventAdapterFactory { 
    @Override
    public String getType() {
        return FileTailEventAdapterConstants.EVENT_ADAPTER_TYPE_FILE;
    }

    @Override
    public List<String> getSupportedMessageFormats() {
        List<String> supportInputMessageTypes = new ArrayList<String>();
        supportInputMessageTypes.add(MessageType.TEXT);
        return supportInputMessageTypes;
    }

    @Override
    public List<Property> getPropertyList() {
        List<Property> propertyList = new ArrayList<Property>();
        Property filePath = new Property(FileTailEventAdapterConstants.EVENT_ADAPTER_CONF_FILEPATH);
        filePath.setDisplayName(
```
filePath.setRequired(true);

filePath.setHint(resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_CONF_FILEPATH_HINT));
propertyList.add(filePath);

Property delayInMillis = new Property(FileTailEventAdapterConstants.EVENT_ADAPTER_DELAY_MILLIS);
delayInMillis.setDisplayName(resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_DELAY_MILLIS));
delayInMillis.setHint(resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_DELAY_MILLIS_HINT));
propertyList.add(delayInMillis);

Property startFromEndProperty = new Property(FileTailEventAdapterConstants.EVENT_ADAPTER_START_FROM_END);
startFromEndProperty.setRequired(true);
startFromEndProperty.setDisplayName(resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_START_FROM_END));
startFromEndProperty.setOptions(new String[] {"true", "false"});
startFromEndProperty.setDefaultValue("true");
startFromEndProperty.setHint(resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_START_FROM_END_HINT));
propertyList.add(startFromEndProperty);

return propertyList;

@Override
public String getUsageTips() {
    return resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_USAGE_TIPS_FILE);
}

@Override
public InputEventAdapter createEventAdapter(InputEventAdapterConfiguration eventAdapterConfiguration,
                                             Map<String, String> globalProperties) {
    return new FileTailEventAdapter(eventAdapterConfiguration,
                                     globalProperties);
Exposing Custom Event Receiver as an OSGI Service

Apart from above, you can maintain a service class under `internal\ds` directory to expose the custom event receiver implementation as an OSGI service. When exposing the service, it needs to expose as “InputEventAdaptorFactory” type. Below is a sample implementation for a service class for a File Tail Receiver:
public class FileTailEventAdapterServiceDS {
    private static final Log log = LogFactory.getLog(FileTailEventAdapterServiceDS.class);

    protected void activate(ComponentContext context) {
        try {
            InputEventAdapterFactory testInEventAdapterFactory = new FileTailEventAdapterFactory();
            context.getBundleContext().registerService(InputEventAdapterFactory.class.getName(),
                testInEventAdapterFactory, null);
            if (log.isDebugEnabled()) {
                log.debug("Successfully deployed the TailFile input event adapter service");
            }
        } catch (RuntimeException e) {
            log.error("Can not create the TailFile input event adapter service ", e);
        }
    }

    protected void setConfigurationContextService(
        ConfigurationContextService configurationContextService) {
        FileTailEventAdapterServiceHolder.registerConfigurationContextService(configurationContextService);
    }

    protected void unsetConfigurationContextService(
        ConfigurationContextService configurationContextService) {
        FileTailEventAdapterServiceHolder.unregisterConfigurationContextService(configurationContextService);
    }
}
Deploying a custom event receiver is very simple in WSO2 CEP 4.0.0. Simply implement the custom event receiver type, build the project and copy the created OSGI bundle that is inside the “target” folder into the `<CEP_HOME>/repository/components/dropins`. In CEP server startup, you can see the newly created event receiver type service in `ository/components/dropins` the server startup logs. The newly created custom event receiver type will also be visible in the UI with necessary properties. Now you can create several instances of this event receiver type.

Analyzing Data

Events can be processed on Realtime using Siddhi CEP engine. Sequence of events having the same type are defined as a stream and processing by Siddhi using SQL like Siddhi Query Language (SiddhiQL), Siddhi works as an upside down database by storing all the queries in memory, matching incoming events against the stored the queries at runtime and producing matched events as streams.

In-order to process events the execution logic should be written as a script using SiddhiQL and deployed in the server. This Execution Plan will act as an isolated execution environment and there will be an instance of Siddhi Execution Runtime created for each Execution Plan. An Execution Plan can import one or more streams from the server for processing and push zero or more output streams back to the server. In-order to import and export streams the script will also contain import and export annotations bindings the streams on the server with the streams defined in the Execution Plan.

Creating a Standalone Execution Plan

WSO2 CEP uses Execution Plans to store event processing logic. An Execution Plan is bound to an instance of the Siddhi complex event processing runtime, which is responsible for the actual processing of the events. The event processor allows users to configure multiple Execution Plans where by providing multiple isolated event processing environments per Execution Plan. A typical Execution Plan consists of a set of queries and related input and output Event Streams.

Writing an execution plan

To write an execution plan, start the CEP, log in to its management console, go to Main --> Manage --> Event Processor --> Execution Plans and click Add Execution Plan.

You will get an execution plan template as shown below.
You can edit this template to create a new execution plan. Follow the steps below to edit the template and create a new execution plan.

The execution plan editor supports auto completion.

To view the suggestions made by the editor, press control+space keys together.

The suggestions contain two sets.

1. Siddhi keywords, in alphabetical order
2. All the other words which are already inserted into the editor. For example, stream names which are defined in step 2 will be suggested when writing the queries in step 3. These will appear in alphabetical order after the keyword list.

In addition to the above, press shift+2 keys together to view suggestions on annotations.

**Supported annotations**

The annotations supported for execution plans are as follows.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>@plan:name (&lt;name for execution plan&gt;)</td>
<td>The name of the execution plan.</td>
<td>@Plan:name('ExecutionPlan')</td>
</tr>
<tr>
<td>Annotation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>@plan:description</td>
<td>This provides a description about the execution plan. Details such as the business requirement of the execution plan can be mentioned here.</td>
<td></td>
</tr>
<tr>
<td>@plan:async</td>
<td>WSO2 CEP has a disruptor based implementation to handle streams that are disabled by default. This annotation can be used to enable the disruptor at execution plan level that affects all the streams in the execution plan. The buffer size is an optional parameter. The default buffer size is 1024.</td>
<td></td>
</tr>
<tr>
<td>@async(bufferSize=&lt;event queue size&gt;)</td>
<td>Using this you can enable the disruptor only for a specific event stream. The event flow of the specified stream passes through the disruptor when enabled.</td>
<td></td>
</tr>
</tbody>
</table>

```java
@async(bufferSize='2')
define stream sensor_stream (meta_sensorId int, humidity float, sensorValue double);
```
| **@import (<CEP stream definition>)** | This is used to import a CEP event stream definition and map it to a Siddhi stream definition. For more information about event stream definitions, see Understanding Event Streams. | @Import('org.wso2.event.sensor.stream:1.0.0')
define stream sensor_stream (meta_sensorId int, humidity float, sensorValue double); |
| **@export (<CEP stream definition>)** | This is used to export a Siddhi event stream definition and map it to a CEP event stream definition. | @Export('org.wso2.sensor.value.projected.stream:1.0.0')
define stream sensor_value_projected_stream (meta_sensorId int, humidity float, value double); |

---

Step 1. Add execution plan info

**Add Execution Plan Name:**

Give a meaningful name to the execution plan by replacing 'ExecutionPlan' in @Plan:name('ExecutionPlan') with the new name.

For example, if the new name should be 'NewExecutionPlan', then replace @Plan:name('ExecutionPlan') with @Plan:name('NewExecutionPlan').

**Description:**

This is the description of the execution plan. Giving a description is optional.

To give a description to the execution plan,

- Uncomment the line "-- @Plan:description('ExecutionPlan');".
- Replace 'ExecutionPlan' with the description

For example, @Plan:description('This is the description for my NewExecutionPlan')

Step 2. Import Streams

An execution plan processes one or more streams. Therefore it is mandatory to import streams into an execution plan.

Importing a stream means mapping an available Event Stream to another internal stream. This internal stream is then used in query expressions, which will be written in step 3: Add query expressions. In fact, this
To import a stream:

1. In the **Import Stream** field, select an existing stream from the list.
2. In the **As** field, enter a meaningful name by which this stream should be referred.
3. Select the **Include Arbitrary** check box if you want to append arbitrary configurations to the stream definition when performing the import.

This imports an available event stream as a new stream.

The form inputs are described as follows:

- **Import Stream**: The name of the CEP Event Stream that should feed its events to the execution plan.
- **As**: The name to be used when feeding the stream to the Siddhi engine. This can contain only alphanumeric characters and underscore (_).
- **Include Arbitrary**: If this check box is selected for an import stream, the `arbitrary.data='true'` setting is added to the import stream definition in the execution plan. In addition, an attribute named `arbitraryDataMap` Object is added to the attribute list of the input stream definition in the execution plan. The keys and values on this attribute of the `object` type can be accessed via the `map extension of Siddhi`. This allows arbitrary data of the import stream to be copied to the export stream. Arbitrary data are events that are received without the `meta_` or a `correlation_` prefix, and at the same time do not match the payload attributes defined in an event stream.

**Step 3. Add query expressions**

Contains the event processing logic written in **Siddhi query language**. When defining more than one query, each query should end with a semi colon.

**Step 4. Export streams**

Defines the mappings between the exported (output) stream of the Siddhi runtime to one of the available Event Streams (defined inside query expressions). The parameters are as follows:
Value Of: The name of the stream exposed by the Siddhi runtime. This can contain only alphanumeric characters and underscore (_).

StreamId: The CEP Event Stream's ID to which the output events are sent from the execution plan.

Include Arbitrary: If this check box is selected, the `arbitrary.data='true'` setting is added to the export stream definition in the execution plan. This allows you to publish arbitrary data from that stream.

- It is not mandatory to define export streams in an execution plan.
- Siddhi Event tables cannot be exposed as streams. Event tables are only considered as streams within Siddhi.

Step 5. Add execution plan

Before adding the execution plan to the Siddhi runtime, it can be validated by clicking Validate Query Expressions.

Finally click the Add Execution Plan button to deploy the Execution plan.

Once an execution plan is created as saved, its configuration in WSO2 Siddhi Query Language format is saved in the `<CEP_HOME>/repository/deployment/server/executionplans` directory.

Editing a deployed execution plan

1. After an execution plan is successfully created, you can change it and redeploy it as follows.
2. Go to Main -> Manage -> Event Processor -> Execution Plans and select the execution plan needs to be edited by clicking on Edit button link associated with it. You can also click other associated links to delete it, enable/disable statistics or enable/disable tracing.
3. Edit the execution plan as required, and click Update.
4. Alternatively, you can write your execution plan in a text file and save it with the extension `.siddhiql` (which stands for Siddhi Query Language) and drop it in the deployment directory `<PRODUCT_HOME>/repository/deployment/server/executionplans`. Since hot deployment is supported you can simply add/remove Execution Plan files to deploy/undeploy Execution Plans from the server.

Deleting a deployed execution plan

Follow the procedure below to delete an execution plan.

1. Start the CEP, and log in to its management console. Click Main and then click Execution Plans to open the Available Execution Plans page. The available execution plans are listed in this page.
2. Click Delete in the row of the execution plan you want to delete. Click Yes in the message which appears to confirm whether the execution plan should be deleted.

Creating a Storm Based Distributed Execution Plan

When WSO2 CEP is run in a distributed set up with Apache Storm, Siddhi queries are executed inside Siddhi engines embedded in the bolts of the Apache Storm topology (for detailed information about Apache Storm topology, see Apache Storm Tutorial).
**Creating a distributed execution plan**

Execution plans in a CEP distributed set up should be created in the Management Console of a CEP manager. The Management consoles of CEP workers cannot be used for this purpose.

The procedure for creating a Storm based distributed execution plan is the same as creating a stand alone execution plan.

**Supported annotations**

The annotations supported for Apache Storm distributed execution plans are as follows.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>@dist (parallel='&lt;number of Storm tasks&gt;')</td>
<td>The number of storm tasks in which the query should be run parallel.</td>
<td>@dist(parallel='4')</td>
</tr>
<tr>
<td>@dist(execGroup='name of the group')</td>
<td>All the Siddhi queries in a particular execGroup will be executed in a single Siddhi bolt.</td>
<td>@dist(execGroup='Filtering')</td>
</tr>
<tr>
<td>@Plan:dist(receiverParallelism='number of receiver spouts')</td>
<td>The number of event receiver spouts to be spawned for the Storm topology.</td>
<td>@Plan:dist(receiverParallelism='1')</td>
</tr>
<tr>
<td>@Plan:dist(publisherParallelism='number of publisher bolts')</td>
<td>The number of event publisher bolts to be spawned for the Storm topology.</td>
<td>@Plan:dist(publisherParallelism='4')</td>
</tr>
</tbody>
</table>

The following is an example of an execution plan populated with the annotations mentioned above.
Generating the Apache Storm topology

Once an execution plan is created and submitted, it is converted to a query plan which defines the components and layout of the storm topology generated for the execution plan in XML. The query plan contains configurations such as input/output streams, number of instances to be spawned which siddhi queries to be executed inside(for bolts), etc.

```java
/* Enter a unique ExecutionPlan */
@Plan:name('PreprocessStat2')
@Plan:dist(receiverParallelism = '1')
@Plan:dist(publisherParallelism = '4')

/* Enter a unique description for ExecutionPlan */
-- @Plan:description('ExecutionPlan')

/* define streams/tables and write queries here ... */

@Import('analytics_Statistics:1.3.0')
define stream analyticsStats (meta_ipAdd string, meta_index long, meta_timestamp long, meta_nanoTime long, userID string, searchTerms string);

@Export('unprocessedStream:1.0.0')
define stream unprocessed (meta_ipAdd string, meta_index long, meta_timestamp long, meta_nanoTime long, userID string, searchTerms string);

@Export('filteredStatStream:1.0.0')
define stream filteredStatStream (meta_ipAdd string, meta_index long, meta_timestamp long, meta_nanoTime long, userID string);

@name('query1') @dist(parallel='4', execGroup='Filtering')
from analyticsStats[meta_ipAdd != '192.168.1.1']
select meta_ipAdd, meta_index, meta_timestamp, meta_nanoTime, userID
insert into filteredStatStream;

@name('query2') @dist(parallel='4', execGroup='Filtering')
from analyticsStats[meta_ipAdd == '192.168.1.1']
select *
insert into unprocessed;
```

Once an execution plan is created as saved, it's configuration in WSO2 Siddhi Query Language format is saved in the `<CEP_HOME>/repository/deployment/server/executionplans` directory.

**Note:**
Every Siddhi query in a particular `execGroup` should have the same number of tasks as shown in the execution plan above (e.g., parallel = '4'). If the queries need to be distributed across different siddhi bolts, the `execGroup` names of the queries should differ from each other.

If you want to view the query plan, add `log4j.logger.org.wso2.carbon.event.processor.core. internal.storm.StormTopologyManager=DEBUG` in the `<CEP_HOME>/repository/conf/log4j.properties` file.

Once the storm topology is generated based on the query plan, it is submitted to Apache Storm together with the org.wso2.cep.storm.dependencies.jar file. This jar file contains the classes required to run the topology in the
Storm cluster. It consists of dependencies such as Siddhi, etc. The Nimbus (i.e. the manager of storm cluster) distributes it across the storm cluster. This jar file is located in the `<CEP_HOME>/repository/conf/cep/storm` directory.

Once the topology is successfully submitted to Apache Storm, a log similar to the following example is printed in the wso2carbon.log file of the CEP manager.

```
Successfully uploaded topology jar to assigned location: storm-local/nimbus/inbox/stormjar:2568e169-911e-4ced-a703-683dc3029718.jar
```  

**Naming execution plans**

A name can be specified for each execution plan with the `@Plan:name` annotation. The CEP manager creates the Storm topology with the name specified in this annotation. In addition, the tenant ID is appended to the name of the topology in order to avoid name collisions of storm topologies belonging to different tenants in a multi-tenanted environment.

For example, if an execution plan is created with the `@Plan:name('StockAnalysis')` annotation for the super-tenant with tenant ID 1234, the Storm topology can be viewed as follows in the Storm UI.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Uptime</th>
<th>Num workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>StockAnalysis[1234]</td>
<td>ACTIVE</td>
<td>31s</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Checking the status of the execution plan**

Before publishing data to an event flow with a distributed execution plan, check whether the following conditions are met to ensure that the execution plan is ready to process data.

- The associated storm topology is in the `ACTIVE` state in the storm cluster.
- All connections between the CEP workers and the event receiver spouts (inflow connections) are made.
- All connections between the event publisher bolts and CEP workers (outflow connections) are made.

To check whether the above conditions are met, click **Main => Execution Plans** in the Management Console of the relevant CEP manager. The status of the execution plan is displayed in the `Distributed Deployment Status` column as shown in the example below.

```
<table>
<thead>
<tr>
<th>Execution Plan Name</th>
<th>Description</th>
<th>Distributed Deployment Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>StockAnalysis</td>
<td></td>
<td>Storm topology : ACTIVE</td>
</tr>
</tbody>
</table>
```

**Things to note**

- If you change the name of an execution plan after creating and saving it, remove the Storm topology generated with the previous name from the Apache Storm UI.
- The number of execution plans that are allowed to be created in a CEP distributed deployment is
See the following samples for more information.

- Sample 0501 - Processing a Simple Filter Query with Apache Storm Deployment
- Sample 0504 - Processing a Distributed Siddhi Query with Partitioning by integrating with Apache Storm

**SiddhiQL Guide 3.1**

This guide provides instructions to use the Siddhi Query Language 3.1 with WSO2 CEP using examples.

- Introduction to Siddhi Query Language
- Event stream
  - Event stream definition
- Query
  - Query projection
    - Function parameters
    - Inbuilt Functions
  - Filter
  - Window
    - Inbuilt Windows
  - Output event categories
  - Aggregate functions
    - Inbuilt aggregate functions
  - Group by
  - Having
  - Output rate limiting
    - Based on number of events
    - Based on time
    - Periodic snapshot
  - Joins
  - Outer joins
    - Left outer join
    - Right outer join
    - Full outer join
  - Pattern
    - Logical pattern
    - Counting pattern
  - Sequence
    - Logical sequence
    - Counting sequence
- Partition
  - Variable partition
  - Range partition
  - Inner streams
- Event table
  - Event table definition
    - Indexing event table
    - Hazelcast event table
Introduction to Siddhi Query Language

Siddhi Query Language (SiddhiQL) is designed to process event streams to identify complex event occurrences. The following table provides definitions of a few terms in the Siddhi Query Language.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream</td>
<td>A logical series of events ordered in time.</td>
</tr>
<tr>
<td>Event Stream Definition</td>
<td>This defines event streams. An event stream has a unique name and a set of attributes assigned specific types, with uniquely identifiable names defining its schema.</td>
</tr>
<tr>
<td>Event</td>
<td>An event is associated with only one event stream, and all events of that stream have an identical set of attributes assigned specific types (or the same schema). An event contains a timestamp and the attribute values according to the schema.</td>
</tr>
<tr>
<td>Attribute</td>
<td>An attribute has a unique name within the event stream. The attribute type can be string, int, long, float, double, bool or object.</td>
</tr>
<tr>
<td>Query</td>
<td>A logical construct that derives new streams by combining existing streams. A query contains one or more input streams, handlers to modify those input streams, and an output stream to which it publishes its output events.</td>
</tr>
<tr>
<td>Partition</td>
<td>A logical container that processes a subset of the queries based on a pre-defined rule of separation.</td>
</tr>
</tbody>
</table>
Siddhi has the following language constructs:

- Event Stream Definitions
- Event Table Definitions
- Partitions
- Queries

The execution logic of Siddhi can be composed together as an execution plan, and all the above language constructs can be written as script in an execution plan. Each construct should be separated by a semicolon ( ; ).

**Event stream**

An event stream is a sequence of events with a defined schema. One or more event streams can be imported and manipulated using queries in order to identify complex event conditions, and new event streams are created to notify query responses.

A type sequence of events that will have a defined schema, one or more events stream can be consumed and manipulated by queries in order to identify complex event conditions and new event streams could be emitted to notify query responses.

**Event stream definition**

The event stream definition defines the event stream schema. An event stream definition contains a unique name and a set of attributes assigned specific types, with uniquely identifiable names within the stream.

```sql
define stream <stream name> (<attribute name> <attribute type>, <attribute name> <attribute type>, ... );
```

E.g. A stream named `TempStream` can be created with the following attributes as shown below.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceID</td>
<td>long</td>
</tr>
<tr>
<td>roomNo</td>
<td>int</td>
</tr>
<tr>
<td>temp</td>
<td>double</td>
</tr>
</tbody>
</table>

```sql
define stream TempStream (deviceID long, roomNo int, temp double);
```

**Query**

Each Siddhi query can consume one or more event streams and create a new event stream from them.

All queries contain an input section and an output section. Some also contain a projection section. A simple query with all three sections is as follows.
Inferred Stream: Here the RoomTempStream is an inferred Stream, i.e. RoomTempStream can be used as an input query for another query without explicitly defining its Event Stream Definition. Because its Event Stream Definition is inferred from the above query.

Query projection

SiddhiQL supports the following for query projection.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selecting required objects for projection</td>
<td>This involves selecting only some of the attributes in an input stream to be inserted into an output stream.</td>
</tr>
<tr>
<td></td>
<td>e.g., The following query selects only the roomNo and temp attributes from the TempStream.</td>
</tr>
</tbody>
</table>

```sql
from TempStream
select roomNo, temp
insert into RoomTempStream;
```
### Selecting all attributes for projection

This involves selecting all the attributes in an input stream to be inserted into an output stream. This can be done by using the asterisk sign (\*) or by omitting the `select` statement.

**e.g.,** Use one of the following queries to select all the attributes in the `TempStream` stream.

```
from TempStream
select *
insert into NewTempStream;
```

or

```
from TempStream
insert into NewTempStream;
```

### Renaming attributes

This involves selecting attributes from the input streams and inserting them into the output stream with different names.

**e.g.,** The following query renames `roomNo` to `roomNumber` and `temp` to `temperature`.

```
from TempStream
select roomNo as roomNumber, temp as temperature
insert into RoomTempStream;
```

### Introducing the default value

This involves adding a default value and assigning it to an attribute using `as`.

**e.g.,**

```
from TempStream
select roomNo, temp, 'C' as scale
insert into RoomTempStream;
```

### Using mathematical and logical expressions

This involves using attributes with mathematical and logical expressions to the precedence order given below, and assigning them to the output attribute using `as`.

**Operator precedence**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Distribution</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Scope</td>
<td><code>(cost + tax) * 0.05</code></td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>IS NULL</td>
<td>Null check</td>
<td>deviceID is null</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT</td>
<td>not (price &gt; 10)</td>
</tr>
<tr>
<td>* / %</td>
<td>Multiplication, division, modulo</td>
<td>temp * 9/5 + 32</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition, subtraction</td>
<td>temp * 9/5 + 32</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Comparisons: less-than, greater-than-equal, greater-than, less-than-equal</td>
<td>totalCost &gt;= price * quantity</td>
</tr>
<tr>
<td>== !=</td>
<td>Comparisons: equal, not equal</td>
<td>totalCost &gt;= price * quantity</td>
</tr>
<tr>
<td>IN</td>
<td>Contains in table</td>
<td>roomNo in ServerRoomsTable</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND</td>
<td>temp &lt; 40 and (humidity &lt; 40 or humidity &gt;= 60)</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR</td>
<td>temp &lt; 40 and (humidity &lt; 40 or humidity &gt;= 60)</td>
</tr>
</tbody>
</table>

e.g., Converting Celsius to Fahrenheit and identifying server rooms
from TempStream
select roomNo, temp * 9/5 + 32 as temp, 'F' as scale,
roomNo >= 100 and roomNo < 110 as isServerRoom
insert into RoomTempStream;

Functions

A function consumes zero, one or more function parameters and produces a result value.

Function parameters

Functions parameters can be attributes (int, long, float, double, string, bool, object), results of other functions, results of mathematical or logical expressions or time parameters.

Time is a special parameter that can be defined using the time value as int and its unit type as <int> <unit>. Following are the supported unit types, Time upon execution will return its expression in the scale of milliseconds as a long value.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>year</td>
</tr>
<tr>
<td>Month</td>
<td>month</td>
</tr>
<tr>
<td>Week</td>
<td>week</td>
</tr>
<tr>
<td>Day</td>
<td>day</td>
</tr>
<tr>
<td>Hour</td>
<td>hour</td>
</tr>
<tr>
<td>Minutes</td>
<td>minute</td>
</tr>
<tr>
<td>Seconds</td>
<td>second</td>
</tr>
<tr>
<td>Milliseconds</td>
<td>millisecond</td>
</tr>
</tbody>
</table>

E.g. Passing 1 hour and 25 minutes to test function.

    test(1 hour 25 min)

Functions, mathematical expressions, and logical expressions can be used in a nested manner.

Inbuilt Functions

Siddhi supports the following inbuilt functions.

- coalesce
- convert
- instanceOfBoolean
- instanceOfDouble
- instanceOfFloat
- instanceOfInteger
- instanceOfLong
- instanceOfString
- UUID
- cast
- ifThenElse

e.g., The following configuration converts the room number to string and adds a message ID to each event using `convert` and `UUID` functions.

```java
from TempStream
select convert(roomNo, 'string') as roomNo, temp, UUID() as messageID
insert into RoomTempStream;
```

Filter

Filters can be used with input streams to filter events based on the given filter condition. Filter conditions should be defined in square brackets next to the input stream name as shown below.

```java
from <input stream name>[<filter condition>]
select <attribute name>, <attribute name>, ... 
insert into <output stream name>
```

e.g., The following configuration filters all server rooms having a temperature greater than 40 degrees.

```java
from TempStream [(roomNo >= 100 and roomNo < 110) and temp > 40 ]
select roomNo, temp
insert into HighTempStream;
```

Window

Windows allow you to capture a subset of events based on criterion from input event stream for calculation. They can be defined next to input streams using the `'#window.'` prefix. Each input stream can only have maximum of one window as follows.

```java
from <input stream name>[<filter condition>]#window.<window name>(<parameter>, <parameter>, ... )
select <attribute name>, <attribute name>, ... 
insert into <output stream name>
```

Windows emit two events for each event they consume: they are current-events and expired-events. A window emits current-event when a new event arrives at the window and emits expired-event whenever an event in a window expires based on that window criteria.
**Inbuilt Windows**

Siddhi supports the following inbuilt windows.

- time
- timeBatch
- length
- lengthBatch
- externalTime
- cron
- firstUnique
- unique
- sort
- frequent
- lossyFrequent

**Output event categories**

Window output can be manipulated based event categories, i.e. current and expired events, use the following keywords with output stream to manipulate the output.

- **current events**: Will emit all the events that arrives to the window. This is the default functionality if no event category is specified.
- **expired events**: Will emit all the events that expires from the window.
- **all events**: Will emit all the events that arrives and expires from the window.

For using with insert into statement use the above keywords between 'insert' and 'into' as given in the example below.

E.g. Delay all events in a stream by 1 minute.

```
from TempStream#window.time(1 min)
select *
insert expired events into DelayedTempStream
```

**Aggregate functions**

Aggregate functions can be used with windows to perform aggregate calculations within the defined window.

**Inbuilt aggregate functions**

Siddhi supports the following inbuilt aggregate functions.

- sum
- average
- max
- min
- count
- stddev

E.g. Notify upon all event arrival and expiry the average temperature of all rooms based on all events arrived during last 10 minutes.
Group by

Group by allows us to group the aggregation based on group by attributes.

E.g. Find the average temperature per room and device ID for the last 10 min.

```
from TempStream#window.time(10 min)
select avg(temp) as avgTemp, roomNo, deviceID
group by roomNo, deviceID
insert into AvgTempStream;
```

Having

Having allows us to filter events after aggregation and after processing at the selector.

E.g. Find the average temperature per room for the last 10 min and alert if it's more than 30 degrees.

```
from TempStream#window.time(10 min)
select avg(temp) as avgTemp, roomNo
group by roomNo
having avgTemp > 30
insert into AlertStream;
```

Output rate limiting

Output rate limiting allows queries to emit events periodically based on the condition specified.

Rate limiting follows the below syntax.

```
from <input stream name>...
select <attribute name>, <attribute name>, ...
output ({<output-type>} every (<time interval>|<event interval> events) | snapshot every <time interval>)
insert into <output stream name>
```

With "<output-type>" the number of events that need to be emitted can be specified, "first", "last" and "all" are possible key wards that can be specified to emit only the first event, last event, or all events from the arrived events. If the key word is omitted it will default to "all" emitting all events.

With "<time interval>" the time interval for the periodic event emotion can be specified.

With "<event interval>" the number of event need to be arrived for the periodic event emotion can be specified.
**Based on number of events**

Here the events will be emitted every time when the predefined number of events have arrived, when emitting it can be specified to emit only the first event, last event, or all events from the arrived events.

E.g., Emit the last temperature event per sensor every 10 events

```java
from TempStream
select temp
    group by deviceID
output last every 10 events
    insert into LowRateTempStream;
```

**Based on time**

Here the events will be emitted for every predefined time interval, when emitting it can be specified to emit only the first event, last event, or all events from the arrived events.

E.g. Emit the all temperature events every 10 seconds

```java
from TempStream
output every 10 sec
    insert into LowRateTempStream;
```

**Periodic snapshot**

This works best with windows, when the input stream as a window attached snapshot rate limiting will emit all current events arrived so far which does not have corresponding expired events for every predefined time interval, at the same time when no window is attached to the input stream it will only emit the last current event for every predefined time interval.

E.g. Emit snapshot of the events in time window of 5 seconds every one second.

```java
from TempStream#window.time(5 sec)
output snapshot every 1 sec
    insert into SnapshotTempStream;
```

**Joins**

Join allows two event streams to be merged based on a condition. Here each stream should be associated with a window (if there are no window assigned `#window.length(0)` with be assigned to the input event stream). During the joining process each incoming event on each stream will be matched against all events in the other input event stream window based on the given condition and for all matching event pairs an output event will be generated.

The syntax of join looks like below.
With "on <join condition>" Siddhi joins only the events that matches the condition.

With "unidirectional" keyword the trigger of joining process can be controlled. By default events arriving on both streams trigger the joining process and when unidirectional keyword is used on an input stream only the events arriving on that stream will trigger the joining process. Note we cannot use unidirectional keyword for both the input streams (as thats equal to the default behaviour, which is not using the unidirectional keyword at all).

With "within <time gap>" the joining process matched the events that are within defined time gap of each other.

When projecting the join events the attributes of each stream need to be referred with the stream name (E.g. <stream name>.<attribute name>) or with its reference Id (specially when events of same streams are joined) (E.g. <stream reference Id>.<attribute name>), "select *" can be used or "select" statement itself can be omitted if all attributes of the joined events need to be projected, but these can only be used when both streams does not have any attributes with same names.

e.g., Assume that the temperature regulators are updated every minute. The following switches on the temperature regulators if they are not already on for all the rooms that have a room temperature greater than 30 degrees.

```
define stream TempStream(deviceID long, roomNo int, temp double);
define stream RegulatorStream(deviceID long, roomNo int, isOn bool);

from TempStream[temp > 30.0]#window.time(1 min) as T
   join RegulatorStream[isOn == false]#window.length(1) as R
   on T.roomNo == R.roomNo
   select T.roomNo, R.deviceID, 'start' as action
insert into RegulatorActionStream;
```

**Outer joins**

The syntax of an outer join is as follows.
from <input stream name>[@<filter condition>]#window.<window name>[@<parameter>, ...] (unidirectional) {as <reference>}
  (left|right|full) outer join <input stream name>[@window.<window name>[@<parameter>, ...] (unidirectional) {as <reference>}
  on <join condition>
  within <time gap>
  select <attribute name>, <attribute name>, ...
  insert into <output stream name>

Left outer join

Outer join allows two event streams to be merged based on a condition. However, it returns all the events of left stream even if there are no matching events in the right stream. Here each stream should be associated with a window. During the joining process, each incoming event of each stream is matched against all the events in the other input event stream window based on the given condition. Incoming events of the right stream are matched against all events in the left event stream window based on the given condition. An output event is generated for all the matching event pairs. An output event is generated for incoming events of the left stream even if there are no matching events in right stream.

e.g., The following generates output events for all the events in the stockStream stream whether there is a match for the symbol in the twitterStream stream or not.

```
from stockStream#window.length(2)  
left outer join twitterStream#window.length(1)
on stockStream.symbol== twitterStream.symbol 
select stockStream.symbol as symbol, twitterStream.tweet, stockStream.price 
insert all events into outputStream ;
```

Right outer join

This is similar to left outer join and, it returns all the events of right stream even if there are no matching events in the left stream. Incoming events of the left stream are matched against all events in the right event stream window based on the given condition. An output event is generated for all the matching event pairs. An output event is generated for incoming events of the right stream even if there are no matching events in left stream.

e.g., The following generates output events for all the events in the twitterStream stream whether there is a match for the symbol in the stockStream stream or not.

```
from stockStream#window.length(2)  
right outer join twitterStream#window.length(1)
on stockStream.symbol== twitterStream.symbol 
select stockStream.symbol as symbol, twitterStream.tweet, stockStream.price 
insert all events into outputStream ;
```

Full outer join

The full outer join combines the results of left outer join and right outer join. An output event is generated for each incoming event even if there are no matching events in the other stream.
e.g., The following generates output events for all the incoming events of each stream whether there is a match for the symbol in the other stream or not.

```java
from stockStream#window.length(2)
full outer join twitterStream#window.length(1)
on stockStream.symbol== twitterStream.symbol
select stockStream.symbol as symbol, twitterStream.tweet, stockStream.price
insert all events into outputStream;
```

**Pattern**

Pattern allows event streams to be correlated over time and detect event patterns based on the order of event arrival. With pattern there can be other events in between the events that match the pattern condition. It will internally create state machines to track the states of the matching process. Pattern can correlate events over multiple input streams or over the same input stream, hence each matched input event need to be referenced such that it can be accessed for future processing and output generation.

The syntax of pattern looks like below.

```java
from {every} <input event reference>=<input stream name>[<filter condition>] -> {every} <input event reference>=<input stream name>[<filter condition>] -> ...
within <time gap>
select <input event reference>.<attribute name>, <input event reference>.<attribute name>, ...
insert into <output stream name>
```

Input Streams cannot be associated with a window.

With "->" we can correlate incoming events arrivals, having zero or many other events arrived in between the matching events.

With "<input event reference>=" the matched event can be stored for future reference.

With "within <time gap>" the pattern will be only matched with the events that are within defined time gap of each other.

Without "every" keyword the pattern can be match only once, use the "every" keyword appropriately to trigger a pattern matching process upon event arrival.

E.g. Alert if temperature of a room increases by 5 degrees within 10 min.

```java
from every( e1=TempStream ) -> e2=TempStream[e1.roomNo==roomNo and (e1.temp + 5) <= temp ]
within 10 min
select e1.roomNo, e1.temp as initialTemp, e2.temp as finalTemp
insert into AlertStream;
```

**Logical pattern**
Pattern not only matches event arriving on the temporal order but it can also correlate events having logical relationships.

Keywords like "and" and "or" can be used interred of "->" to illustrate the logical relationship.

With "and" occurrence of two events in any order can be matched

With "or" occurrence of an event from either of the input steams in any order can be matched

E.g. Alert when the room temperature reaches the temperature set on the regulator, (the pattern matching should be reseted whenever the temperature set on the regulator changes).

```java
define stream TempStream(deviceID long, roomNo int, temp double);
define stream RegulatorStream(deviceID long, roomNo int, tempSet double);
from every( e1=RegulatorStream ) -> e2=TempStream[e1.roomNo==roomNo and e1.tempSet <= temp ] or e3=RegulatorStream[e1.roomNo==roomNo]
select e1.roomNo, e2.temp as roomTemp
having e3 is null
insert into AlertStream;
```

**Counting pattern**

Counting pattern enable us to match multiple events based on the same matching condition. The expected number of events can be limited using the following postfix.

With <1:4> matches 1 to 4 events

With <2:> matches 2 or more events and with <:5> up to 5 events.

With <5> matches exactly 5 events.

To refer the specific occurrences of the event what are matched based on count limits, square brackets could be used with numerical and "last" keywords, such as e1[3] will refer to the third event, e1[last] will refer to the last event and e1[last - 1] will refer to the event before the last event of the matched event group.

E.g Get the temperature difference between two regulator events.

```java
define stream TempStream(deviceID long, roomNo int, temp double);
define stream RegulatorStream(deviceID long, roomNo int, tempSet double, isOn bool);
from every( e1=RegulatorStream ) -> e2=TempStream[e1.roomNo==roomNo]<1:> -> e3=RegulatorStream[e1.roomNo==roomNo]
select e1.roomNo, e2[0].temp - e2[last].temp as tempDiff
insert into TempDiffStream;
```

**Sequence**

Sequence allows event streams to be correlated over time and detect event sequences based on the order of event
arrival. With sequence there can not be other events in between the events that match the sequence condition. It will internally create state machines to track the states of the matching process. Sequence can correlate events over multiple input streams or over the same input stream, hence each matched input event need to be referenced such that it can be accessed for future processing and output generation.

The syntax of sequence looks like below.

```
from {every} <input event reference>=<input stream name>[<filter condition>], <input event reference>=<input stream name>[<filter condition>] (+|*|?), ..., within <time gap>
select <input event reference>.<attribute name>, <input event reference>.<attribute name>, ...
insert into <output stream name>
```

Input Streams cannot be associated with a window.

With ",," we can correlate immediate next incoming events arrivals, having no other events arrived in between the matching events.

With "<input event reference>=" the matched event can be stored for future reference.

With "within <time gap>" the sequence will be only matched with the events that are within defined time gap of each other.

Without "every" keyword the pattern can be match only once, use the "every" keyword in the beginning to trigger a sequence matching process upon every event arrival.

E.g. Alert if there is more than 1 degree increase in temperature between two consecutive temperature events.

```
from every e1=TempStream, e2=TempStream[e1.temp + 1 < temp]
select e1.temp as initialTemp, e2.temp as finalTemp
insert into AlertStream;
```

**Logical sequence**

Sequence not only matches consecutive event arriving on the temporal order but it can also correlate events having logical relationships.

Keywords like "and" and "or" can be used interred of ",," to illustrate the logical relationship.

With "and" occurrence of two events in any order can be matched

With "or" occurrence of an event from either of the input steams in any order can be matched

E.g. Notify when a regulator event is followed by both the temperature and humidity events.
define stream TempStream(deviceID long, temp double);
define stream HumidStream(deviceID long, humid double);
define stream RegulatorStream(deviceID long, isOn bool);

from every e1=RegulatorStream, e2=TempStream and e3=HumidStream
select e2.temp, e3.humid
insert into StateNotificationStream;

**Counting sequence**

Counting sequence enable us to match multiple consecutive events based on the same matching condition. The expected number of events can be limited using the following postfix.

With "**" zero or more events can be matched.

With "+" one or more events can be matched.

With "?" zero or one events can be matched.

To refer the specific occurrences of the event what are matched based on count limits, square brackets could be used with numerical and "last" keywords, such as e1[3] will refer to the third event, e1[last] will refer to the last event and e1[last - 1] will refer to the event before the last event of the matched event group.

E.g Identify peak temperatures.

```
define stream TempStream(deviceID long, roomNo int, temp double);
define stream RegulatorStream(deviceID long, roomNo int, tempSet double, isOn bool);

from every e1=TempStream, e2=TempStream[e1.temp <= temp]+,
e3=TempStream[e2[last].temp > temp]
select e1.temp as initialTemp, e2[last].temp as peakTemp
insert into TempDiffStream;
```

**Partition**

With partition Siddhi can divide both incoming events & queries and process them parallel in isolation. Each partition will be tagged with a partition key and only the events corresponding to the given partition key will be processed at that partition. Each partition will have separate instances of Siddhi queries providing isolation of processing states. Partition can contain more than one query.

Partition key can be defined using the categorical (string) attribute of the input event stream as Variable Partition or by defining separate ranges when the partition need to be defined using numeral attributes of the input event stream as Range Partition.

**Variable partition**

Partition using categorical (string) attributes will adhere to the following syntax.
partition with ( <attribute name> of <stream name>, <attribute name> of <stream name>, ... )
begin
<query>
<query>
...
end;

E.g. Per sensor, calculate the maximum temperature over last 10 temperature events each sensor has emitted.

partition with ( deviceID of TempStream )
begin
from TempStream#window.length(10)
select roomNo, deviceID, max(temp) as maxTemp
insert into DeviceTempStream
end;

Range partition
Partition using numerical attributes will adhere to the following syntax.

partition with ( <condition> as <partition key> or <condition> as <partition key> or ... of <stream name>, ... )
begin
<query>
<query>
...
end;

E.g. Per office area calculate the average temperature over last 10 minutes.

partition with ( roomNo>=1030 as 'serverRoom' or roomNo<1030 and roomNo>=330 as 'officeRoom' or roomNo<330 as 'lobby' of TempStream )
begin
from TempStream#window.time(10 min)
select roomNo, deviceID, avg(temp) as avgTemp
insert into AreaTempStream
end;

Inner streams
Inner streams can be used for query instances of a partition to communicate between other query instances of the same partition. Inner Streams are denoted by a "#" in front of them, and these streams cannot be accessed outside of the partition block.

E.g. Per sensor, calculate the maximum temperature over last 10 temperature events when the sensor is having an
average temperature greater than 20 over the last minute.

partition with ( deviceID of TempStream )
begin
  from TempStream#window.time(1 min)
  select roomNo, deviceID, temp, avg(temp) as avgTemp
  insert into #AvgTempStream

  from #AvgTempStream[avgTemp > 20]#window.length(10)
  select roomNo, deviceID, max(temp) as maxTemp
  insert into deviceTempStream
end;

_Event table_

Event table allows Siddhi to work with stored events, and this can be viewed as a stored version of Event Stream or a table of events. By default events will be stored in-memory and Siddhi also provides an extension to work with data/events stored in RDBMS data stores.

_Event table definition_

Event Table Definition defines the event table schema. An Event Table Definition contains a unique name and a set of typed attributes with uniquely identifiable names within the table.

```plaintext
define table <table name> (<attribute name> <attribute type>, <attribute name> <attribute type>, ... );
```

With the above definition, events will be stored in-memory via a linked list data structure.

E.g. Room type table with name RoomTypeTable can be created as below with attributes room number as int and type as string.

```plaintext
define table RoomTypeTable (roomNo int, type string);
```

_Indexing event table_

Event table can be indexed for fast event access using the "IndexBy" annotation. With "IndexBy" only one attribute can be indexed, and when indexed it uses a map data structure to hold the events. Therefore if multiple events are inserted to the event table having the same index value only last inserted event will remain in the table.

E.g. An indexed room type table with attribute room number can be created as bellow with name RoomTypeTable and attributes room number as int & type as string.

```plaintext
@indexBy('roomNo')
define table RoomTypeTable (roomNo int, type string);
```

_Hazelcast event table_
Event tables also support persisting event data in a distributed manner using Hazelcast in-memory data grids. This functionality can easily be enabled using the "From" annotation. Also, the connection instructions for Hazelcast cluster can be assigned to the event table using "From" annotation.

Connection instructions that can be used with "From" annotation:

- `cluster.name`: Hazelcast cluster/group name [Optional] (i.e. `cluster.name='cluster_a'`).
- `cluster.password`: Hazelcast cluster/group password [Optional] (i.e. `cluster.password='pass@cluster_a'`).
- `cluster.addresses`: Hazelcast cluster addresses (ip:port) as a comma separated string [Optional, client mode only] (i.e. `cluster.addresses='192.168.1.1:5700,192.168.1.2:5700'`).
- `well.known.addresses`: Hazelcast WKAs (ip) as a comma separated string [Optional, server mode only] (i.e. `well.known.addresses='192.168.1.1,192.168.1.2'`).
- `collection.name`: Hazelcast collection object name [Optional, can be used to share single table between multiple EPs] (i.e. `collection.name='stockTable'`).

E.g. 1. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by a new Hazelcast Instance.

```java
@from(eventtable = 'hazelcast')
define table RoomTypeTable(roomNo int, type string);
```

E.g. 2. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by a new Hazelcast Instance in a new Hazelcast Cluster.

```java
@from(eventtable = 'hazelcast', cluster.name = 'cluster_a',
cluster.password = 'pass@cluster_a')
define table RoomTypeTable(roomNo int, type string);
```

E.g. 3. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by a existing Hazelcast Instance in an existing Hazelcast Cluster.

```java
@from(eventtable = 'hazelcast', cluster.name = 'cluster_a',
cluster.password = 'pass@cluster_a',
cluster.addresses='192.168.1.1:5700,192.168.1.2.5700')
define table RoomTypeTable(roomNo int, type string);
```

**RDBMS event table**

Event table can be backed with an RDBMS event store using the "From" annotation. With "From" the data source or the connection instructions can be assigned to the event table. The RDBMS table name can be different from the
event table name defined in Siddhi, and Siddhi will always refer to the defined event table name. However the defined event table name cannot be same as an already existing stream name, since syntactically both are considered the same with in the Siddhi query language.

RDBMS event table has been tested with the following databases:

- MySQL
- H2
- Oracle

E.g. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by RDBMS table named RoomTable from the data source named AnalyticsDataSource.

```sql
@From(eventtable='rdbms', datasource.name='AnalyticsDataSource',
      table.name='RoomTable')
define table RoomTypeTable (roomNo int, type string);
```

### Note

The `datasource.name` given here is injected to the Siddhi engine by the CEP server. To configure data sources in the CEP, see [Datasources](#) in the Admin Guide.

E.g. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by MySQL table named RoomTable from the database cepdb located at localhost:3306 having user name "root" and password "root".

```sql
@From(eventtable='rdbms', jdbc.url='jdbc:mysql://localhost:3306/cepdb',
       username='root', password='root', driver.name='com.mysql.jdbc.Driver',
       table.name='RoomTable')
define table RoomTypeTable (roomNo int, type string);
```

### Caching events with RDBMS event table

Several caches can be used with RDBMS backed event tables in order to reduce I/O operations and improve their performance. Currently all cache implementations provides size-based algorithms. Caches can be added using the "cache" element and the size of the cache can be defined using the "cache.size" element of the "From" annotation.

The supported cache implementations are as follows;

1. **Basic**: Events are cached in a FIFO manner where the oldest event will be dropped when the cache is full.
2. **LRU (Least Recently Used)**: The least recently used event is dropped when the cache is full.
3. **LFU (Least Frequently Used)**: The least frequently used event is dropped when the cache is full.

In the "From" annotation, if the "cache" element is not specified the "Basic" cache will be assigned by default, and if the "cache.size" element is not assigned the default value 4096 will be assigned as the cache size.

E.g. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by RDBMS table using least recently used caching algorithm for caching 3000 events.
Using Bloom filters

A Bloom Filter is an algorithm or an approach that can be used to perform quick searches. If you apply a Bloom Filter to a data set and carry out an `isAvailable` check on that specific Bloom Filter instance, an accurate answer is returned if the search item is not available. This allows the quick improvement of updates, joins and `isAvailable` checks.

The following example shows how to include Bloom Filters in an event table update query.

```sql
@from(eventtable='rdbms', datasource.name='AnalyticsDataSource',
       table.name='RoomTable', cache='LRU', cache.size='3000')
define table RoomTypeTable (roomNo int, type string);

define stream StockStream (symbol string, price float, volume long);
define stream CheckStockStream (symbol string, volume long);
@from(eventtable = 'rdbms' ,datasource.name = 'cepDB' , table.name =
       'stockInfo' , bloom.filters = 'enable')
define table StockTable (symbol string, price float, volume long);

@info(name = 'query1')
from StockStream
insert into StockTable ;

@info(name = 'query2')
from CheckStockStream[(StockTable.symbol==symbol) in StockTable]
insert into OutStream;
```

For more information about In-memory event tables, see Sample 0106 - Using in-memory event tables

For more information about RDBMS event tables, see Sample 0107 - Using RDBMS event tables

Insert into

Query for inserting events into table is similar to the query of inserting events into event streams, where we will be using "insert into <table name>" code snippet. To insert only the specified output event category use "current events", "expired events" or "all events" keywords between 'insert' and 'into' keywords.

E.g. Insert all temperature events from TempStream to temperature table

```sql
from TempStream
select *
insert into TempTable;
```

Delete

Query for deleting events on event table can be written using a delete query having following syntax
Here the "on <condition>" can be used to select the events for deletion, and when writing this condition attribute names of the event tables should be always referred with table name and attributes of the select should not be have reference associated with them.

E.g. Delete the entries of the RoomTypeTable associated to the room numbers of DeleteStream.

```
define table RoomTypeTable (roomNo int, type string);
define stream DeleteStream (roomNumber int);

from DeleteStream
delete RoomTypeTable
  on RoomTypeTable.roomNo == roomNumber;
```

To execute delete only for the specified output event category instead of "delete <table name> on <condition>" code snippet use "delete <table name> for <output event category> on <condition>", where "<output event category>" could be "current events", "expired events" or "all events" keywords.

**Update**

Query for updating events on event table can be written using an update query having following syntax

```
from <input stream name>
select <attribute name> as <table attribute name>, <attribute name> as <table attribute name>, ...
update <table name>
on <condition>
```

Here the "on <condition>" can be used to select the events for update, and when writing this condition attribute names of the event tables should be always referred with table name and attributes of the select should not be have reference associated with them.

With "<table attribute name>" the attributes could be referred with the same name thats defined in event table, allowing Siddhi to identify which attributes need to be updated on event table.

E.g. For each room denoted by its number, update the room types of the RoomTypeTable based on the event in UpdateStream.

```
define table RoomTypeTable (roomNo int, type string);
define stream UpdateStream (roomNumber int);

from UpdateStream
update RoomTypeTable
  on RoomTypeTable.roomNo == roomNumber;
```
define table RoomTypeTable (roomNo int, type string);
define stream UpdateStream (roomNumber int, roomType string);

from UpdateStream
select roomType as type
update RoomTypeTable
on RoomTypeTable.roomNo == roomNumber;

To execute update only for the specified output event category instead of "update <table name> on <condition>" code snippet use "update <table name> for <output event category> on <condition>", where "<output event category>" could be "current events", "expired events" or "all events" keywords.

**Insert Overwrite**

Query for insert or overwrite events on event table can be written using an insert-overwrite query having following syntax

```sql
from <input stream name>
select <attribute name> as <table attribute name>, <attribute name> as <table attribute name>, ...
insert overwrite <table name>
on <condition>
```

Here the "on <condition>" can be used to select the events for update or insert, and when writing this condition attribute names of the event tables should be always referred with table name and attributes of the select should not be have reference associated with them.

With "<table attribute name>" the attributes could be referred with the same name thats defined in event table, allowing Siddhi to identify which attributes need to be updated/inserted on event table.

E.g. For each room denoted by its number, update the room types of the RoomTypeTable based on the event sin UpdateStream or insert if it is not exist.

```sql
define table RoomTypeTable (roomNo int, type string);
define stream UpdateStream (roomNumber int, roomType string);

from UpdateStream
select roomNumber as roomNo, roomType as type
insert overwrite RoomTypeTable
on RoomTypeTable.roomNo == roomNo;
```

**In**

Query for checking whether an attribute is in event table can be checked using conditions having the following syntax

```sql
<condition> in <table name>
```
Here the "<condition>" can be used to select the matching attribute, and when writing this condition attribute names of the event tables should be always referred with table name and attributes of the incoming stream should not be have reference associated with them.

E.g. By checking ServerRoomTable output only the temperature events associated with the saver rooms.

```
define table ServerRoomTable (roomNo int);
define stream TempStream (deviceID long, roomNo int, temp double);

from TempStream[ServerRoomTable.roomNo == roomNo in ServerRoomTable]
insert into ServerTempStream;
```

Join

A stream can be joined with event table and retrieve data from the event table. In order to join a stream with an event table a simple join query could be used, and at join the event table should not be associated with window operations as event table is not an active construct. Because of the same reason event table cannot be joined with another event table in Siddhi.

E.g. Update the events in temperature stream with their room type based on the RoomTypeTable.

```
define table RoomTypeTable (roomNo int, type string);
define stream TempStream (deviceID long, roomNo int, temp double);

from TempStream join RoomTypeTable
on RoomTypeTable.roomNo == TempStream.roomNo
select deviceID, RoomTypeTable.roomNo as roomNo, type, temp
insert into EnhancedTempStream;
```

Event window

An event window is a window that can be shared across multiple queries. The events should be inserted from one or more streams. The event window publishes current and/or expired events as the output, and the time these events are published depends on the window type.

Event window definition

The syntax for an event window definition is as follows.

```
define window <event window name> (<attribute name> <attribute type>,
<attribute name> <attribute type>, ...) <window type>(<parameter>,
<parameter>, ...) <output event type>;
```

The above syntax contains the following:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;event window name&gt;</td>
<td>A unique name for the window.</td>
</tr>
</tbody>
</table>
<attribute name> <attribute type>
These elements define a set of attributes assigned specific types.

>window type>
Any inbuilt window type available in Siddhi. For the complete list of available window types, see Inbuilt Windows.

<output event type>
The possible values for this parameter are as follows:

- **output current events**: Windows with this output event type emit only current events.
- **output expired events**: Windows with this output event type emit only expired events.
- **output all events**: Windows with this output event type emit both current and expired events.

If the output event type is not specified for an event window, it emits both current and expired events.

**Sample event window definitions**

- The window type is not specified in the following window definition. Therefore, it emits both current and expired events as the output.

  ```
  define window SensorWindow (name string, value float, roomNo int, deviceID string) timeBatch(1 second);
  ```

- The window type of the following window is **output all events**. Therefore, it emits both current and expired events as the output.

  ```
  define window SensorWindow (name string, value float, roomNo int, deviceID string) timeBatch(1 second) output all events;
  ```

**Insert Into**

The query for inserting events into a window is similar to the query for inserting events into event streams where the `insert into <window name>` code snippet is used. To restrict the events inserted into the window by a specific category, use one of the following key words between the `insert` and `into` keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>current events</td>
<td>To insert only current events into the event window.</td>
</tr>
<tr>
<td>expired events</td>
<td>To insert only expired events into the event window.</td>
</tr>
<tr>
<td>all events</td>
<td>To insert both current and expired events into the event window.</td>
</tr>
</tbody>
</table>

e.g., The following query inserts both current and expired events from an event stream named `sensorStream` to an event window named `sensorWindow`. 
Output

An event window can be used as a stream in any query. However, an ordinary window cannot be applied to the output of an event window.

e.g., The following query selects the name and the maximum values for the value and roomNo attributes from an event window named SensorWindow, and inserts them into an event stream named MaxSensorReadingStream.

```sql
from SensorWindow
select name, max(value) as maxValue, roomNo
insert into MaxSensorReadingStream;
```

Join

An event window can be joined with another event stream, an event table or an event window.

e.g., The following query sends an alert to an event stream named RegulatorActionStream if the temperature is greater than 30 and the regulator is off.

```sql
define stream TempStream(deviceID long, roomNo int, temp double);
define stream RegulatorStream(deviceID long, roomNo int, isOn bool);
define window TempWindow(deviceID long, roomNo int, temp double) time(1 min);

from TempStream[temp > 30.0]
insert into TempWindow;

from TempWindow
join RegulatorStream[isOn == false]#window.length(1) as R
on TempWindow.roomNo == R.roomNo
select TempWindow.roomNo, R.deviceID, 'start' as action
insert into RegulatorActionStream;
```

Event trigger

Triggers allow us to create events periodically based on time and at Siddhi start. Event trigger will generate events on event stream with name same as the event trigger having only one attribute with name "triggered_time" and type long.

Event trigger definition

Event Trigger Definition defines the event triggering interval following the below syntax.
define trigger <trigger name> at {'start'| every <time interval>| '<cron expression>'};

With "start" an event will be triggered at Siddhi start.

With "every <time interval>" an event will be triggered periodically on the given time interval.

With "<cron expression>" an event will be triggered periodically based on the given cron expression, refer quartz-scheduler for config details.

E.g Trigger an event every 5 minutes.

define trigger FiveMinTriggerStream at every 5 min;

E.g Trigger an event at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.

define trigger FiveMinTriggerStream at '0 15 10 ? * MON-FRI';

**Siddhi logger**

The Siddhi logger is used to log events that arrive in different logger priorities such as INFO, DEBUG, WARN, FATAL, ERROR, OFF, and TRACE.

The following syntax is used.

```<void> log(<string> priority, <string> logMessage, <bool> isEventLogged)```

The parameters used in the query are as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>The logging priority. Possible values are INFO, DEBUG, WARN, FATAL, ERROR, OFF, and TRACE. If no value is specified for this parameter, INFO is printed as the priority by default.</td>
</tr>
<tr>
<td>logMessage</td>
<td>This parameter allows you to specify a message to be printed in the log.</td>
</tr>
<tr>
<td>isEventLogged</td>
<td>This parameter specifies whether the event body should be included in the log. Possible values are true and false. If no value is specified, the event body is not printed in the log by default.</td>
</tr>
</tbody>
</table>

The following examples illustrate the variations of the Siddhi logger.

e.g.,

- The following query logs the event with the INFO logging priority. This is because the priority is not specified.
from StockStream#log()
select *
insert into OutStream;

The following query logs the event with the INFO logging priority (because the priority is not specified) and the test message text.

from StockStream#log('test message')
select *
insert into OutStream;

The following query logs the event with the INFO logging priority because a priority is not specified. The event itself is printed in the log.

from StockStream#log(true)
select *
insert into OutStream;

The following query logs the event with the INFO logging priority (because the priority is not specified) and the test message text. The event itself is printed in the log.

from StockStream#log('test message', true)
select *
insert into OutStream;

The following query logs the event with the WARN logging priority and the test message text.

from StockStream#log('warn','test message')
select *
insert into OutStream;

The following query logs the event with the WARN logging priority and the test message text. The event itself is printed in the log.

from StockStream#log('warn','test message',true)
select *
insert into OutStream;

**Eval script**

Eval script allows Siddhi to process events using other programming languages by defining functions by them. Eval script functions can be defined like event tables or streams and referred in the queries as Inbuilt Functions of Siddhi.
Eval script definition

Eval Script Definition defines the function operation following the below syntax.

```plaintext
define function <function name>[<language name>] return <return type> {
    <operation of the function>
};
```

With "<function name>" a function will be defined to be using in the queries. Note this function will overwrite the Inbuilt Functions.

With "<language name>" the execution language will be defined e.g. JavaScript, R, Scala.

With "<return type>" the return type of the function is defined, it can either be int, long, float, double, string, bool or object. Here the function implementer should be responsible for returning the output on the defined <return type> for proper functionality.

With "<operation of the function>" the execution logic of the function should be written on the defined language defined in <language name> returning <return type>.

Supported Eval Script Languages

- JavaScript
- R
- Scala

JavaScript

E.g Concatenating function in JavaScript.

```javascript
define function concatFn[JavaScript] return string {
    var str1 = data[0];
    var str2 = data[1];
    var str3 = data[2];
    var response = str1 + str2 + str3;
    return response;
};

define stream TempStream(deviceID long, roomNo int, temp double);

from TempStream
select concatFn(roomNo,'-',deviceID) as id, temp
insert into DeviceTempStream;
```

R

E.g Concatenating function in R.

```r
```
define function concatFn[R] return string {
    return(paste(data, collapse=""));
};

define stream TempStream(deviceID long, roomNo int, temp double);

from TempStream
select concatFn(roomNo,'-',deviceID) as id, temp
insert into DeviceTempStream;

Scala
E.g Concatenating function in Scala.

define function concatFn[Scala] return string {
    var concatenatedString =
    for(i <- 0 until data.length){
        concatenatedString += data(i).toString
    }
    concatenatedString
};

define stream TempStream(deviceID long, roomNo int, temp double);

from TempStream
select concatFn(roomNo,'-',deviceID) as id, temp
insert into DeviceTempStream;

Siddhi extensions

Siddhi supports an extension architecture to support custom code and functions to be incorporated with Siddhi in a seamless manner. Extension will follow the following syntax;

<namespace>:<function name>(<parameter1>, <parameter2>, ...)

Here the namespace will allow Siddhi to identify the function as an extension and its extension group, the function name will denote the extension function within the given group, and the parameters will be the inputs that can be passed to the extension for evaluation and/or configuration.

E.g. A window extension created with namespace foo and function name unique can be referred as follows:

from StockExchangeStream[price >= 20]#window.foo:unique(symbol)
select symbol, price
insert into StockQuote

Extension types
Siddhi supports following five type of extensions:

**Function Extension**

For each event it consumes zero or more parameters and output a single attribute as an output. This could be used to manipulate event attributes to generate new attribute like Function operator. Implemented by extending "org.wso2.siddhi.core.executor.function.FunctionExecutor".

E.g. "math:sin(x)" here the sin function of math extension will return the sin value its parameter x.

**Aggregate Function Extension**

For each event it consumes zero or more parameters and output a single attribute having an aggregated results based in the input parameters as an output. This could be used with conjunction with a window in order to find the aggregated results based on the given window like Aggregate Function operator. Implemented by extending "org.wso2.siddhi.core.query.selector.attribute.aggregator.AttributeAggregator".

E.g. "custom:std(x)" here the std aggregate function of custom extension will return the standard deviation of value x based on the assigned window to its query.

**Window Extension**

Allows events to be collected and expired without altering the event format based on the given input parameters like the Window operator. Implemented by extending "org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor".

E.g. "custom:unique(key)" here the unique window of custom extension will return all events as current events upon arrival as current events and when events arrive with the same value based on the "key" parameter the corresponding to a previous event arrived the previously arrived event will be emitted as expired event.

**Stream Function Extension**

Allows events to be altered by adding one or more attributes to it. Here events could be outputted upon each event arrival. Implemented by extending "org.wso2.siddhi.core.query.processor.stream.function.StreamFunctionProcessor".

E.g. "custom:pol2cart(theta,rho)" here the pol2cart function of custom extension will return all events by calculating the cartesian coordinates x & y and adding them as new attributes to the existing events.

**Stream Processor Extension**

Allows events to be collected and expired with altering the event format based on the given input parameters. Implemented by extending "org.wso2.siddhi.core.query.processor.stream.StreamProcessor".

E.g. "custom:perMinResults(arg1, arg2, ...)" here the perMinResults function of custom extension will return all events by adding one or more attributes the events based on the conversion logic and emitted as current events upon arrival as current events and when at expiration expired events could be emitted appropriate expiring events attribute values for matching the current events attributes counts and types.

**Available Extensions**

Siddhi currently have several prewritten extensions as follows;

Extensions released under Apache License v2:

- **math**: Supporting mathematical operations
- **str**: Supporting String operations
- **geo**: Supporting geocode operations
• regex: Supporting regular expression operations
• time: Supporting time expression operations
• ml: Supporting Machine Learning expression operations
• timeseries: Supporting Time Series operations
• kf (Kalman Filter): Supporting filtering capabilities by detecting outliers of the data.
• map: Supporting to send a map object inside Siddhi stream definitions and use it inside queries.
• reorder: Supporting for reordering events from an unordered event stream using Kslack algorithm.

Extensions released under GNU/GPL License v3:

• geo: Supporting geographical processing operations
• r: Supporting R executions
• nlp: Supporting Natural Language Processing expression operations
• pmml: Supporting Predictive Model Markup Language expression operations

You can get them from https://github.com/wso2-gpl/siddhi

Writing Custom Extensions

Custom extensions can be written in order to cater usecase specific logics that are not out of the box available in Siddhi or as an extension.

To create custom extensions two things need to be done.

1. Implementing the extension logic by extending well defined Siddhi interfaces. E.g implementing a UniqueWindowProcessor by extending org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor.

```java
package org.wso2.test;

public class UniqueWindowProcessor extends WindowProcessor {
    ...
}
```

2. Add an extension mapping file to map the written extension class with the extension function name and namespace. Here extension mapping file should be named as "<namespace>.siddhiext". E.g Mapping the written UniqueWindowProcessor extension with function name "unique" and namespace "foo", to do so the mapping file should be named as foo.siddhiext and the context of the file should as below;

```plaintext
# function name to class mapping of 'foo' extension
unique=org.wso2.test.UniqueWindowProcessor
```

Refer following for implementing different types of Siddhi extensions with examples

• Custom Function Extension
• Custom Aggregate Function Extension
• Custom Window Extension
• Custom Stream Function Extension
• Custom Stream Processor Extension

Inbuilt Functions

Following are the supported inbuilt functions of Siddhi
### coalesce

**Syntax**

```<int|long|float|double|string|bool|object>
coalesce(<int|long|float|double|string|bool|object> arg1, <int|long|float|double|string|bool|object> arg2,..., <int|long|float|double|string|bool|object> argN>)```

**Extension Type**

Function

**Description**

Returns the value of the first input parameter that is not null.

**Parameters**

This function accepts one or more parameters. They can belong to any one of the available types. All the specified parameters should be of the same type.

**Return Type**

This will be the same as the type of the first input parameter.

**Examples**

- `coalesce('123', null, '789')` returns 123
- `coalesce(null, 76, 567)` returns 76
- `coalesce(null, null, null)` returns null

### convert

**Syntax**

```<int|long|float|double|string|bool>
convert(<int|long|float|double|string|bool> BeConverted, <string> convertedTo)```

**Extension Type**

Function

**Description**

Converts the first input parameter according to the `convertedTo` parameter.
### Parameters

- **toBeConverted**: This specifies the value to be converted. The type of this value can be any of the available types other than `object`.
- **convertedTo**: A string constant parameter expressing the converted to type using one of the following strings values: `int, long, float, double, string, bool`.

### Return Type

The return type is the same as the type specified by the `convertedTo` parameter.

### Examples

- `convert('123', 'double')` returns `123.0`
- `convert(45.9, 'int')` returns `46`
- `convert(true, 'string')` returns `true`

---

#### instanceOfBoolean

**Syntax**

```<bool> instanceOfBoolean(<int, long, float, double, string, bool, object> arg)```

**Extension Type**

Function

**Description**

Checks whether the parameter is an instance of Boolean or not.

**Parameters**

- **arg**: The parameter to be checked.

**Return Type**

Returns `bool`: `true` if the parameter is an instance of Boolean, or `false` if the parameter is not an instance of Boolean.

**Examples**

- `instanceOfBoolean(123)` returns `false`
- `instanceOfBoolean(true)` returns `true`
- `instanceOfBoolean(false)` returns `true`.

---

#### instanceOfDouble

**Syntax**

```<bool> instanceOfDouble(<int, long, float, double, string, bool, object> arg)```

**Extension Type**

Function

**Description**

Checks whether the parameter is an instance of Double or not.

**Parameters**

- **arg**: The parameter to be checked.

**Return Type**

Returns `bool`: `true` if the parameter is an instance of Double, or `false` if the parameter is not an instance of Double.
### Examples
- `instanceOfDouble(123)` returns `false`
- `instanceOfDouble(56.45)` returns `true`
- `instanceOfDouble(false)` returns `false`

### instanceOfFloat

| Syntax | `<bool> instanceOfFloat(<int|long|float|double|string|bool|object> arg)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Checks whether the parameter is an instance of Float or not. |
| Parameter | The parameter to be checked. |
| Return Type | Returns bool: `true` if the parameter is an instance of Float, or `false` if the parameter is not an instance of Float. |

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instanceOfFloat(123)</code> returns <code>false</code></td>
<td></td>
</tr>
<tr>
<td><code>instanceOfFloat(56.45)</code> returns <code>false</code></td>
<td></td>
</tr>
<tr>
<td><code>instanceOfFloat(56.45f)</code> returns <code>true</code></td>
<td></td>
</tr>
</tbody>
</table>

### instanceOfInteger

| Syntax | `<bool> instanceOfInteger(<int|long|float|double|string|bool|object> arg)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Checks whether the parameter is an instance of Integer or not. |
| Parameter | The parameter to be checked. |
| Return Type | Returns bool: `true` if the parameter is an instance of Integer, or `false` if the parameter is not an instance of Integer. |

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instanceOfInteger(123)</code> returns <code>true</code></td>
<td></td>
</tr>
<tr>
<td><code>instanceOfInteger(56.45)</code> returns <code>false</code></td>
<td></td>
</tr>
<tr>
<td><code>instanceOfInteger(56.45f)</code> returns <code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

### instanceOfLong

| Syntax | `<bool> instanceOfLong(<int|long|float|double|string|bool|object> arg)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Checks whether the parameter is an instance of Long or not. |
| Parameter | The parameter to be checked. |
### Return Type
Returns bool:
- `true` if the parameter is an instance of Long, or `false` if the parameter is not an instance of Long.

### Examples
- `instanceOfLong(123)` returns `false`
- `instanceOfLong(5667l)` returns `true`
- `instanceOfLong(56.67)` returns `false`

### instanceOfLong

| Syntax | `<bool> instanceOfLong(<int|long|float|double|string|bool|object> arg)` |
|---|---|
| Description | Checks whether the parameter is an instance of String or not. |
| Parameter | The parameter to be checked. |
| Return Type | Returns bool:
- `true` if the parameter is an instance of String, or `false` if the parameter is not an instance of String. |

### Examples
- `instanceOfString('test')` returns `true`
- `instanceOfString('5667')` returns `true`
- `instanceOfString(56.67)` returns `false`

### UUID

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; UUID()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Generates a UUID (Universally Unique Identifier).</td>
</tr>
<tr>
<td>Return Type</td>
<td>Returns a UUID string.</td>
</tr>
</tbody>
</table>

### Examples
- `UUID()` returns `a34eec40-32c2-44fe-8075-7f4fde2e2dd8`
- The following converts a room number to string, introducing a message ID to each event.

```java
from TempStream
select convert(roomNo, 'string') as roomNo, temp, UUID() as messageID
insert into RoomTempStream;
```

### maximum

| Syntax | `<int|long|float|double> maximum(<int|long|float|double>)` |
|---|---|
### maximum

**Syntax**

```<int|long|float|double> maximum(<int|long|float|double>)```

**Extension Type**

Function

**Description**

Returns the maximum value of the input parameters.

**Parameter**

This function accepts one or more parameters. They can belong to any one of the available types. All the specified parameters should be of the same type.

**Return Type**

This will be the same as the type of the first input parameter.

**Examples**

- `maximum(37.88, 38.12, 37.62)` returns `38.12`.
- `maximum(15, 30, 25, 57)` returns `57`.

---

### minimum

**Syntax**

```<int|long|float|double> minimum(<int|long|float|double>)```

**Extension Type**

Function

**Description**

Returns the minimum value of the input parameters.

**Parameter**

This function accepts one or more parameters. They can belong to any one of the available types. All the specified parameters should be of the same type.

**Return Type**

This will be the same as the type of the first input parameter.

**Examples**

- `minimum(37.88, 38.12, 37.62)` returns `37.62`.
- `minimum(15, 30, 25, 57)` returns `15`.

---

### cast

**Syntax**

```<int|long|float|double|string|bool|object > cast(<object> toBeCasted, <string> cast To)```  

**Extension Type**

Function

**Description**

Converts the first parameter according to the `castTo` parameter. Incompatible arguments cause `Class Cast` exceptions if further processed. This function is used with the `map` extension that returns attributes of the `object` type. You can use this function to cast the object to an accurate and concrete type.
### Parameters

- **toBeCasted**: This specifies the attribute to be casted.
- **castTo**: A string constant parameter expressing the cast to type using one of the following strings values: int, long, float, double, string, bool.

### Return Type

This is the same as the type of the second input parameter.

### Examples

- `cast(100.3, 'double')` returns `100.3d`
- `cast(true, 'double')` returns `true`
- `cast(null, 'double')` returns `null`

### ifThenElse

#### Syntax

```java
<bool> ifThenElse(<bool> condition, <int|long|float|double|string|bool> arg1, <int|long|float|double|string|bool|object> arg2)
```

#### Extension Type

Function

#### Description

Returns the the value of the `arg1` parameter if the `condition` parameter is set to `true`, or returns the value of the `arg2` parameter if the `condition` parameter is set to `false`.

#### Parameters

- **condition**: This specifies the if then else condition value. The data type for this parameter is `bool`.
- **arg1**: This specifies the value to be returned if the value of the `condition` parameter is `true`. The data type of the value specified should be `int`, `long`, `float`, `double`, `string`, `bool`, or `object`.
- **arg2**: This specifies the value to be returned if the value of the `condition` parameter is `false`. The data type of the value specified should be `int`, `long`, `float`, `double`, `string`, `bool`, or `object`.

#### Return Type

Returns `<int|long|float|double|string|bool|object>arg1` if the value of the `condition` parameter is `true`, or `<int|long|float|double|string|bool|object>arg2` if the value of the `condition` parameter is `false`. 
### Examples

- `ifThenElse(sensorValue > 35, 'High', 'Low')` returns `High` if `sensorValue = 50.`
- `ifThenElse(voltage < 5, 0, 1)` returns `1` if `voltage = 12.`
- `ifThenElse(password == 'admin', true, false)` returns `true` if `password = admin.`

### Inbuilt Windows

Following are the supported inbuilt windows of Siddhi:

- `time`
- `timeBatch`
- `length`
- `lengthBatch`
- `externalTime`
- `cron`
- `firstUnique`
- `unique`
- `sort`
- `frequent`
- `lossyFrequent`
- `externalTimeBatch`
- `timeLength`
- `uniqueExternalTimeBatch`

#### time

| Syntax       | `<event> time(<int|long|time> windowTime)` |
|--------------|-------------------------------------------|
| Extension    | Window                                    |
| Description  | A sliding time window that holds events that arrived during the last `windowTime` period at a given time, and gets updated for each event arrival and expiry. |
| Parameter    | - `windowTime`: The sliding time period for which the window should hold events. |
| Return Type  | Returns current and expired events. |
| Examples     | - `time(20)` for processing events that arrived within the last 20 milliseconds. |
|              | - `time(2 min)` for processing events that arrived within the last 2 minutes. |
### timeBatch

**Syntax**

```
<event> timeBatch(<int|long|time> windowTime, <int> startTime)
```

**Extension Type**

Window

**Description**

A batch (tumbling) time window that holds events that arrive during `windowTime` periods, and gets updated for each `windowTime`.

**Parameter**

- `windowTime`: The batch time period for which the window should hold events.
- `startTime` (Optional): This specifies an offset in milliseconds in order to start the window at a time different to the standard time.

**Return Type**

Returns current and expired events.

**Examples**

- `timeBatch(20)` processes events that arrive every 20 milliseconds.
- `timeBatch(2 min)` processes events that arrive every 2 minutes.
- `timeBatch(10 min, 0)` processes events that arrive every 10 minutes starting from the 0th minute. e.g., If you deploy your window at 08:22 and the first event arrives at 08:26, this event occurs within the time window 08.20 - 08.30. Therefore, this event is emitted at 08.30.
- `timeBatch(10 min, 1000*60*5)` processes events that arrive every 10 minutes starting from 5th minute. e.g., If you deploy your window at 08:22 and the first event arrives at 08:26, this event occurs within the time window 08.25 - 08.35. Therefore, this event is emitted at 08.35.

### length

**Syntax**

```
<event> length(<int> windowLength)
```

**Extension Type**

Window

**Description**

A sliding length window that holds the last `windowLength` events at a given time, and gets updated for each arrival and expiry.

**Parameter**

- `windowLength`: The number of events that should be included in a sliding length window.

**Return Type**

Returns current and expired events.

**Examples**

- `length(10)` for processing the last 10 events.
- `length(200)` for processing the last 200 events.

### lengthBatch

**Syntax**

```
<event> lengthBatch(<int> windowLength)
```

**Extension Type**

Window
**Description**
A batch (tumbling) length window that holds a number of events specified as the `windowLength`. The window is updated each time a batch of events that equals the number specified as the `windowLength` arrives.

**Parameter**
`windowLength`: The number of events the window should tumble.

**Return Type**
Returns current and expired events.

**Examples**
- `lengthBatch(10)` for processing 10 events as a batch.
- `lengthBatch(200)` for processing 200 events as a batch.

---

### externalTime

**Syntax**
```java
<event> externalTime(<long> timestamp, <int|long|time> windowTime)
```

**Extension Type**
Window

**Description**
A sliding time window based on external time. It holds events that arrived during the last `windowTime` period from the external timestamp, and gets updated on every monotonically increasing timestamp.

**Parameter**
- `windowTime`: The sliding time period for which the window should hold events.

**Return Type**
Returns current and expired events.

**Examples**
- `externalTime(eventTime, 20)` for processing events arrived within the last 20 milliseconds from the `eventTime`
- `externalTime(eventTimestamp, 2 min)` for processing events arrived within the last 2 minutes from the `eventTimestamp`

---

### cron

**Syntax**
```java
<event> cron(<string> cronExpression)
```

**Extension Type**
Window

**Description**
This window returns events processed periodically as the output in time-repeating patterns, triggered based on time passing.

**Parameter**
`cronExpression`: cron expression that represents a time schedule.

**Return Type**
Returns current and expired events.

**Examples**
- `cron('*/5 * * * * ?')` returns processed events as the output every 5 seconds.

---

firstUnique
## Syntax

\[
<\text{event}> \ \text{firstUnique}(<\text{string}> \ \text{attribute})
\]

### Extension Type
Window

### Description
First unique window processor keeps only the first events that are unique according to the given unique attribute.

### Parameter
- **attribute**: The attribute that should be checked for uniqueness.

### Return Type
Returns current and expired events.

### Examples
\[
\text{firstUnique(ip)} \ \text{returns the first event arriving for each unique ip.}
\]

## unique

### Syntax

\[
<\text{event}> \ \text{unique}(<\text{string}> \ \text{attribute})
\]

### Extension Type
Window

### Description
This window keeps only the latest events that are unique according to the given unique attribute.

### Parameter
- **attribute**: The attribute that should be checked for uniqueness.

### Return Type
Returns current and expired events.

### Examples
\[
\text{unique(ip)} \ \text{returns the latest event that arrives for each unique ip.}
\]

## sort

### Syntax

\[
<\text{event}> \ \text{sort}(<\text{int}> \ \text{windowLength})
\]

\[
<\text{event}> \ \text{sort}(<\text{int}> \ \text{windowLength}, <\text{string}> \ \text{attribute}, <\text{string}> \ \text{order})
\]

\[
<\text{event}> \ \text{sort}(<\text{int}> \ \text{windowLength}, <\text{string}> \ \text{attribute}, <\text{string}> \ \text{order}, \ldots, <\text{string}> \ \text{attributeN}, <\text{string}> \ \text{orderN})
\]

### Extension Type
Window

### Description
This window holds a batch of events that equal the number specified as the windowLength and sorts them in the given order.

### Parameter
- **attribute**: The attribute that should be checked for the order.

### Return Type
Returns current and expired events.

### Examples
\[
\text{sort(5, price, 'asc')} \ \text{keeps the events sorted by price in the ascending order. Therefore, at any given time, the window contains the 5 lowest prices.}
\]
### frequent

**Syntax**

\[
\text{<event> frequent(<int> eventCount)}
\]

\[
\text{<event> frequent(<int> eventCount, <string> attribute, .., <string> attributeN)}
\]

**Extension Type**

Window

**Description**

This window returns the latest events with the most frequently occurred value for a given attribute(s). Frequency calculation for this window processor is based on Misra-Gries counting algorithm.

**Parameter**

- **eventCount**: The number of most frequent events to be emitted to the stream.

**Return Type**

Returns current and expired events.

**Examples**

- `frequent(2)` returns the 2 most frequent events.
- `frequent(2, cardNo)` returns the 2 latest events with the most frequently appeared card numbers.

### lossyFrequent

**Syntax**

\[
\text{<event> lossyFrequent(<double> supportThreshold, <double> errorBound)}
\]

\[
\text{<event> lossyFrequent(<double> supportThreshold, <double> errorBound, <string> attribute, .., <string> attributeN)}
\]

**Extension Type**

Window

**Description**

This window identifies and returns all the events of which the current frequency exceeds the value specified for the `supportThreshold` parameter.

**Parameters**

- **errorBound**: The error bound value.
- **attribute**: The attributes to group the events. If no attributes are given, the concatenation of all the attributes of the event is considered.

**Return Type**

Returns current and expired events.
### Examples

- `lossyFrequent(0.1, 0.01)` returns all the events of which the current frequency exceeds 0.1, with an error bound of 0.01.
- `lossyFrequent(0.3, 0.05, cardNo)` returns all the events of which the `cardNo` attribute frequency exceeds 0.3, with an error bound of 0.05.

### externalTimeBatch

#### Syntax

```plaintext
<event> externalTimeBatch(<long> timestamp, <int|long|time> windowTime, <int|long|time> startTime, <int|long|time> timeout)
```

#### Extension Type

Window

#### Description

A batch (tumbling) time window based on external time, that holds events arrived during `windowTime` periods, and gets updated for every `windowTime`.

#### Parameters

- **timestamp**: The time which the window determines as current time and will act upon. The value of this parameter should be monotonically increasing.
- **windowTime**: The batch time period for which the window should hold events.
- **startTime** (Optional): User defined start time. This could either be a constant (of type int, long or time) or an attribute of the corresponding stream (of type long). If an attribute is provided, initial value of attribute would be considered as startTime. When startTime is not given, initial value of timestamp is used as the default.
- **timeout** (Optional): Time to wait for arrival of new event, before flushing and giving output for events belonging to a specific batch. If timeout is not provided, system waits till an event from next batch arrives to flush current batch.

#### Return Type

Returns current and expired events.
Examples

- `externalTimeBatch(eventTime, 20)` for processing events that arrive every 20 milliseconds from the `eventTime`.
- `externalTimeBatch(eventTimestamp, 2 min)` for processing events that arrive every 2 minutes from the `eventTimestamp`.
- `externalTimeBatch(eventTimestamp, 2 sec, 0)` for processing events that arrive every 2 seconds from the `eventTimestamp`. Starts on 0th millisecond of an hour.
- `externalTimeBatch(eventTimestamp, 2 sec, eventTimestamp, 100)` for processing events that arrive every 2 seconds from the `eventTimestamp`. Considers the first event's `eventTimestamp` value as startTime. Waits 100 milliseconds for the arrival of a new event before flushing current batch.

**timeLength**

| Syntax                                      | `<event> timeLength(<int|long|time> windowTime, <int> windowLength)` |
|---------------------------------------------|---------------------------------------------------------------------|
| Extension Type                             | Window                                                              |
| Description                                | A sliding time window that, at a given time holds the last `windowLength` events that arrived during last `windowTime` period, and gets updated for every event arrival and expiry. |
| Parameters                                 | windowTime: The sliding time period for which the window should hold events. |
|                                              | windowLength: The number of events that should be included in a sliding length window. |
| Return Type                                | Returns current and expired events.                                 |
| Examples                                   | timeLength(20 sec, 10) for processing the last 10 events that arrived within the last 20 seconds. |
|                                              | timeLength(2 min, 5) for processing the last 5 events that arrived within the last 2 minutes. |

**uniqueExternalTimeBatch**

| Syntax                                      | `<event> uniqueExternalTimeBatch(<string> attribute, <long> timestamp, <int|long|time> windowTime, <int|long|time> startTime, <int|long|time> timeout, <bool> replaceTimestampWithBatchEndTime)` |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| Extension Type                             | Window                                                             |

**Description**  
A batch (tumbling) time window based on external time that holds latest unique events that arrive during the windowTime periods, and gets updated for each window Time.

**Parameters**

- **attribute**: The attribute that should be checked for uniqueness.
- **timestamp**: The time considered as the current time for the window to act upon. The value of this parameter should be monotonically increasing.
- **windowTime**: The batch time period for which the window should hold events.
- **startTime** (Optional): The user-defined start time. This can either be a constant (of the int, long or time type) or an attribute of the corresponding stream (of the long type). If an attribute is provided, the initial value of this attribute is considered as the startTime. When the startTime is not given, the initial value of the timestamp is used by default.
- **timeout** (Optional): The time duration to wait for the arrival of new event before flushing and giving output for events belonging to a specific batch. If the timeout is not provided, the system waits until an event from next batch arrives to flush the current batch.
- **replaceTimestampWithBatchEndTime** (Optional): Replaces the timestamp value (That is pointed by the 2nd parameter) with the corresponding batch end time stamp.

**Return Type**

Returns current and expired events.
### Examples

- `uniqueExternalTimeBatch(ip, eventTime, 20)` processes unique events based on the `ip` attribute that arrive every 20 milliseconds from the `eventTime`.
- `uniqueExternalTimeBatch(ip, eventTimestamp, 2 min)` processes unique events based on the `ip` attribute that arrive every 2 minutes from the `eventTimestamp`.
- `uniqueExternalTimeBatch(ip, eventTimestamp, 2 sec, 0)` processes unique events based on the `ip` attribute that arrive every 2 seconds from the `eventTimestamp`. It starts on 0th millisecond of an hour.
- `uniqueExternalTimeBatch(ip, eventTimestamp, 2 sec, eventTimestamp, 100)` processes unique events based on the `ip` attribute that arrive every 2 seconds from the `eventTimestamp`. It considers the `eventTimestamp` value of the first event as `startTime`, and waits 100 milliseconds for the arrival of a new event before flushing current batch.
- `uniqueExternalTimeBatch(ip, eventTimestamp, 2 sec, eventTimestamp, 100, true)` processes unique events based on the `ip` attribute that arrive every 2 seconds from the `eventTimestamp`. It considers the `eventTimestamp` value of the first event as `startTime`, and waits 100 milliseconds for the arrival of a new event before flushing current batch. Here, the value for `eventTimestamp` is replaced with the timestamp of the batch end.

### Inbuilt Aggregate Functions

Aggregate functions are used to perform operations such as sum on aggregated set of events through a window. Usage of an aggregate function is as follows.

```siddhi
@info(name = 'query1')
from cseEventStream#window.timeBatch(1 sec)
select symbol, sum(price) as price
insert into resultStream;
```

Following are the supported inbuilt aggregate functions of Siddhi.

- **Sum**
- **Average**
- **Maximum**
- **Minimum**
- **Count**
- **Standard Deviation**
- **Distinct Count**
- **Forever Maximum**
**Forever Minimum**

### Sum

| Syntax       | \(<\text{long}|\text{double}\>\ \text{sum}(\text{<int}|\text{long}|\text{double}|\text{float}\ > \ arg) |
|--------------|-------------------------------------------------------------------------------------------------|
| Extension Type | Aggregate Function                                                                          |
| Description  | Calculates the sum for all the events.                                                        |
| Parameter    | The value that needs to be summed.                                                            |
| Return Type  | Returns long if the input parameter type is int or long, and returns double if the input parameter type is float or double. |
| Examples     | • \(\text{sum}(20)\) returns the sum of 20s as a long value for each event arrival and expiry. |
|              | • \(\text{sum}(\text{temp})\) returns the sum of all temp attributes based on each event arrival and expiry. |

### Average

| Syntax       | \(<\text{double}\>\ \text{avg}(\text{<int}|\text{long}|\text{double}|\text{float}\ > \ arg) |
|--------------|-------------------------------------------------------------------------------------------------|
| Extension Type | Aggregate Function                                                                          |
| Description  | Calculates the average for all the events.                                                      |
| Parameter    | \(\text{arg}\): The value that need to be averaged.                                            |
| Return Type  | Returns the calculated average value as a double.                                               |
| Example      | \(\text{avg}(\text{temp})\) returns the average temp value for all the events based on their arrival and expiry. |

### Maximum

| Syntax       | \(<\text{int}|\text{long}|\text{double}|\text{float}\>\ \text{max}(\text{<int}|\text{long}|\text{double}|\text{float}\ > \ arg) |
|--------------|-------------------------------------------------------------------------------------------------|
| Extension Type | Aggregate Function                                                                          |
| Description  | Returns the maximum value for all the events.                                                  |
| Parameter    | \(\text{arg}\): The value that needs to be compared to find the maximum value.               |
| Return Type  | Returns the maximum value in the same data type as the input.                                 |
| Example      | \(\text{max}(\text{temp})\) returns the maximum temp value recorded for all the events based on their arrival and expiry. |

### Minimum

| Syntax       | \(<\text{int}|\text{long}|\text{double}|\text{float}\>\ \text{min}(\text{<int}|\text{long}|\text{double}|\text{float}\ > \ arg) |
|--------------|-------------------------------------------------------------------------------------------------|
| Extension Type | Aggregate Function                                                                          |
| Description  |                                                                                               |
| Parameter    |                                                                                               |
| Return Type  |                                                                                               |
| Example      |                                                                                               |
**Description**  
Returns the minimum value for all the events.

**Parameter**  
**arg**: The value that needs to be compared to find the minimum value.

**Return Type**  
Returns the minimum value in the same type as the input.

**Example**  
`min(temp)` returns the minimum temp value recorded for all the events based on their arrival and expiry.

---

**Count**

**Syntax**  
`<long> count()`

**Extension Type**  
Aggregate Function

**Description**  
Returns the count of all the events.

**Return Type**  
Returns the event count as a long.

**Example**  
`count()` returns the count of all the events.

---

**Standard Deviation**

**Syntax**  
`<double> stddev(<int|long|double|float> arg)`

**Extension Type**  
Aggregate Function

**Description**  
Returns the calculated standard deviation for all the events.

**Parameter**  
**arg**: The value that should be used to calculate the standard deviation.

**Return Type**  
Returns the calculated standard deviation value as a double.

**Example**  
`stddev(temp)` returns the calculated standard deviation of temp for all the events based on their arrival and expiry.

---

**Distinct Count**

**Syntax**  
`<long> distinctcount(<int|long|double|float|string> arg)`

**Extension Type**  
Aggregate Function

**Description**  
Returns the count of distinct occurrences for a given arg.

**Parameter**  
**arg**: The value that should be counted.

**Return Type**  
Returns the count of distinct occurrences for a given arg.
## Example

```
distinctcount(pageID) for the following output returns 3.

"WEB_PAGE_1"
"WEB_PAGE_1"
"WEB_PAGE_2"
"WEB_PAGE_3"
"WEB_PAGE_1"

"WEB_PAGE_2"
```

## Forever Maximum

| Syntax                     | `<int|long|double|float> maxForever (<int|long|double|float> arg)` |
|----------------------------|------------------------------------------------------------|
| Extension Type             | Aggregate Function                                         |
| Description                | This is the attribute aggregator to store the maximum value for a given attribute throughout the lifetime of the query regardless of any windows in-front. |
| Parameter                  | **arg**: The value that needs to be compared to find the maximum value. |
| Return Type                | Returns the maximum value in the same data type as the input. |
| Example                    | `maxForever(temp)` returns the maximum temp value recorded for all the events throughout the lifetime of the query. |

## Forever Minimum

| Syntax                     | `<int|long|double|float> minForever (<int|long|double|float> arg)` |
|----------------------------|------------------------------------------------------------|
| Extension Type             | Aggregate Function                                         |
| Description                | This is the attribute aggregator to store the minimum value for a given attribute throughout the lifetime of the query regardless of any windows in-front. |
| Parameter                  | **arg**: The value that needs to be compared to find the minimum value. |
| Return Type                | Returns the minimum value in the same data type as the input. |
| Example                    | `minForever(temp)` returns the minimum temp value recorded for all the events throughout the lifetime of the query. |

### Writing Extensions to Siddhi

Custom extensions can be written in Siddhi in order to cater to usecase specific logic that is not available out of the box in Siddhi.

For a general idea of how to write an extension, see [SiddhiQL Guide - Writing Custom Extensions](#).

The following sections contain detailed information on how to create different types of Siddhi extensions.

- Writing a Custom Aggregate Function
- Writing a Custom Window Extension
Writing a Custom Aggregate Function

The Siddhi Aggregate Function consumes zero or more parameters for each event and outputs a single attribute with an aggregated result based on the input parameters as an output. This could be used in conjunction with a window in order to find the aggregated results based on a given window such as the Aggregate Function operator.

To implement a custom aggregate function, follow the procedure below.

1. Create a class extending the org.wso2.siddhi.core.query.selector.attribute.aggregator.AttributeAggregator.
2. Create an appropriate .siddhiext extension mapping file.
3. Compile the class.
4. Build the jar containing the .class and .siddhiext files.
5. Add the jar to the Siddhi class path. If you need to run the extension on WSO2 CEP, add it to the <CEP_HOME>/repository/components/dropins directory.

For example, an aggregate function extension with custom as the namespace and std as the function name can be referred in a query as follows.

```
from pizzaOrder#window.length(20)
select custom:count(orderNo) as totalOrders
insert into orderCount;
```

Sample implementation of a custom aggregate function extension can be found below;

```java
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 * WSO2 Inc. licenses this file to you under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except
 * in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
package org.wso2.siddhi.extension.customAggregateFunction;

import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.core.query.selector.attribute.aggregator.AttributeAggregat
import org.wso2.siddhi.query.api.definition.Attribute;

/**
 * Custom Count Extension which returns event count as a long
 */
public class CountAggregateFunction extends AttributeAggregator {
    private static Attribute.Type type = Attribute.Type.LONG;
    private long value = 0l;

    /**
     * The initialization method for CountAggregateFunction
     *
     * @param attributeExpressionExecutors are the executors of each
     * attributes in the function
     * @param executionPlanContext Execution plan runtime context
     */
    @Override
    protected void init(ExpressionExecutor[] attributeExpressionExecutors,
                        ExecutionPlanContext executionPlanContext) {
        //Implement class specific initialization
    }

    /**
     * The process add method of the CountAggregateFunction, used when zero
     * or one function parameter is provided
     *
     * @param data null if the function parameter count is zero or runtime
     * data value of the function parameter
     * @return the count value
     */
    @Override
    public Object processAdd(Object data) {
        value++;
        return value;
    }

    /**
     * The process add method of the CountAggregateFunction, used when more
     * than one function parameters are provided
     *
     * @param data the data values for the function parameters
     * @return the count value
     */
    @Override
    public Object processAdd(Object[] data) {
        value++;
        return value;
    }

    /**
     * The process remove method of the CountAggregateFunction, used when
     * zero or one function parameter is provided
     *
/**
 * @param data null if the function parameter count is zero or runtime data value of the function parameter
 * @return the count value
 */
@Override
public Object processRemove(Object data) {
    value--;
    return value;
}

/**
 * The process remove method of the CountAggregateFunction, used when more than one function parameters are provided
 * @param data the data values for the function parameters
 * @return the count value
 */
@Override
public Object processRemove(Object[] data) {
    value--;
    return value;
}

/**
 * Reset count value
 * @return reset value
 */
@Override
public Object reset() {
    value = 0L;
    return value;
}

/**
 * This will be called only once and this can be used to acquire required resources for the processing element.
 * This will be called after initializing the system and before starting to process the events.
 */
@Override
public void start() {
    //Implement start logic to acquire relevant resources
}

/**
 * This will be called only once and this can be used to release the acquired resources for processing.
 * This will be called before shutting down the system.
 */
@Override
public void stop() {

//Implement stop logic to release the acquired resources
}

/**
 * Used to collect the serializable state of the processing element, that need to be
 * persisted for the reconstructing the element to the same state on a different point of time
 * @return stateful objects of the processing element as an array */
@override
public Object[] currentState() {
    return new Object[]{value};
}

/**
 * Used to restore serialized state of the processing element, for reconstructing
 * the element to the same state as if was on a previous point of time.
 * @param state the stateful objects of the element as an array on the same order provided by currentState().
 */
@override
public void restoreState(Object[] state) {
    value = (Long) state[0];
}

public Attribute.Type getReturnType() {
    return type;
}
Sample custom.siddhiext extension mapping file for the custom aggregate function extension can be found below:

```java
std=org.wso2.siddhi.core.query.selector.attribute.aggregator.StrandedDeviationAggregateFunction
```

Writing a Custom Window Extension

The Siddhi Window Extension allows events to be collected and expired without altering the event format based on the given input parameters such as the Window operator.

To write a custom window, follow the procedure below.

1. Create a class extending `org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor`.
2. Create an appropriate .siddhiext extension mapping file.
3. Compile the class.
4. Build the jar containing the .class and the .siddhiext files.
5. Add the jar to the Siddhi class path. If you need to run the extension on WSO2 CEP, add the jar to the `<CEP_HOME>/repository/components/dropins` directory.

For example, a window extension created with `custom` as the namespace and `customWindow` as the function name can be referred in a query as shown below.

```java
from TempStream#window.custom:customWindow(10)
select *
insert into AvgRoomTempStream ;
```

For the window extension to be used in a `join` query, it should be possible to find the window extension. To enable this, the `org.wso2.siddhi.core.query.processor.stream.window.FindableProcessor` interface should be implemented.
The following is a sample implementation of a custom window extension.

```java
package org.wso2.siddhi.extension.customWindow;

import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.event.ComplexEvent;
import org.wso2.siddhi.core.event.ComplexEventChunk;
import org.wso2.siddhi.core.event.MetaComplexEvent;
import org.wso2.siddhi.core.event.stream.StreamEvent;
import org.wso2.siddhi.core.event.stream.StreamEventCloner;
import org.wso2.siddhi.core.executor.ConstantExpressionExecutor;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.core.executor.VariableExpressionExecutor;
import org.wso2.siddhi.core.query.processor.Processor;
import org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor;
import org.wso2.siddhi.core.query.processor.stream.window.FindableProcessor;
import org.wso2.siddhi.core.table.EventTable;
import org.wso2.siddhi.core.util.collection.operator.Finder;
import org.wso2.siddhi.core.util.parser.CollectionOperatorParser;
import org.wso2.siddhi.query.api.exception.ExecutionPlanValidationException;
import org.wso2.siddhi.query.api.expression.Expression;
import java.util.List;
import java.util.Map;

/**
 * Custom Sliding Length Window implementation which holds last length events, and gets updated on every event arrival and expiry.
 */
public class CustomWindow extends WindowProcessor implements FindableProcessor {
    private int length;
```
private int count = 0;
private ComplexEventChunk<StreamEvent> expiredEventChunk;

/**
 * The init method of the WindowProcessor, this method will be called before other methods
 * @param attributeExpressionExecutors the executors of each function parameters
 * @param executionPlanContext the context of the execution plan
 */
@override
protected void init(ExpressionExecutor[] attributeExpressionExecutors,
ExecutionPlanContext executionPlanContext) {
    expiredEventChunk = new ComplexEventChunk<StreamEvent>();
    if (attributeExpressionExecutors.length == 1) {
        length = (Integer) ((ConstantExpressionExecutor)
attributeExpressionExecutors[0]).getValue();
    } else {
        throw new ExecutionPlanValidationException("Length window should only have one parameter (<int> windowLength), but found " +
attributeExpressionExecutors.length + " input attributes");
    }
}

/**
 * The main processing method that will be called upon event arrival
 * @param streamEventChunk the stream event chunk that need to be processed
 * @param nextProcessor the next processor to which the success events need to be passed
 * @param streamEventCloner helps to clone the incoming event for local storage or modification
 */
@override
protected synchronized void process(ComplexEventChunk<StreamEvent> streamEventChunk,
Processor nextProcessor, StreamEventCloner streamEventCloner) {
    while (streamEventChunk.hasNext()) {
        StreamEvent streamEvent = streamEventChunk.next();
        StreamEvent clonedEvent =
streamEventCloner.copyStreamEvent(streamEvent);
        clonedEvent.setType(StreamEvent.Type.EXPIRED);
        if (count < length) {
            count++;
            this.expiredEventChunk.add(clonedEvent);
        } else {
            StreamEvent firstEvent = this.expiredEventChunk.poll();
            if (firstEvent != null) {
                streamEventChunk.insertBeforeCurrent(firstEvent);
                this.expiredEventChunk.add(clonedEvent);
            }
        }
    }
}
} else {
    streamEventChunk.insertBeforeCurrent(clonedEvent);
}
}
nextProcessor.process(streamEventChunk);

/**
 * To find events from the processor event pool, that the matches the
 * matchingEvent based on finder logic.
 * @param matchingEvent the event to be matched with the events at the
 * processor
 * @param finder the execution element responsible for finding
 * the corresponding events that matches
 * the matchingEvent based on pool of events at Processor
 * @return the matched events
 */
@Override
public synchronized StreamEvent find(ComplexEvent matchingEvent, Finder finder) {
    return finder.find(matchingEvent, expiredEventChunk, streamEventCloner);
}

/**
 * To construct a finder having the capability of finding events at the
 * processor that corresponds to the incoming
 * matchingEvent and the given matching expression logic.
 * @param expression the matching expression
 * @param metaComplexEvent the meta structure of the
 * incoming matchingEvent
 * @param executionPlanContext current execution plan context
 * @param variableExpressionExecutors the list of variable
 * ExpressionExecutors already created
 * @param eventTableMap map of event tables
 * @param matchingStreamIndex the stream index of the incoming
 * matchingEvent
 * @param withinTime the maximum time gap between the
 * events to be matched
 * @return finder having the capability of finding events at the
 * processor against the expression and incoming
 * matchingEvent
 */
@Override
public Finder constructFinder(Expression expression, MetaComplexEvent metaComplexEvent, ExecutionPlanContext executionPlanContext,
    List<VariableExpressionExecutor> variableExpressionExecutors, Map<String, EventTable> eventTableMap, int matchingStreamIndex, long withinTime) {
    return CollectionOperatorParser.parse(expression, metaComplexEvent,
executionPlanContext, variableExpressionExecutors, eventTableMap, matchingStreamIndex, inputDefinition, withinTime);
}

/**
* This will be called only once and this can be used to acquire
* required resources for the processing element.
* This will be called after initializing the system and before
* starting to process the events.
*/
@Override
public void start() {
    //Implement start logic to acquire relevant resources
}

/**
* This will be called only once and this can be used to release
* the acquired resources for processing.
* This will be called before shutting down the system.
*/
@Override
public void stop() {
    //Implement stop logic to release the acquired resources
}

/**
* Used to collect the serializable state of the processing element,
* that need to be persisted for the reconstructing the element to the same state on a
* different point of time
* *
* @return stateful objects of the processing element as an array
*/
@Override
public Object[] currentState() {
    return new Object[]{expiredEventChunk, count};
}

/**
* Used to restore serialized state of the processing element, for
* reconstructing
* the element to the same state as if was on a previous point of time.
* *
* @param state the stateful objects of the element as an array on
* the same order provided by currentState().
*/
@Override
public void restoreState(Object[] state) {
    expiredEventChunk = (ComplexEventChunk<StreamEvent>) state[0];
count = (Integer) state[1];
}
}

Sample custom.siddhiext extension mapping file for the custom window extension can be found below;

# Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
#
# WSO2 Inc. licenses this file to you under the Apache License,
# Version 2.0 (the "License"); you may not use this file except
# in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#

customWindow=org.wso2.siddhi.extension.customWindow.CustomWindow

Writing a Custom Function Extension

The Siddhi Function Extension consumes zero or more parameters for each event and outputs a single attribute. This could be used to manipulate event attributes to generate new attributes such as the Function operator.

To implement a custom function extension, follow the procedure below.

1. Create a class extending org.wso2.siddhi.core.executor.function.FunctionExecutor.
2. Create an appropriate .siddhiext extension mapping file.
3. Compile the class.
4. Build the jar containing the .class and the .siddhiext files.
5. Add the jar to the Siddhi class path. If you need to run the extension on WSO2 CEP, add it to the <CEP_HOME>/repository/components/dropins directory.

For example, a custom function extension created with math as the namespace and sin as the function name can be referred in a query as shown below.

```sql
from InValueStream
select math:sin(inValue) as sinValue
insert into OutMediationStream;
```
The following is a sample implementation of a custom function extension.

```java
package org.wso2.siddhi.extension.math;

import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.exception.ExecutionPlanRuntimeException;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.core.executor.function.FunctionExecutor;
import org.wso2.siddhi.query.api.exception.ExecutionPlanValidationException;

public class SinFunctionExtension extends FunctionExecutor {

  /**
   * sin(a);
   * Returns the sine of a (a is in radians).
   * Accept Type(s) :DOUBLE/INT/FLOAT/LONG
   * Return Type(s): DOUBLE
   */
  public void init(ExpressionExecutor[] attributeExpressionExecutors, ExecutionPlanContext executionPlanContext) {
    // initialization code
  }

  public Object execute(Object[] attributes) {
    // execution logic
    return null;
  }
}
```

**Note**

From CEP 3.0.0 onwards, FunctionExecutor is supposed to be used for writing both custom expressions and conditions.
@Override
protected void init(ExpressionExecutor[] attributeExpressionExecutors, ExecutionPlanContext executionPlanContext) {
    if (attributeExpressionExecutors.length != 1) {
        throw new ExecutionPlanValidationException("Invalid no of arguments passed to math:sin() function, " + 
            "required 1, but found " + attributeExpressionExecutors.length);
    }
    Attribute.Type attributeType = attributeExpressionExecutors[0].getReturnType();
    if (!(attributeType == Attribute.Type.DOUBLE) || (attributeType == Attribute.Type.INT) || 
        (attributeType == Attribute.Type.FLOAT) || (attributeType == Attribute.Type.LONG)) {
        throw new ExecutionPlanValidationException("Invalid parameter type found for the argument of math:sin() function, " + 
            "required " + Attribute.Type.INT + " or " + Attribute.Type.LONG + 
            " or " + Attribute.Type.FLOAT + " or " + Attribute.Type.DOUBLE + 
            ", but found " + attributeType.toString());
    }
}

/**
 * The main execution method which will be called upon event arrival
 * when there are more than one function parameter
 * @param data the runtime values of function parameters
 * @return the function result
 */
@Override
protected Object execute(Object[] data) {
    return null;
}

/**
 * The main execution method which will be called upon event arrival
 * when there are zero or one function parameter
 * @param data null if the function parameter count is zero or runtime data value of the function parameter
 * @return the function result
 */
@Override
protected Object execute(Object data) {
    if (data != null) {
        //type-conversion
        if (data instanceof Integer) {
            int inputInt = (Integer) data;
            return Math.sin((double) inputInt);
```java
} else if (data instanceof Long) {
    long inputLong = (Long) data;
    return Math.sin((double) inputLong);
} else if (data instanceof Float) {
    float inputFloat = (Float) data;
    return Math.sin((double) inputFloat);
} else if (data instanceof Double) {
    return Math.sin((Double) data);
} else {
    throw new ExecutionPlanRuntimeException("Input to the
    math:sin() function cannot be null");
}
return null;

/**
 * This will be called only once and this can be used to acquire
 * required resources for the processing element.
 * This will be called after initializing the system and before
 * starting to process the events.
 */
@Override
public void start() {
    //Implement start logic to acquire relevant resources
}

/**
 * This will be called only once and this can be used to release
 * the acquired resources for processing.
 * This will be called before shutting down the system.
 */
@Override
public void stop() {
    //Implement stop logic to release the acquired resources
}

@Override
public Attribute.Type getReturnType() {
    return Attribute.Type.DOUBLE;
}

/**
 * Used to collect the serializable state of the processing element,
 * that need to be
 * persisted for the reconstructing the element to the same state on a
 * different point of time
 */
@Override
public Object[] currentState() {
```
return null;
}

/**
 * Used to restore serialized state of the processing element, for reconstructing
 * the element to the same state as if was on a previous point of time.
 * @param state the stateful objects of the element as an array on
 * the same order provided by currentState().
 */
@Override
public void restoreState(Object[] state) {
//Implement restore state logic.

Sample math.siddhiext extension mapping file for the custom function extension can be found below;

```
# Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
# WSO2 Inc. licenses this file to you under the Apache License,
# Version 2.0 (the "License"); you may not use this file except
# in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#
sin=org.wso2.siddhi.extension.math.SinFunctionExtension
```

Writing a Custom Stream Function Extension

The Stream Function Extension allows events to be modified by adding one or more attributes to it. Events can be output upon each event arrival.

To implement a custom stream function, follow the procedure below.

2. Create an appropriate `.siddhiext` extension mapping file.
3. Compile the class.
4. Build the jar containing the `.class` and the `.siddhiext` files.
5. Add the jar to the Siddhi class path. If you need to run the extension on WSO2 CEP, add the jar to the `<CEP_HOME>/repository/components/drops directory`.

For example, a Stream Function extension created with `geo` as the namespace and `geocode` as the function name can be referred in a query as shown below.

```
from geocodeStream#geo:geocode(location)
select latitude, longitude, formattedAddress
insert into dataOut;
```

The following is a sample implementation of a custom stream function extension.
package org.wso2.siddhi.extension.geo;

import com.google.code.geocoder.Geocoder;
import com.google.code.geocoder.GeocoderRequestBuilder;
import com.google.code.geocoder.model.GeocodeResponse;
import com.google.code.geocoder.model.GeocoderRequest;
import org.apache.log4j.Logger;
import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.exception.ExecutionPlanCreationException;
import org.wso2.siddhi.core.exception.ExecutionPlanRuntimeException;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.core.query.processor.stream.function.StreamFunctionProcess
or;
import org.wso2.siddhi.query.api.definition.AbstractDefinition;
import org.wso2.siddhi.query.api.definition.Attribute;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * This extension transforms a location into its geo-coordinates and
 * formatted
 * address
 */
public class GeocodeStreamFunctionProcessor extends StreamFunctionProcessor 
{
    private static final Logger LOGGER =
    Logger.getLogger(GeocodeStreamFunctionProcessor.class);
    private final Geocoder geocoder = new Geoder();
    private boolean debugModeOn;

    /**
     * This extension transforms a location into its geo-coordinates and
     * formatted
     * address
     */
    public class GeocodeStreamFunctionProcessor extends StreamFunctionProcessor 
{
* The process method of the GeocodeStreamFunctionProcessor, used when more than one function parameters are provided
* @param data the data values for the function parameters
* @return the data for additional output attributes introduced by the function
*
@Override
protected Object[] process(Object[] data) {
    return process(data[0]);
}

/**
 * The process method of the GeocodeStreamFunctionProcessor, used when zero or one function parameter is provided
 * @param data null if the function parameter count is zero or runtime data value of the function parameter
 * @return the data for additional output attribute introduced by the function
 */
@Override
protected Object[] process(Object data) {
    String location = data.toString();
    // Make the geocode request to API library
    GeocoderRequest geocoderRequest = new GeocoderRequestBuilder()
        .setAddress(location)
        .setLanguage("en")
        .getGeocoderRequest();
    double latitude, longitude;
    String formattedAddress;
    try {
        GeocodeResponse geocoderResponse =
            geocoder.geocode(geocoderRequest);
        if (!geocoderResponse.getResults().isEmpty()) {
            latitude =
                geocoderResponse.getResults().get(0).getGeometry().getLocation()
                    .getLat().doubleValue();
            longitude =
                geocoderResponse.getResults().get(0).getGeometry().getLocation()
                    .getLng().doubleValue();
            formattedAddress =
                geocoderResponse.getResults().get(0).getFormattedAddress();
        } else {
            latitude = -1.0;
            longitude = -1.0;
            formattedAddress = "N/A";
        }
    } catch (IOException e) {
        }
throw new ExecutionPlanRuntimeException("Error in connection to Google Maps API.", e);
}

if (debugModeOn) {
    LOGGER.debug("Formatted address: " + formattedAddress + ", Location coordinates: (" + 
        latitude + ", " + longitude + ")");
}
return new Object[]{formattedAddress, latitude, longitude};

/**
 * The init method of the GeocodeStreamFunctionProcessor, this method will be called before other methods
 * @param inputDefinition the incoming stream definition
 * @param attributeExpressionExecutors the executors of each function parameters
 * @param executionPlanContext the context of the execution plan
 * @return the additional output attributes introduced by the function
 */
@Override
protected List<Attribute> init(AbstractDefinition inputDefinition, ExpressionExecutor[] attributeExpressionExecutors, ExecutionPlanContext executionPlanContext) {
    debugModeOn = LOGGER.isDebugEnabled();
    if (attributeExpressionExecutors[0].getReturnType() != Attribute.Type.STRING) {
        throw new ExecutionPlanCreationException("First parameter should be of type string");
    }
    ArrayList<Attribute> attributes = new ArrayList<Attribute>(6);
    attributes.add(new Attribute("formattedAddress", Attribute.Type.STRING));
    attributes.add(new Attribute("latitude", Attribute.Type.DOUBLE));
    attributes.add(new Attribute("longitude", Attribute.Type.DOUBLE));
    return attributes;
}

/**
 * This will be called only once and this can be used to acquire required resources for the processing element.
 * This will be called after initializing the system and before starting to process the events.
 */
@Override
public void start() {
    //Implement start logic to acquire relevant resources
}
/**
* This will be called only once and this can be used to release
  * the acquired resources for processing.
  * This will be called before shutting down the system.
  */
@Override
public void stop() {
    //Implement stop logic to release the acquired resources
}

/**
 * Used to collect the serializable state of the processing element,
 * that need to be
 * persisted for the reconstructing the element to the same state on a
different point of time
 *
 * @return stateful objects of the processing element as an array
 */
@Override
public Object[] currentState() {
    return new Object[0];
}

/**
 * Used to restore serialized state of the processing element, for
reconstructing
 * the element to the same state as if was on a previous point of time.
 *
 * @param state the stateful objects of the element as an array on
 * the same order provided by currentState().
 */
@Override
public void restoreState(Object[] state) {

Writing a Custom Stream Processor Extension

The Siddhi Stream Processor Extension allows events to be collected and expired by modifying the event format based on the given input parameters.

To implement a custom stream processor, follow the procedure below.

1. Create a class extending `org.wso2.siddhi.core.query.processor.stream.StreamProcessor`.
2. Create an appropriate `.siddhiext` extension mapping file.
3. Compile the class.
4. Build the jar containing the `.class` and the `.siddhiext` files.
5. Add the jar to the Siddhi class path. If you need to run the extension on WSO2 CEP, add the jar to the `<CEP_HOME>/repository/components/dropins` directory.

For example, a Stream Processor extension created with `timeseries` as the namespace and `regress` as the function name can be referred in a query as shown below.

```siddhi
from baseballData#timeseries:regress(2, 10000, 0.95, salary, rbi, walks, strikeouts, errors)
select *
insert into regResults;
```
The following is a sample implementation of a custom stream processor extension.

```java
package org.wso2.siddhi.extension.timeseries;

import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.event.ComplexEvent;
import org.wso2.siddhi.core.event.ComplexEventChunk;
import org.wso2.siddhi.core.event.stream.StreamEvent;
import org.wso2.siddhi.core.event.stream.populater.ComplexEventPopulater;
import org.wso2.siddhi.core.exception.ExecutionPlanCreationException;
import org.wso2.siddhi.core.executor.ConstantExpressionExecutor;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.core.query.processor.Processor;
import org.wso2.siddhi.core.query.processor.stream.StreamProcessor;
import org.wso2.siddhi.extension.timeseries.linreg.MultipleLinearRegressionCalculator;
import org.wso2.siddhi.extension.timeseries.linreg.RegressionCalculator;
import org.wso2.siddhi.extension.timeseries.linreg.SimpleLinearRegressionCalculator;
import org.wso2.siddhi.query.api.definition.AbstractDefinition;
import org.wso2.siddhi.query.api.definition.Attribute;
import java.util.ArrayList;
import java.util.List;

/**
 * The methods supported by this function are
 * timeseries:regress(int/long/float/double y, int/long/float/double x1,
 * int/long/float/double x2 ...) and
 * timeseries:regress(int calcInterval, int batchSize, double
```
public class LinearRegressionStreamProcessor extends StreamProcessor {

    private int paramCount = 0;                                         // Number of x variables +1
    private int calcInterval = 1;                                       // The frequency of regression calculation
    private int batchSize = 1000000000;                                 // Maximum # of events, used for regression calculation
    private double ci = 0.95;                                           // Confidence Interval

    private final int SIMPLE_LINREG_INPUT_PARAM_COUNT = 2;              // Number of Input parameters in a simple linear regression
    private RegressionCalculator regressionCalculator = null;
    private int paramPosition = 0;

    /**
     * The init method of the LinearRegressionStreamProcessor, this method will be called before other methods
     *
     * @param inputDefinition              the incoming stream definition
     * @param attributeExpressionExecutors the executors of each function parameters
     * @param executionPlanContext         the context of the execution plan
     * @return the additional output attributes introduced by the function
     */
    @Override
    protected List<Attribute> init(AbstractDefinition inputDefinition, ExpressionExecutor[] attributeExpressionExecutors, ExecutionPlanContext executionPlanContext) {
        paramCount = attributeExpressionLength;

        // Capture constant inputs
        if (attributeExpressionExecutors[0] instanceof ConstantExpressionExecutor){
            paramCount = paramCount - 3;
            paramPosition = 3;
            try {
                calcInterval =
                        ((Integer)attributeExpressionExecutors[0].execute(null));
                batchSize =
                        ((Integer)attributeExpressionExecutors[1].execute(null));
            } catch(ClassCastException c) {
                throw new ExecutionPlanCreationException("Calculation interval, batch size and range should be of type int");
            }
            try {
                ci =
        */

    private int paramCount = 0;                                         // Number of x variables +1
    private int calcInterval = 1;                                       // The frequency of regression calculation
    private int batchSize = 1000000000;                                 // Maximum # of events, used for regression calculation
    private double ci = 0.95;                                           // Confidence Interval

    private final int SIMPLE_LINREG_INPUT_PARAM_COUNT = 2;              // Number of Input parameters in a simple linear regression
    private RegressionCalculator regressionCalculator = null;
    private int paramPosition = 0;

    /**
     * The init method of the LinearRegressionStreamProcessor, this method will be called before other methods
     *
     * @param inputDefinition              the incoming stream definition
     * @param attributeExpressionExecutors the executors of each function parameters
     * @param executionPlanContext         the context of the execution plan
     * @return the additional output attributes introduced by the function
     */
    @Override
    protected List<Attribute> init(AbstractDefinition inputDefinition, ExpressionExecutor[] attributeExpressionExecutors, ExecutionPlanContext executionPlanContext) {
        paramCount = attributeExpressionLength;

        // Capture constant inputs
        if (attributeExpressionExecutors[0] instanceof ConstantExpressionExecutor){
            paramCount = paramCount - 3;
            paramPosition = 3;
            try {
                calcInterval =
                        ((Integer)attributeExpressionExecutors[0].execute(null));
                batchSize =
                        ((Integer)attributeExpressionExecutors[1].execute(null));
            } catch(ClassCastException c) {
                throw new ExecutionPlanCreationException("Calculation interval, batch size and range should be of type int");
            }
            try {
                ci =
        */
((Double)attributeExpressionExecutors[2].execute(null));
    } catch(ClassCastException c) {
        throw new ExecutionPlanCreationException("Confidence interval should be of type double and a value between 0 and 1");
    }
}

// Pick the appropriate regression calculator
if (paramCount > SIMPLE_LINREG_INPUT_PARAM_COUNT) {
    regressionCalculator = new MultipleLinearRegressionCalculator(paramCount, calcInterval, batchSize, ci);
} else {
    regressionCalculator = new SimpleLinearRegressionCalculator(paramCount, calcInterval, batchSize, ci);
}

// Add attributes for standard error and all beta values
String betaVal;
ArrayList<Attribute> attributes = new ArrayList<Attribute>(paramCount);
attributes.add(new Attribute("stderr", Attribute.Type.DOUBLE));
for (int itr = 0; itr < paramCount; itr++) {
    betaVal = "beta" + itr;
    attributes.add(new Attribute(betaVal, Attribute.Type.DOUBLE));
}
return attributes;

/**
 * The main processing method that will be called upon event arrival
 * @param streamEventChunk the event chunk that need to be processed
 * @param nextProcessor the next processor to which the success events need to be passed
 * @param streamEventCloner helps to clone the incoming event for local storage or modification
 * @param complexEventPopulater helps to populate the events with the resultant attributes
 */
@Override
protected void process(ComplexEventChunk<StreamEvent> streamEventChunk,
Processor nextProcessor, StreamEventCloner streamEventCloner,
ComplexEventPopulater complexEventPopulater) {
    while (streamEventChunk.hasNext()) {
        ComplexEvent complexEvent = streamEventChunk.next();
        Object[] inputData = new Object[attributeExpressionLength-paramPosition];
        for (int i = paramPosition; i < attributeExpressionLength; i++)

{  
  inputData[i-paramPosition] = attributeExpressionExecutors[i].execute(complexEvent);
}
Object[] outputData = regressionCalculator.calculateLinearRegression(inputData);

// Skip processing if user has specified calculation interval
if (outputData == null) {
  streamEventChunk.remove();
} else {
  complexEventPopulater.populateComplexEvent(complexEvent, outputData);
}
}

nextProcessor.process(streamEventChunk);

/**
 * This will be called only once and this can be used to acquire
 * required resources for the processing element.
 * This will be called after initializing the system and before
 * starting to process the events.
 */
@Override
public void start() {
  //Implement start logic to acquire relevant resources
}

/**
 * This will be called only once and this can be used to release
 * the acquired resources for processing.
 * This will be called before shutting down the system.
 */
@Override
public void stop() {
  //Implement stop logic to release the acquired resources
}

/**
 * Used to collect the serializable state of the processing element,
 * that need to be
 * persisted for the reconstructing the element to the same state on a
 * different point of time
 *
 * @return stateful objects of the processing element as an array
 */
@Override
public Object[] currentState() {
  return new Object[0];
}
/**
 * Used to restore serialized state of the processing element, for
 * reconstructing
 * the element to the same state as if was on a previous point of time.
 * @param state the stateful objects of the element as an array on
 * the same order provided by currentState().
 */
@override
public void restoreState(Object[] state) {
    //Implement restore state logic
Sample timeseries.siddhiext extension mapping file for the custom stream processor extension can be found below:

```plaintext
# Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
# WSO2 Inc. licenses this file to you under the Apache License,
# Version 2.0 (the "License"); you may not use this file except
# in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
#
regress=org.wso2.siddhi.extension.timeseries.LinearRegressionStreamProcess
```

**Siddhi Extensions**

Siddhi language has a wide variety of extensions ranging from string processing to natural language processing. These extensions can be utilized in numerous scenarios to manipulate attributes with ease. When using extensions within Siddhi queries, you need to follow the convention of `extensionName:functionName` to refer to a particular extension. If the extension is of the **Function** type you can refer to it as shown in the following example in different parts of the query.

```plaintext
FROM inputStream[str:contains(description, "Pi-Value")]
SELECT id, math:round(math:pi()) as roundedPiValue
INSERT INTO outputStream;
```

If the extension is of the **Stream Processor** type, you can refer to it as follows. `#reorder:kslack(eventTimestamp)`.

e.g.,
@info(name = 'query1')
FROM inputStream#reorder:kslack(eventTimestamp)
SELECT eventTimestamp, price, volume
INSERT INTO outputStream;

The extensions that are currently available for Siddhi are as follows:

- Time Series Extension
- Math Extension
- String Extension
- Geo Extension
- R Language Extension
- Regular Expression Extension
- Time Extension
- Natural Language Processing Extension
- Map Extension
- Reorder Extension
- Markov Models Extension
- PMML Based Predictive Analytics Extension
- Machine Learning Extension
- Kalman Filter Extension

### Time Series Extension

The Siddhi Time Series Extension enables users to forecast and detect outliers in time series data, using Linear Regression Models.

The following functions are available in the Siddhi Time Series Extension:

- Outlier
- Forecast
- Regression
- kalmanMinMax (kalman based minima maxima detection)
- kernelMinMax (Kernel based minima maxima detection)

#### Forecast

Siddhi allows you to forecast future events using linear regression on real time data streams. The `forecast` function uses a dependent event stream (Y), an independent event stream (X) and a user-specified next X value, and returns the forecast Y value based on the regression equation of the historical data.

The two implementations of the `forecast` function can be distinguished as follows:

- **forecast**: This allows you to specify a batch size (optional) that defines the number of events to be considered for the regression calculation when forecasting the Y value.
- **lengthTimeForecast**: This allows you to restrict the number of events considered for the regression calculation when forecasting the Y value based on a specified time window and/or batch size.

**Input parameters for the forecast function**
The following table describes the input parameters available for the forecast function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculation should be carried out.</td>
<td>Optional</td>
<td>1 (i.e., for every event)</td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events that should be used for a regression calculation.</td>
<td>Optional</td>
<td>1,000,000,000</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Next X Value</td>
<td>The value to be used to forecast the Y value. This can be a constant or an expression (e.g., x+5).</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream</td>
<td>The data stream of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

**Format:** forecast(nextX, Y, X) or forecast(calculation interval, batch size, confidence interval, nextX, Y, X)

**Input parameters for the lengthTimeForecast function**

The following table describes the input parameters available for the lengthTimeForecast function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Window</td>
<td>The maximum time duration that should be considered for a regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events that should be used for a regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Next X Value</td>
<td>The value to be used to forecast the Y value. This can be a constant or an expression (e.g., x+5).</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculation should be carried out.</td>
<td>Optional</td>
<td>1 (i.e., for every event)</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream</td>
<td>The data stream of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

**Format:** lengthTimeForecast(time window, batch size, nextX, Y, X) or lengthTimeForecast(time window, batch size, nextX, calculation interval, confidence interval, Y, X)

**Output parameters**

The following table describes the output parameters.

The same output parameters are available for each implementation.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forecast Y</td>
<td>forecastY</td>
<td>The forecast Y value based on next X and regression equation.</td>
</tr>
<tr>
<td>Standard Error</td>
<td>stdError</td>
<td>The standard error of the regression equation.</td>
</tr>
<tr>
<td>coefficients</td>
<td>beta0, beta1</td>
<td>coefficients of the simple linear regression.</td>
</tr>
<tr>
<td>Input Data</td>
<td>Stream</td>
<td>The name given in the input stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All the items sent in the input stream.</td>
</tr>
</tbody>
</table>

**Examples**

The queries given in the examples below return the following when executed.

- Y value based on the regression equation established using the Y stream and the X stream
- The standard error of the regression equation ()
- coefficients
- All the items available in the input stream

**Example 1**

The following query submits an expression to be used as the next X value (X+2), a dependent input stream (Y), and an independent input stream (X) that are used to perform linear regression between Y and X streams, and compute the forecast Y value based on the next X value specified by you.

```sql
from StockExchangeStream#timeseries:forecast(X+5, Y, X)
select *
insert into StockForecaster
```

**Example 2**

The following query submits a time window (2 seconds), a batch size (100 events), a constant to be used as the next X value (10), a dependent input stream (Y) and an independent input stream (X) that are used to perform linear regression between Y and X streams, and compute the forecast Y value based on the next X value specified by you.

```sql
from StockExchangeStream#timeseries:lengthTimeForecast(2 sec, 100, 10, Y, X)
select *
insert into StockForecaster
```

**Outlier**

Siddhi allows you to identify outliers using linear regression on real-time data streams. The `outlier` function takes a dependent event stream (Y), an independent event stream (X) and a user specified range for outliers, and returns an output to indicate whether the current event is an outlier based on the regression equation that fits
historical data.

The two implementations of `outlier` function can be distinguished as follows.

- **outlier**: This allows you to specify a batch size (optional) that defines the number of events to be considered for the calculation of regression when finding outliers.
- **lengthTimeOutlier**: This allows you to restrict the number of events considered for the regression calculation performed when finding outliers based on a specified time window and/or a batch size.

Input parameters of each implementation are as follows.

**Input parameters for the outlier function**

The following table describes the input parameters available for the `outlier` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculations should be carried out.</td>
<td>Optional</td>
<td>1 (i.e., for every event)</td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events to be used for a regression calculation.</td>
<td>Optional</td>
<td>100,000,000</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Range</td>
<td>The number of standard deviations from the regression equation.</td>
<td>Required</td>
<td>0.95</td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream</td>
<td>The data stream of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

**Format**: `outlier(range, Y, X)` or `outlier(calculation interval, batch size, confidence interval, range, Y, X)`

**Input Parameters for Length Time Outlier Function**

The following table describes the input parameters available for the `lengthTimeOutlier` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Window</td>
<td>The maximum time duration to be considered for a regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events to be used for a regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>The number of standard deviations from the regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculation should be carried out.</td>
<td>Optional</td>
<td>1 (for every event)</td>
</tr>
<tr>
<td>Confidence Level</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream</td>
<td>The data stream of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>
**Format:** 

```
lengthTimeOutlier(time window, batch size, range, Y, X) or lengthTimeOutlier(time window, batch size, range, calculation interval, confidence interval, Y, X)
```

**Output parameters**

The following table describes the output parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlier</td>
<td>outlier</td>
<td>True if the event is an outlier, False if not.</td>
</tr>
<tr>
<td>Standard Error</td>
<td>stdError</td>
<td>The standard error of the regression equation.</td>
</tr>
<tr>
<td>coefficients</td>
<td>beta0, beta1</td>
<td>coefficients of the regression equation.</td>
</tr>
<tr>
<td>Input Stream Data</td>
<td>The name given in the input stream.</td>
<td>All the items sent in the input stream.</td>
</tr>
</tbody>
</table>

**Examples**

In each example given below, the query returns an indication whether the current event is an outlier or not together with the standard error of the regression equation ($\epsilon$), coefficients and all the items available in the input stream.

**Example 1**

The following query submits the number of standard deviations to be used as a range (2), a dependent input stream (Y) and an independent input stream (X) that are used to perform linear regression between Y and X. It returns an output that indicates whether the current event is an outlier or not.

```sql
from StockExchangeStream#timeseries:outlier(2, Y, X)
select *
insert into StockForecaster
```

**Example 2**

The following query submits a time window (2 seconds), a batch size (100 events), the number of standard deviations to be used as a range (2), a dependent input stream (Y) and an independent input stream (X), that are used to perform linear regression between Y and X. It returns an output that indicates whether the current event is an outlier or not.

```sql
from StockExchangeStream#timeseries:lengthTimeOutlier(2 sec, 100, 2, Y, X)
select *
insert into StockForecaster
```

**Regression**

Siddhi enables users to perform linear regression on real time data streams. The **regress function** takes in a dependent event stream (Y), any number of independent event streams (X1, X2,...Xn) and returns all coefficients of the regression equation

\[ y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon \]

The two implementations of regression could be distinguished as follows.
- **regress**: This allows you to specify the batch size (optional) that defines the number of events to be considered for the calculation of regression.
- **lengthTimeRegress**: This allows you to specify the time window and batch size (required). The number of events considered for the regression calculation can be restricted based on the time window and/or the batch size.

Input parameters for regress function

The following table describes the input parameters available for the `regress` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculation should be carried out.</td>
<td>Optional</td>
<td>1 (i.e., for every event)</td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events to be used for a regression calculation.</td>
<td>Optional</td>
<td>1,000,000,000</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream(s)</td>
<td>The data stream(s) of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

**Format**: `regress(Y, X1, X2,....,Xn)` or `regress(calculation interval, batch size, confidence interval, Y, X1, X2,....,Xn)`

Input parameters for `lengthTimeRegress` function

The following table describes the input parameters available for the `lengthTimeRegress` function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required/Optional</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Window</td>
<td>The maximum time duration to be considered for the regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Batch Size</td>
<td>The maximum number of events to be used for a regression calculation.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>Calculation Interval</td>
<td>The frequency with which the regression calculation should be carried out.</td>
<td>Optional</td>
<td>1 (for every event)</td>
</tr>
<tr>
<td>Confidence Interval</td>
<td>The confidence interval to be used for a regression calculation.</td>
<td>Optional</td>
<td>0.95</td>
</tr>
<tr>
<td>Y Stream</td>
<td>The data stream of the dependent variable.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>X Stream(s)</td>
<td>The data stream(s) of the independent variable.</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

**Format**: `lengthTimeRegress(time window, batch size, Y, X1, X2,....,Xn)` or `lengthTimeRegress(time window, batch size, calculation interval, confidence interval, Y, X1, X2,....,Xn)`.

Output parameters

The following table describes the output parameters.
The same output parameters are available for each implementation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Error</td>
<td>stdError</td>
<td>The standard error of the regression equation.</td>
</tr>
<tr>
<td>coefficients</td>
<td>beta0, betal, beta2 etc.</td>
<td>(n+1) coefficients where (n) is the number of (x) parameters</td>
</tr>
<tr>
<td>Input Stream</td>
<td>The name given in the input stream</td>
<td>All the attributes sent in the input stream.</td>
</tr>
</tbody>
</table>

The `regress` and `lengthTimeRegress` functions nullify any coefficients that fail the T-test based on the confidence interval. You can access any of the output parameters using its name (as given in the table above).

**Examples**

**Example 1**

The following query submits a calculation interval (every 10 events), a batch size (100,000 events), a confidence interval (0.95), a dependent input stream (Y) and 3 independent input streams (X1, X2, X3) that are used to perform linear regression between Y and all the X streams.

```plaintext
from StockExchangeStream#timeseries:regress(10, 100000, 0.95, Y, X1, X2, X3)
select *
insert into StockForecaster
```

When this query is executed, it returns the standard error of the regression equation ()\(, 4\) coefficients \((0, 1, 2, 3)\) and all the items available in the input stream. These results can be used to build a relationship between Y and all the Xs (regression equation) as follows.

\[ y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon \]  

**Example 2**

The following query submits a time window (200 milliseconds), a batch size (10,000 events), a calculation interval (every 2 events), a confidence interval (0.95), a dependent input stream (Y) and an independent input stream (X) that are used to perform linear regression between Y and all the X streams.

```plaintext
from StockExchangeStream#timeseries:lengthTimeRegress(200, 10000, 2, 0.95, Y, X)
select *
insert into StockForecaster
```

When this query is executed, it returns the standard error of the regression equation (), 2 coefficients \((0, 1)\) and all the items available in the input stream.

**kernelMinMax (Kernel based minima maxima detection)**

The `kernelMinMax` function uses Gaussian Kernel to smooth the time series values in the given window size, and then determine the maxima and minima of that set of values.

Input parameters
**Parameter** | **Required/Optional** | **Description**
---|---|---
Variable | Required | The time series value to be considered for minima maxima detection.
bandwidth | Required | The bandwidth of the Gaussian Kernel calculation.
window size | Required | The number of values to be considered for smoothing and determining the extremes.
Extrema type | Required | This can be min, max or minmax.

### Output parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min or max value</td>
<td>The variable name specified in the input parameter.</td>
<td>The value of the min or max point.</td>
</tr>
<tr>
<td>extremaType</td>
<td>extrema Type</td>
<td>Indicates whether the returned value is a min value or a max value.</td>
</tr>
</tbody>
</table>

### Examples

**Minimum values**

The following returns the maximum values for a set of price values.

```sql
from inputStream#timeseries:kernelMinMax(price, 3, 7, 'min')
select *
insert into outputStream;
```

**Maximum values**

The following returns the minimum values for a set of price values.

```sql
from inputStream#timeseries:kernelMinMax(price, 3, 7, 'max')
select *
insert into outputStream;
```

**Minimum and maximum values**

The following returns both the minimum values and the maximum values for a set of price values.

```sql
from inputStream#timeseries:kernelMinMax(price, 3, 7, 'minmax')
select *
insert into outputStream;
```

*kalmanMinMax (kalman based minima maxima detection)*
The `kalmanMinMax` function uses the `kalman` filter to smooth the time series values in the given window size, and then determine the maxima and minima of that set of values.

**Input parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Required</td>
<td>The time series value to be considered for minima maxima detection.</td>
</tr>
<tr>
<td>Q</td>
<td>Required</td>
<td>The standard deviation of the process noise.</td>
</tr>
<tr>
<td>R</td>
<td>Required</td>
<td>The standard deviation of the measurement noise.</td>
</tr>
<tr>
<td>Window Size</td>
<td>Required</td>
<td>The number of values to be considered for smoothing and determining the extremes.</td>
</tr>
<tr>
<td>Extrema Type</td>
<td>Required</td>
<td>This can be <code>min</code>, <code>max</code> or <code>minmax</code>.</td>
</tr>
</tbody>
</table>

**Output parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min or max value</td>
<td>min or max value</td>
<td>The value of the <code>min</code> or <code>max</code> point.</td>
</tr>
<tr>
<td>extrema Type</td>
<td>extrema Type</td>
<td>Indicates whether the returned value is a <code>min</code> value or a <code>max</code> value.</td>
</tr>
</tbody>
</table>

**Examples**

**Minimum values**

The following returns the maximum values for a set of `price` values.

```sql
from inputStream#timeseries:kalmanMinMax(price, 0.000001, 0.0001, 25, 'min')
select *
insert into outputStream;
```

**Maximum values**

The following returns the minimum values for a set of `price` values.

```sql
from inputStream#timeseries:kalmanMinMax(price, 0.000001, 0.0001, 25, max)
select *
insert into outputStream;
```

**Minimum and maximum values**

The following returns both the minimum values and the maximum values for a set of `price` values.
from inputStream#timeseries:kalmanMinMax(price, 0.000001, 0.0001, 25, 'minmax')
select *
insert into outputStream;

Math Extension

Math extension provides basic mathematical functions such as calculating absolute value, sin, cos, tan, base conversion, parsing, etc. Following are the functions of the Math extension.

- Absolute Value function
- acos function
- asin function
- atan function
- Binary function
- Ceiling function
- Convert function
- CopySign function
- cos function
- cosh function
- Cube Root function
- e function
- Exponential function
- Floor function
- Get Exponent function
- Hexadecimal function
- Is Infinite function
- Is Not A Number function
- ln function
- log2 function
- log10 function
- log function
- Max function
- Min function
- Octal function
- Parse Double function
- Parse Float function
- Parse Int function
- Parse Long function
- Percentile function
- pi function
- Power function
- Random function
- Round function
- Sign of Number function
- sin function
- sinh function
- Square Root function
- tan function
- tanh function
- To Degrees function
- To Radians function

**Absolute Value function**
### math:abs

| Syntax          | <double> math:abs(<float|double> p1) |
|-----------------|-------------------------------------|
| Extension Type  | Function                            |
| Description     | Returns the absolute value of p1. This function wraps the java.lang.Math.abs() function. |
| Examples        | Both the following queries return 3 since the absolute value of both 3 and -3 is 3.  
- abs(3)  
- abs(-3) |

### acos function

| Syntax          | <double> math:acos(<float|double> p1) |
|-----------------|-------------------------------------|
| Extension Type  | Function                            |
| Description     | If -1 <= p1 <= 1, this function returns the arc-cosine (inverse cosine) of p1. If not, it returns NULL. The return value is in radian scale. This function wraps the java.lang.Math.acos() function. |
| Examples        | acos(0.5) returns 1.0471975511965979. |

### asin function

| Syntax          | <double> math:asin (<float|double> p1) |
|-----------------|-------------------------------------|
| Extension Type  | Function                            |
| Description     | If -1 <= p1 <= 1, this function returns the arc-sin (inverse sine) of p1. If not, it returns NULL. The return value is in radian scale. This function wraps the java.lang.Math.asin() function. |
| Examples        | asin(0.5) returns 0.5235987755982989. |

### atan function

| Syntax          | <double> math:atan(<int|long|float|double> p1) |
|-----------------|-------------------------------------|
| Extension Type  | Function                            |
| Description     | Returns the arc-tangent (inverse tangent) of p1. The return value is in radian scale. This function wraps the java.lang.Math.atan() function. |
| Examples        | atan(6d) returns 1.4056476493802699. |

Syntax

<p>| Syntax          | &lt;double&gt; math:atan (&lt;int|long|float|double&gt; p1) |
|-----------------|-------------------------------------|
| Description     | Returns the arc-tangent (inverse tangent) of p1. The return value is in radian scale. This function wraps the java.lang.Math.atan() function. |
| Examples        | atan(6d) returns 1.4056476493802699. |</p>
<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the arc-tangent (inverse tangent) of ( p1 ) and ( p2 ) coordinates. The return value is in radian scale. This function wraps the ( \text{java.lang.Math.atan2()} ) function.</td>
</tr>
<tr>
<td>Examples</td>
<td>( \text{atan}(12d, 5d) ) returns 1.1760052070951352.</td>
</tr>
</tbody>
</table>

### Binary function

| Syntax          | \( \text{<string> \ math:bin(<int|long> p1)} \) |
|-----------------|---------------------------------------------|
| Extension Type  | Function                                   |
| Description     | Returns a string representation of the integer/long \( p1 \) argument as an unsigned integer in base 2. This function wraps the \( \text{java.lang.Integer.toBinaryString and java.lang.Long.toBinaryString} \) methods. |
| Examples        | \( \text{bin}(9) \) returns "1001". |

### Ceiling function

| Syntax          | \( \text{<double> \ math:ceil(<float|double> p1)} \) |
|-----------------|------------------------------------------------------|
| Extension Type  | Function                                             |
| Description     | Returns the smallest (closest to negative infinity) double value that is greater than or equal to the \( p1 \) argument, and is equal to a mathematical integer. This function wraps the \( \text{java.lang.Math.ceil()} \) method. |
| Example         | \( \text{ceil}(423.187d) \) returns 424.0. |

### Convert function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>( \text{&lt;string&gt; \ math:conv(&lt;string&gt; a, &lt;int&gt; fromBase, &lt;int&gt; toBase)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Converts ( a ) from the \text{fromBase} base to the \text{toBase} base.</td>
</tr>
<tr>
<td>Example</td>
<td>( \text{conv(&quot;7f&quot;, 16, 10) returns &quot;127&quot;.} )</td>
</tr>
</tbody>
</table>

### CopySign function
### copySign Function

**Syntax**

<double> math:copySign(<int|long|float|double> magnitude, <int|long|float|double> sign)

**Extension Type**

Function

**Description**

Returns the magnitude of `magnitude` with the sign of `sign`. This function wraps the `java.lang.Math.copySign()` function.

**Example**

copySign(5.6d, -3.0d) returns -5.6.

### cos Function

**Syntax**

<double> math:cos(<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Returns the cosine of `p1` (p1 is in radians). This function wraps the `java.lang.Math.cos()` function.

**Example**

cos(6d) returns 0.9601702866503661.

### cosh Function

**Syntax**

<double> math:cosh(<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Returns the hyperbolic cosine of `p1` (p1 is in radians). This function wraps the `java.lang.Math.cosh()` function.

**Example**

cosh(6d) returns 201.7156361224559.

### Cube Root function

**Syntax**

<double> math:cbrt(<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Returns the cube-root of `p1` (p1 is in radians). This function wraps the `java.lang.Math.cbrt()` function.

**Example**

cbrt(17d) returns 2.5712815906582356.

### e function

**Syntax**

<double> math:e

**Extension Type**

Function

**Description**

Returns the value of Euler's number (e) which is approximately 2.718281828459045.

**Example**

`e` returns 2.718281828459045.
### Syntax

**<double> math:e()**

### Function

**Description**
Returns the java.lang.Math.E constant, which is the closest double value to \(e\) (which is the base of the natural logarithms).

**Example**
\(e()\) returns 2.7182818284590452354.

---

### Exponential function

**Syntax**

**<double> math:exp(<int|long|float|double> p1)**

### Function

**Description**
Returns Euler’s number \(e\) raised to the power of \(p1\). This function wraps the java.lang.Math.exp() function.

**Example**
\(exp(10.23)\) returns 27722.51006805505.

---

### Floor function

**Syntax**

**<double> math:floor(<int|long|float|double> p1)**

### Function

**Description**
This function wraps the java.lang.Math.floor() function that returns the largest (closest to positive infinity) value that is less than or equal to \(p1\), and is equal to a mathematical integer.

**Example**
\(floor(10.23)\) returns 10.0.

---

### Get Exponent function

**Syntax**

**<double> math:getExponent(<int|long|float|double> p1)**

### Function

**Description**
Returns the unbiased exponent used in the representation of \(p1\). This function wraps the java.lang.Math.getExponent() function.

**Example**
\(getExponent(60984.1)\) returns 15.

---

### Hexadecimal function

**Syntax**

**<string> math:hex(<int|long|float|double> p1)**
### Extension Type
<table>
<thead>
<tr>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function wraps the <code>java.lang.Double.toHexString()</code> function that returns a hexadecimal string representation of <code>p1</code>.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>hex(200) returns &quot;c8&quot;</code>.</td>
</tr>
</tbody>
</table>

### `isInfinite` function

| Syntax | `<boolean> math:isInfinite(<float|double> p1)` |
|--------|-----------------------------------------------|
| Extension Type | Function |
| Description | This function wraps the `java.lang.Float.isInfinite()` and `java.lang.Double.isInfinite()` functions that return `true` if `p1` is infinitely large in magnitude, or return `false` otherwise. |
| Example | `isInfinite(java.lang.Double.POSITIVE_INFINITY) returns true.` |

### `isNaN` function

| Syntax | `<boolean> math:isNaN(<float|double> p1)` |
|--------|------------------------------------------|
| Extension Type | Function |
| Description | This function wraps the `java.lang.Float isNaN()` and `java.lang.Double isNaN()` functions that return `true` if `p1` is a NaN (Not-a-Number) value, or return `false` otherwise. |
| Example | `isNaN(java.lang.Math.log(-12d)) returns true.` |

### `ln` function

| Syntax | `<double> math:ln (<int|long|float|double> p1)` |
|--------|-----------------------------------------------|
| Extension Type | Function |
| Description | Returns the natural logarithm (base e) of `p1`. |
| Example | `ln(11.453) returns 2.438251704415579`. |

### `log2` function

<p>| Syntax | <code>&lt;double&gt; math:log2 (&lt;int|long|float|double&gt; p1)</code> |
|--------|-----------------------------------------------|
| Extension Type | Function |</p>
<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the base 2 logarithm of p1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>log2(91d) returns 6.507794640198696.</td>
</tr>
</tbody>
</table>

**log10 function**

| Syntax | `<double> math:log10 ( < int|long|float|double > p1 )` |
|--------|-------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns the base 10 logarithm of p1. |
| Example | log10(19.234) returns 1.2840696117100832. |

**log function**

| Syntax | `<double> math:log (< int|long|float|double > number, < int|long|float|double > base)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns the logarithm (base=base) of number. |
| Example | log(34, 2f) returns 5.08746284125034. |

**Max function**

| Syntax | `<double> math:max (< int|long|float|double > p1, <int|long|float|double> p2)` |
|--------|--------------------------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns the greater value out of p1 and p2. |
| Example | max(123.67d, 91) returns 123.67. |

**Min function**

| Syntax | `<double> math:min (< int|long|float|double > p1, <int|long|float|double> p2)` |
|--------|--------------------------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns the smaller value out of p1 and p2. |
| Example | min(123.67d, 91) returns 91. |
Octal function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;string&gt; math:oct (&lt;int,long&gt; p1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Converts p1 to octal.</td>
</tr>
<tr>
<td>Example</td>
<td>oct(99L) returns &quot;143&quot;.</td>
</tr>
</tbody>
</table>

Parse Double function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double&gt; math:parseDouble (&lt;string&gt; str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns str as a double.</td>
</tr>
<tr>
<td>Example</td>
<td>parseDouble(&quot;123&quot;) returns 123.0.</td>
</tr>
</tbody>
</table>

Parse Float function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;float&gt; math:parseFloat (&lt;string&gt; str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns str as a float.</td>
</tr>
<tr>
<td>Example</td>
<td>parseFloat(&quot;123&quot;) returns 123.0.</td>
</tr>
</tbody>
</table>

Parse Int function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;int&gt; math:parseInt (&lt;string&gt; str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns str as an int.</td>
</tr>
<tr>
<td>Example</td>
<td>parseInt(&quot;123&quot;) returns 123.</td>
</tr>
</tbody>
</table>

Parse Long function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;long&gt; math:parseLong (&lt;string&gt; str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns str as a long.</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Example</td>
<td><code>parseLong(&quot;123&quot;) returns 123.</code></td>
</tr>
</tbody>
</table>

**Percentile function**

| Syntax | `<double> math:percentile (<int|long|float|double> arg, <double> p)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns the pth percentile value of the arg values. |
| Example | `from inputStream#window.length(100)`  
`select math:percentile(temperature, 97.0) as percentile`  
`insert into outputStream;`  
`returns 97th percentile value of last 100 temperature values.` |

**pi function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;double&gt; math:pi ( )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the java.lang.Math.PI constant, which is the closest value to pi (i.e. the ratio of the circumference of a circle to its diameter).</td>
</tr>
<tr>
<td>Example</td>
<td><code>pi() always returns 3.141592653589793</code>.</td>
</tr>
</tbody>
</table>

**Power function**

| Syntax | `<double> math:power (<int|long|float|double> value, <int|long|float|double> toPower)` |
|--------|---------------------------------------------------------------------|
| Extension Type | Function |
| Description | Returns value raised to the power of toPower. |
| Example | `power(5.6d, 3.0d) returns 175.61599999999996`. |

**Random function**
### rand

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double&gt; <code>math:rand</code> ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>A sequence of calls to <code>rand()</code> generates a stream of pseudo-random numbers. This function uses the <code>java.util.Random</code> class internally.</td>
</tr>
<tr>
<td>Example</td>
<td>Two sequential calls to <code>rand()</code> may return <code>0.8263929447650588</code> and <code>0.24425883860361197</code> respectively.</td>
</tr>
</tbody>
</table>

| Syntax          | <double> `math:rand` (< int|long > `seed`) |
|-----------------|------------------------------------------|
| Extension Type  | Function                                 |
| Description     | A sequence of calls to `rand(seed)` generates a stream of pseudo-random numbers. This function uses the `java.util.Random` class internally. |
| Example         | Two sequential calls to `rand(12)` may return `0.7298928061101974` and `0.2750691655200749`, respectively. |

### Round function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;int&gt; <code>math:round</code> (&lt;float&gt; <code>value</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Funcion</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the closest integer value to the argument.</td>
</tr>
<tr>
<td>Example</td>
<td><code>round(3.35)</code> returns <code>3</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;long&gt; <code>math:round</code> (&lt;double&gt; <code>value</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the closest long value to the argument.</td>
</tr>
<tr>
<td>Example</td>
<td><code>round(3252.353)</code> returns <code>3252</code>.</td>
</tr>
</tbody>
</table>

### Sign of Number function

| Syntax          | <int> `math:signum` (< int|long|float|double > `p1`) |
|-----------------|-----------------|
| Extension Type  | Function        |
| Description     |                 |
| Example         |                 |
### Description

- If `a` is a positive, this returns the sign of `p1` as `1.0`.
- If `a` is a negative, this returns the sign of `p1` as `-1.0`.
- If `a` is neither a positive or a negative, this returns the sign of `p1` as `0.0`.

This function wraps the `java.lang.Math.signum()` function.

### Example

```
signum(-6.32d) returns -1.
```

---

### sin function

| Syntax | `<double> math:sin (< int|long|float|double > p1)` |
|---|---|
| Extension Type | Function |
| Description | Returns the sine of `p1` (p1 is in radians). This function wraps the `java.lang.Math.sin()` function. |
| Example | `sin(6d) returns -0.27941549819892586.` |

---

### sinh function

| Syntax | `<double> math:sinh (< int|long|float|double > p1)` |
|---|---|
| Extension Type | Function |
| Description | Returns the hyperbolic sine of `p1` (p1 is in radians). This function wraps the `java.lang.Math.sinh()` function. |
| Example | `sinh(6d) returns 201.71315737027922.` |

---

### Square Root function

| Syntax | `<double> math:sqrt (< int|long|float|double > p1)` |
|---|---|
| Extension Type | Function |
| Description | Returns the square-root of `p1`. This function wraps the `java.lang.Math.sqrt()` function. |
| Example | `sqrt(4d) returns 2.` |

---

### tan function

| Syntax | `<double> math:tan (< int|long|float|double > p1)` |
### Char At function

**Syntax**

<double> **math:charAt** (<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Returns the character at position `p1` in the string. This function wraps the `java.lang.String.charAt()` function.

**Example**

t.charAt(6) returns "E".

---

### tanh function

**Syntax**

<double> **math:tanh** (<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Returns the hyperbolic tangent of `p1` (p1 is in radians). This function wraps the `java.lang.Math.tanh()` function.

**Example**

tanh(6d) returns 0.9999877116507956.

---

### To Degrees function

**Syntax**

<double> **math:toDegrees** (<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Converts `p1` from radians to degrees. This function wraps the `java.lang.Math.toDegrees()` function.

**Example**

toDegrees(6d) returns 343.77467707849394.

---

### To Radians function

**Syntax**

<double> **math:toRadians** (<int|long|float|double> p1)

**Extension Type**

Function

**Description**

Converts `p1` from degrees to radians. This function wraps the `java.lang.Math.toRadians()` function.

**Example**

toRadians(6d) returns 0.10471975511965977.

---

### String Extension

This extension provides basic string handling capabilities such as con-cat, length, convert to lowercase, replace all, etc. String extension can be used in a query as follows. Following are the functions of the String extension.

- **Char At function**
Coalesce function
Concatenation function
Contains function
Hexadecimal function
Length function
Lower Case function
Regular Expression function
Repeat function
Replace All function
Replace First function
Reverse function
Split function
String Compare function
Sub String function
Trim function
Unhexadecimal function
UpperCase function

Coalesce function

Syntax

\[ \text{str:coalesce}(\text{arg1}, \ldots, \text{argN}) \]

Extension Type
Function

Description
Returns the value of the first of its input parameters that is not null.

Parameters
This function accepts any number of parameters. The parameters can be of different types.

Return Type
This is the same as the type of the first input parameter that is not null.

Examples
- \text{coalesce}("123", \text{null}, "789") returns "123".
- \text{coalesce}(\text{null}, "BBB", "CCC") returns "BBB".
- \text{coalesce}(\text{null}, \text{null}, \text{null}) returns null.

Concatenation function
### Syntax

```
<string> str:concat ( <int|long|float|double|string|boolean > arg1, < int|long|float|double|string|boolean > arg2, ..., < int|long|float|double|string|boolean > argN )
```

### Extension Type

Function

### Description

Returns a string that is the result of concatenating the given arguments: `arg1, arg2, .., argN`.

### Examples

- `concat("D533", "8JU^", "XYZ")` returns `"D53 38JU^XYZ"`
- `concat("AAA", null, "CCC")` returns `"AAACCC"`

### Contains function

#### Syntax

```
<bool> str:contains ( < string > inputSequence, < string > searchingSequence )
```

#### Extension Type

Function

#### Description

This method returns `true` if `inputSequence` contains the specified sequence of char values in the `searchingSequence`.

#### Example

`contains("21 products are produced by WSO2 currently", "WSO2")` returns `true`.

### Hexadecimal function

#### Syntax

```
<string> str:hex ( < string > str )
```

#### Extension Type

Function

#### Description

Returns a hexadecimal string representation of `str`.

#### Example

`hex("MySQL")` returns `"4d7953514c"`.

### Length function

#### Syntax

```
<int> str:length ( < string > str )
```

#### Extension Type

Function
<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the length of the string: <code>str</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td><code>length(&quot;Hello World&quot;)</code> returns 11.</td>
</tr>
</tbody>
</table>

**Lower Case function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:lower ( &lt; string&gt; str)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Converts the capital letters in the <code>str</code> input string to the equivalent simple letters.</td>
</tr>
<tr>
<td>Example</td>
<td><code>lower(&quot;WSO2 cep &quot;)</code> returns &quot;wso2 cep &quot;.</td>
</tr>
</tbody>
</table>

**Regular Expression function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;boolean&gt; str:regexp ( &lt; string&gt; str, &lt;string&gt; regex )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns <code>true</code> if the given string (i.e. <code>str</code>) matches the given regular expression (i.e. <code>regex</code>). Returns <code>false</code> if the string does not match the regular expression.</td>
</tr>
<tr>
<td>Example</td>
<td><code>regexp(&quot;WSO2 abcdh&quot;, &quot;WSO(.*h)&quot;)</code> returns <code>true</code>.</td>
</tr>
</tbody>
</table>

**Repeat function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:repeat ( &lt; string&gt; str, &lt;int&gt; times)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Repeats the specified string (i.e. <code>string</code>) for the specified number of times (i.e. <code>times</code>).</td>
</tr>
<tr>
<td>Example</td>
<td><code>repeat(&quot;String 1&quot;, 3)</code> returns &quot;String 1String 1String 1&quot;.</td>
</tr>
</tbody>
</table>

**Replace All function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:replaceAll ( &lt; string&gt; str, &lt;string&gt; regex , &lt;string&gt; replacement)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Replaces each substring of the given string (i.e. <code>str</code>) that matches the given regular expression (i.e. <code>regex</code>) with the string specified as the replacement (i.e. <code>replacement</code>).</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>function</th>
<th>code</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>replaceAll</td>
<td><code>replaceAll(&quot;hello hi hello&quot;, 'hello', 'test')</code></td>
<td>&quot;test hi test&quot;</td>
</tr>
<tr>
<td>Replace First function</td>
<td><code>str:replaceFirst(&lt;string&gt; str, &lt;string&gt; regex, &lt;string&gt; replacement)</code></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>replaceFirst(&quot;hello WSO2 A hello&quot;, 'WSO2(.*)A', 'XXXX')</code></td>
<td>&quot;hello XXXX hello&quot;</td>
</tr>
<tr>
<td>Reverse function</td>
<td><code>str:reverse(&lt;string&gt; str)</code></td>
<td>&quot;dlroW olleH&quot;</td>
</tr>
<tr>
<td>Split function</td>
<td><code>str:split(&lt;string&gt; sourceString, &lt;string&gt; splitCharacter, &lt;int&gt; returnedOutputPosition)</code></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>split(&quot;WSO2,ABM,NSFT&quot;, &quot;,&quot;, 0)</code></td>
<td>WSO2</td>
</tr>
<tr>
<td>String Compare function</td>
<td><code>str:strcmp(&lt;string&gt; str, &lt;string&gt; compareTo)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Extension Type
Function

## Description
Compares `str` with `compareTo` strings lexicographically.

## Examples
- `strcmp("Hello", 'Hello')` returns 0.

### Sub String function

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:substr ( &lt; string&gt; sourceText, &lt;int&gt; startIndex )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns a new string that is a substring of <code>sourceText</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>substr(&quot;AbCDefghiJ KLMN&quot;, 4)</code> returns &quot;efghiJ KLMN&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:substr ( &lt; string&gt; sourceText, &lt;int&gt; startIndex, &lt;int&gt; length )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns a new string that is a substring of <code>sourceText</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>substr(&quot;AbCDefghiJ KLMN&quot;, 2, 4)</code> returns &quot;CDeff&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:substr ( &lt; string&gt; sourceText, &lt;string&gt; regex)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns a new string that is a substring of <code>sourceText</code>.</td>
</tr>
<tr>
<td>Examples</td>
<td><code>substr(&quot;WSO2D efghiJ KLMN&quot;, '^WSO2(.*)')</code> returns &quot;WSO2D efghiJ KLMN&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; str:substr ( &lt; string&gt; sourceText, &lt;string&gt; regex, &lt;int&gt; groupNumber)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns a new string that is a substring of <code>sourceText</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>substr(&quot;WSO2 cep WSO2 XX E hi hA WSO2 heAllo&quot;, 'WSO2(.*)', 2)</code> returns &quot;hello&quot;.</td>
</tr>
</tbody>
</table>

### Trim function
### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt; str:trim ( &lt; string&gt; str)</code></td>
<td>Function</td>
</tr>
</tbody>
</table>

### Extension Type

- **Description**: Returns a copy of `str`, with the leading and/or trailing white-spaces omitted.

### Example

```
trim(" AbCDefghiJ KLMN ") returns "AbCDefghiJ KLMN".
```

---

### Unhexadecimal function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt; str:unhex ( &lt; string&gt; str)</code></td>
<td>Function</td>
</tr>
</tbody>
</table>

### Description

This is the equivalent of the `unhex` function in mysql 5.0. `unhex(str)` interprets each pair of characters in `str` as a hexadecimal number. Also see `hex () string extension`.

### Example

```
unhex("4d7953514c") returns "MySQL".
```

---

### Upper Case function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt; str:upper ( &lt;string&gt; str)</code></td>
<td>Function</td>
</tr>
</tbody>
</table>

### Description

Converts the simple letters in the given input string (i.e. `str`) to the equivalent capital letters.

### Example

```
upper("Hello World") returns "HELLO WORLD".
```

---

### Geo Extension

This extension provides geo data related functionality such as checking whether a given geo coordinate is within a predefined geo-fence, etc. Following are the functions of the Geo extension:

- Intersects function
- Within function
- Within Distance function
- Crosses function
- Stationary function
- Proximity function
- Geo Coordinates function
- Geo Distance function

### Intersects function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt; geo:intersects ( &lt;string&gt; geoJSONGeometry , &lt;string&gt; geoJSONGeometryFence )</code></td>
<td>Function</td>
</tr>
</tbody>
</table>
Description

Returns true if the `geoJSONGeometry` incoming event intersects the given string (i.e., `geoJSONGeometryFence`). Returns false otherwise.

Example

```
intersects(
{'type':'Polygon','coordinates':[[0.5, 0.5],[0.5, 1.5],[1.5, 1.5],[1.5, 0.5],[0.5, 0.5]]},
{'type':'Polygon','coordinates':[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]}) , returns true because geoJSONGeometry intersects geoJSONGeometryFence.
```

Syntax

```
<bool> geo:intersects (<double> longitude , <double> latitude , <string> geoJSONGeometryFence )
```

Extension Type

Function

Description

Returns true if the location specified in terms of longitude and latitude intersects the given `geoJSONGeometryFence`. Returns false otherwise.

Example

```
intersects(0.5, 0.5 ,
{'type':'Polygon','coordinates':[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]}) , returns true because the location specified in terms of longitude and latitude intersects geoJSONGeometryFence.
```

Within function

Syntax

```
<bool> geo:within (<double> longitude , <double> latitude , <string> geoJSONGeometryFence )
```

Extension Type

Function

Description

Returns true if the location specified in terms of longitude and latitude is within the `geoJSONGeometryFence`.

Examples

- `within(0.5, 0.5,
{'type':'Polygon','coordinates':[[0,0],[0,2],[1,2],[1,0],[0,0]]})` returns true.
- `within(2, 2,
{'type':'Polygon','coordinates':[[0,0],[0,2],[1,2],[1,0],[0,0]]})` returns false.

Syntax

```
<bool> geo:within (<string> geoJSONGeometry , <string> geoJSONGeometryFence )
```
<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns true if the <code>geoJSONGeometry</code> is within the <code>geoJSONGeometryFence</code>. Returns false otherwise.</td>
</tr>
</tbody>
</table>
| Example        | - `within( {'type': 'Circle', 'radius': 110575, 'coordinates':[1.5, 1.5]}, {'type':'Polygon','coordinates':[[0,0],[0,4],[3,4],[3,0],[0,0]]}) returns true.`  
- `within( {'type': 'Circle', 'radius': 110575, 'coordinates':[0.5, 1.5]}, {'type':'Polygon','coordinates':[[0,0],[0,4],[3,4],[3,0],[0,0]]}) returns false.` |

**Within Distance function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;bool&gt; geo:withindistance (&lt;double&gt; longitude, &lt;double&gt; latitude, &lt;string&gt; geoJSONGeometryFence)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns true if the location specified in terms of longitude and latitude is within distance of the <code>geoJSONGeometry</code>. Returns false otherwise.</td>
</tr>
<tr>
<td>Example</td>
<td><code>withindistance( 0.5, 0.5, {'type':'Polygon','coordinates':[[0,0],[0,1],[1,1],[1,0],[0,0]], 110574.6108775787}) returns true because the location specified in terms of longitude and latitude is within the distance of the </code>geoJSONGeometryFence<code>.</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;bool&gt; geo:withindistance (&lt;string&gt; geoJSONGeometry , &lt;string&gt; geoJSONGeometryFence , &lt;double&gt; distance)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns true if the area given by <code>geoJSONGeometry</code> is within distance of the <code>geoJSONGeometryFence</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>withindistance( {'type':'Polygon','coordinates':[[0.5, 0.5],[0.5, 1.5],[1.5, 1.5],[1.5, 0.5],[0.5, 0.5]]}, {'type':'Polygon','coordinates':[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]}, 110574.6108775787) returns true because geoJSONGeometry is within the distance of geoJSONGeometryFence.</code></td>
</tr>
</tbody>
</table>
### Crosses function

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt;</code> <code>geo:crosses</code> (&lt;string&gt; <code>id</code> , &lt;double&gt; <code>longitude</code>, &lt;double&gt; <code>latitude</code>, &lt;string&gt; <code>geoJSONGeometryFence</code>)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns <code>true</code> when the specified object of which the location is specified in terms of <code>longitude</code> and <code>latitude</code> crosses the geographic location specified in <code>geoJSONGeometryFence</code>. Returns <code>false</code> when the object crosses out of the location specified in <code>geoJSONGeometryFence</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| * crosses(km-4354, -0.5, 0.5, 
  `{type':'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]]}) return `true`.
* km-4354, 1.5, 0.5, 
  `{type':'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]]}) return `true`. |

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt;</code> <code>geo:crosses</code> (&lt;string&gt; <code>id</code> , &lt;string&gt; <code>geoJSONGeometry</code>, &lt;string&gt; <code>geoJSONGeometryFence</code>)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamProcessor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns <code>true</code> when the object (i.e. <code>geoJSONGeometry</code>) crosses the specified geographic location (i.e. <code>geoJSONGeometryFence</code>). Returns <code>false</code> when the object crosses out of <code>geoJSONGeometryFence</code>.</td>
</tr>
</tbody>
</table>

### Stationary function

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt;</code> <code>geo:stationary</code> (&lt;string&gt; <code>id</code> , &lt;double&gt; <code>longitude</code>, &lt;double&gt; <code>latitude</code>, &lt;string&gt; <code>geoJSONGeometryFence</code>, &lt;double&gt; <code>radius</code>)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamProcessor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns <code>true</code> when the object (defined in terms of <code>longitude</code> and <code>latitude</code>) becomes stationary within the specified <code>radius</code>. Returns <code>false</code> when the object moves out of the specified <code>radius</code>.</td>
</tr>
</tbody>
</table>
### Example

- `stationary(km-4354, 0, 0, 110574.61087757687)` returns `true`.
- `stationary(km-4354, 1, 1, 110574.61087757687)` returns `true`.
- `stationary(km-4354, 1, 1.5, 110574.61087757687)` returns `true`.

### Syntax

```c
<bool> geo:stationary (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius )
```

### Extension Type

StreamProcessor

### Description

Returns `true` when the object (i.e. `geoJSONGeometry`) becomes stationary within the specified `radius`. Returns `false` when the object moves out of the specified `radius`.

---

### Proximity function

#### Syntax

```c
<bool,string> geo:proximity (<string> id , <double> longitude, <double> latitude, <string> geoJSONGeometryFence , <double> radius )
```

#### Extension Type

StreamProcessor

#### Description

Returns `true` when two objects (specified in terms of `longitude` and `latitude`) are within the specified `radius` to another object. Returns `false` when the specified object moves out of the specified `radius`. The `proximityWith` optional attribute indicates the ID of the object that the object specified is in close proximity with. `proximityID` is a unique ID for the two objects in close proximity.

#### Example

The following return `true` with `id 3`.
- `proximity(1, 0, 0, 110574.61087757687)`
- `proximity(2, 1, 1, 110574.61087757687)`
- `proximity(3, 2, 2, 110574.61087757687)`
- `proximity(1, 1.5, 1.5, 110574.61087757687)`

#### Syntax

```c
<bool> proximity (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius )
```

#### Extension Type

StreamProcessor
Description

Returns true when an object (i.e. `geoJSONGeometry`) is within the specified `radius` from another object. Returns false when one or both objects move away from each other and are no longer within the specified `radius` of each other. The `proximityWith` optional attribute indicates the ID of the object that the object specified is in close proximity with. `proximityID` is a unique ID for the two objects in close proximity.

**Geo Coordinates function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double, double, string&gt; <code>geocode</code> (&lt;string&gt; <code>location</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>StreamProcessor</td>
</tr>
<tr>
<td>Description</td>
<td>Transforms a location to its geo-coordinates (longitude and latitude) and formatted address.</td>
</tr>
<tr>
<td>Example</td>
<td><code>geocode(duplication rd)</code> returns the following data with adhering latitude, longitude, and formattedAddress attribute names respectively. 6.8995244d, 79.8556202d, &quot;R A De Mel Mawatha, Colombo, Sri Lanka&quot;</td>
</tr>
</tbody>
</table>

**Geo Distance function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double&gt; <code>distance</code> (&lt;double&gt; <code>latitude1</code>, &lt;double&gt; <code>longitude1</code>, &lt;double&gt; <code>latitude2</code>, &lt;double&gt; <code>longitude2</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Gives distance between two geo locations in meters.</td>
</tr>
<tr>
<td>Example</td>
<td><code>distance(latitude, longitude, prevLatitude, prevLongitude)</code> returns 2322119.848252557</td>
</tr>
</tbody>
</table>

**R Language Extension**

First you need to setup the prerequisites.

This extension allows you to execute R scripts within Siddhi query. Following are the functions of the R extension.

- **Evaluation function**
- **Evaluation Source function**

**Evaluation function**
### Syntax

```
<int|long|float|double|string|boolean>, <int|long|float|double|string|boolean> output1, ... ] r:eval ( <string > script, <string > outputAttributes, <int|long|float|double|string|boolean> input1, ... ) input2
```

### Extension Type

StreamProcessor

### Description

This runs the R script for each event and produces aggregated outputs based on the provided input variable parameters and expected output attributes.

### Parameters

- **script**: R script as a string produces aggregated outputs based on the provided input variable parameters and expected output attributes.
- **outputAttributes**: A set of output attributes separated by commas as string. Each attribute is denoted as `<name><space><type>`. e.g., `'output1 string, output2 long'`

### Return Parameters

Output parameters are generated based on the outputAttributes provided.

### Example

```
eval('totalItems <- sum(items); totalTemp <- sum(temp);', 'totalItems int, totalTemp double', items, temp), where the data type of items is int and that of temp is double returns [ totalItems, totalTemp ], where the data type of totalItems is int and that of totalTemp is double.
```

```r
@info(name = 'query1')
from
dataIn#window.lengthBatch(2)#r :eval('totalItems <- sum(items); totalTemp <- sum(temp);', 'totalItems int, totalTemp double', items, temp)
select *
insert into dataOut;
```

### Evaluation Source function
Syntax

```plaintext
[<int|long|float|double|string|boolean> output1 ,  <int|long|float|double|string|boolean> output2, ... ] r:eval ( <string> filePath, <string> outputAttributes,<int|long|float|double|string|boolean> input1 , <int|long|float|double|string> input2 , ... )
```

Extension Type

Stream Processor

Description

This runs the R script loaded from a file to each event and produces aggregated outputs based on the provided input variable parameters and expected output attributes.

Parameters

- `filePath`: The file path of the R script where this script uses the input variable parameters and produces the expected output attributes.
- `outputAttributes`: A set of output attributes separated by commas as string. Each attribute is denoted as `<name><space><type>`, e.g., `output1 string, output2 long`

Return

Output parameters are generated based on the outputAttributes provided.

Example

```plaintext
evalSource('/home/user/test/script1.R', 'totalItems int, totalTemp double', items, temp), where the data type of items is int and that of temp is double returns [ totalItems, totalTemp ] where the data type of totalTemp is int and that of totalTemp is double.
```

```plaintext
@info(name = 'query1')
from
dataIn#window.lengthBatch(2)#r :evalSource('/home/user/test/script1.R', 'totalItems int, totalTemp double', items, temp)
select *
insert into dataOut;
```

Regular Expression Extension

This extension provides basic RegEx execution capabilities to Siddhi. Following are the functions of the RegEx extension.

- Find function
- Group function
- Looking At function
- Matches function
**Find function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;bool&gt; regex:find (&lt;string&gt; regex , &lt;string&gt; inputSequence )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>This method attempts to find the next sub-sequence of the <code>inputSequence</code> that matches the <code>regex</code> pattern. It returns <code>true</code> if such a sub sequence exists, or returns <code>false</code> otherwise.</td>
</tr>
</tbody>
</table>
| Examples | • `find("\d\d(.*)WSO2", "21 products are produced by WSO2 currently")` returns `true`.  
• `find("\d\d(.*)WSO2", "21 products are produced currently")` returns `false`. |

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;bool&gt; regex:find (&lt;string&gt; regex , &lt;string&gt; inputSequence , &lt;int&gt; startingIndex )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>This method attempts to find the next sub-sequence of the <code>inputSequence</code> that matches the <code>regex</code> pattern starting from given index (i.e. <code>startingIndex</code>). It returns <code>true</code> if such a sub sequence exists, or returns <code>false</code> otherwise.</td>
</tr>
</tbody>
</table>
| Examples | • `find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 30)` returns `true`.  
• `find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 35)` returns `false`. |

**Group function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; regex:group (&lt;string&gt; regex , &lt;string&gt; inputSequence , &lt;int&gt; groupId )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the input sub-sequence captured by the given group during the previous match operation. Returns <code>null</code> if no sub-sequence was found during the previous match operation. For more information about the match operation, see matches.</td>
</tr>
</tbody>
</table>
### Example

| group("\((\d\d)\)(.*)\)WSO2.*", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 3) returns "WSO2 employees". |

---

### Looking At function

**Syntax**

```
<string> regex:lookingAt (<string> regex , <string> inputSequence )
```

**Extension Type**

Function

**Description**

This method attempts to match the `inputSequence` against the `regex` pattern starting at the beginning.

**Examples**

- `lookingAt("\d\d(.*)WSO2", "21 products are produced by WSO2 currently in Sri Lanka") returns true.
- `lookingAt("WSO2(.*)middleware(.*)", "sample test string and WSO2 is situated in trace and its a middleware company") returns false.`

### Matches function

**Syntax**

```
<string> regex:matches (<string> regex , <string> inputSequence )
```

**Extension Type**

Function

**Description**

This method attempts to match the entire `inputSequence` against the `regex` pattern.

**Examples**

- `matches("WSO2(.*)middleware(.*)", "WSO2 is situated in trace and its a middleware company") returns true.
- `matches("WSO2(.*)middleware", "WSO2 is situated in trace and its a middleware company") returns false.`

### Time Extension

This extension provides time related functionality to Siddhi such as getting the current time, current date, manipulating/formatting dates, etc. Following are the functions of the time extension.

- Current Date function
- Current Time function
- Current Timestamp function
- Date Adding function
- Date Subtraction function
- Date Difference function
- Date Format function
- Extract function
- Date function
### Current Date function

**Syntax**

```
<string> time: currentDate ( )
```

**Extension Type**

Function

**Description**

Returns the current system date in the `yyyy-MM-dd` format.

**Example**

```
currentDate() returns 2015-08-20.
```

### Current Time function

**Syntax**

```
<string> currentTime ( )
```

**Extension Type**

Function

**Description**

Returns the current system time in the `HH:mm:ss` format.

**Example**

```
```

### Current Timestamp function

**Syntax**

```
<string> time: currentTimestamp ( )
```

**Extension Type**

Function

**Description**

Returns the current system timestamp in the `yyyy-MM-dd HH:mm:ss` format.

**Example**

```
```

### Date Adding function

The common parameters of this function are described below.

- **expr**: The amount by which the selected part of the date format should be incremented. e.g., 2, 5, 10 etc.
- **unit**: The part of the date format that needs to be manipulated. e.g., "MINUTE", "HOUR", "MONTH", "YEAR", "QUARTER", "WEEK", "DAY", "SECOND"
- **dateFormat**: The date format of the date value provided. e.g., `yyyy-MM-dd HH:mm:ss.SSS`
- **timestampInMilliseconds**: The date value in milliseconds (from the epoch). e.g., `1415712224000L`

**Syntax**

```
<string> time: dateAdd (<string> dateValue, <long> expr, <string> unit, <string> dateFormat )
```

**Extension Type**

Function
### Description

Returns the specified date and time with the selected unit of the specified `dateValue` incremented by the given amount (i.e. `expr`).

### Example

```
```

### Syntax

```
<string> time: dateAdd (<string> dateValue , < long > expr, <string> unit )
```

### Extension Type

Function

### Description

Returns the specified date and time with the selected unit of the specified `dateValue` incremented by the given amount (i.e. `expr`).

### Example

```
```

### Syntax

```
<string> time: dateAdd (<long> timestampInMilliseconds, < long > expr, <string> unit )
```

### Extension Type

Function

### Description

Returns the specified time stamp with the selected unit of the specified `timestampInMilliseconds` incremented by the given amount (i.e. `expr`).

### Example

```
dateAdd(1415692424000L, 2, 'year') returns "2016-11-11 13:23:44".
```

---

### Date Subtraction function

The common parameters of this function are described below.

- **expr**: The amount by which the selected part of the date format should be reduced. e.g., 2, 5, 10 etc.
- **unit**: The part of the date format that needs to be manipulated. e.g., "MINUTE", "HOUR", "MONTH", "YEAR", "QUARTER", "WEEK", "DAY", "SECOND"
- **dateFormat**: The date format of the date value provided. e.g., "yyyy-MM-dd HH:mm:ss.SSS"
- **timestampInMilliseconds**: The date value in milliseconds (from the epoch). e.g., 1415712224000L

### Syntax

```
<string> time: dateSub (<string> dateValue , < long > expr, <string> unit, <string> dateFormat )
```

### Extension Type

Function

### Description

Returns the specified date and time with the selected unit of the specified `dateValue` reduced by the given amount (i.e. `expr`).
**Example**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Date Sub Example</th>
</tr>
</thead>
</table>

**Syntax**

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns the specified date and time stamp with the selected unit of the specified dateValue reduced by the given amount (i.e. expr).</td>
</tr>
</tbody>
</table>

**Syntax**

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns the specified time stamp with the selected unit of the specified timestampInMilliseconds reduced by the given amount (i.e. expr).</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>dateSub(1415692424000L, 2, 'year') returns 1352620424000.</td>
</tr>
</tbody>
</table>

**Date Difference function**

The common parameters of this function are described below.

- **dateFormat1**: The date format of dateValue1. e.g., yyyy-MM-dd HH:mm:ss.SSS
- **dateFormat2**: The date format of dateValue2. e.g., yyyy-MM-dd HH:mm:ss.SSS
- **timestampInMilliseconds1**: A date value in milliseconds (from the epoch) e.g., 1415712224000L
- **timestampInMilliseconds2**: A date value in milliseconds (from the epoch) e.g., 1415712224000L

**Syntax**

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns the number of days between the two dates specified (i.e. dateValue1 and dateValue2).</td>
</tr>
</tbody>
</table>
### Syntax

```c
<int> time: dateDiff (<string> dateValue1 , < string > dateValue2 )
```

### Extension Type

Function

### Description

Returns the number of days between the two dates specified (i.e. `dateValue1` and `dateValue2`).

### Example

```c
```

### Syntax

```c
<int> time: dateDiff (<string> timestampInMilliseconds1 , < string > timestampInMilliseconds2 )
```

### Extension Type

Function

### Description

Returns the number of days between the two date and time stamps specified (i.e. `timestampInMilliseconds1` and `timestampInMilliseconds2`).

### Example

```c
dateDiff(1415692424000, 1415519624000) returns 2.
```

---

**Date Format function**

The common parameters of this function are described below.

- **`dateTargetFormat`**: The date format to which the specified `date value` needs to be converted. e.g., `yy/MM/dd HH:mm:ss`
- **`dateSourceFormat`**: The date format of the `date value` provided. e.g., `yyyy-MM-dd HH:mm:ss.SSS`
- **`timestampInMilliseconds`**: A date value in milliseconds (from the epoch) e.g., `1415712224000L`

### Syntax

```c
<string> time: dateFormat (<string> dateValue , <string> dateTargetFormat , <string> dateSourceFormat )
```

### Extension Type

Function

### Description

Returns a formatted date string.

### Example

```c
```

### Syntax

```c
<string> time: dateFormat (<string> dateValue , <string> dateTargetFormat )
```

### Extension Type

Function

### Description

Returns a formatted date string.

### Example

```c
```
### Syntax

<string>  

**Extension Type**

Function

**Description**

Returns a formatted date string.

**Example**

dateFormat(1415692424000, 'yyyy-MM-dd') returns 2014-11-11.

### Extract function

- **unit**: The part of the date format that needs to be manipulated. e.g., "MINUTE", "HOUR", "MONTH", "YEAR", "QUARTER", "WEEK", "DAY", "SECOND"
- **dateFormat**: The date format of the date value provided. e.g., yyyy-MM-dd HH:mm:ss
- **timestampInMilliseconds**: A date value in milliseconds (from the epoch) e.g., 1415712224000L

| Syntax | <int>  

**Extension Type**

Function

**Description**

Returns the specified **unit** extracted from the specified **dateValue**.

**Example**


| Syntax | <int>  

**Extension Type**

Function

**Description**

Returns the specified **unit** extracted from the specified **dateValue**.

**Example**

extract('year', '2014-3-11 02:23:44.234') returns 2014.

| Syntax | <int>  

**Extension Type**

Function

**Description**

Returns the specified **unit** extracted from the specified **timestampInMilliseconds**

**Example**

extract(1394484824000, 'year') returns 2014.

### Date function

| Syntax | <string>  

**Extension Type**

Function

**Description**

**Example**

### Timestamp In Milliseconds function

**Syntax**

```
<long> time: timestampInMilliseconds ()
```

**Extension Type** Function

**Description** Returns the current time stamp in milliseconds.

**Example**

```java
timestampInMilliseconds() returns 1440160328693.
```

**Syntax**

```
<long> time: timestampInMilliseconds (<string> dateValue)
```

**Extension Type** Function

**Description** Returns the time stamp of the specified `dateValue` in milliseconds. In order to use this function, the date format of the specified `dateValue` should be `yyyy-MM-dd HH:mm:ss.SSS`.

**Example**

```java
timestampInMilliseconds('2007-11-30 10:30:19.000') returns 1196398819000.
```

**Syntax**

```
<long> time: timestampInMilliseconds (<string> dateValue, <string> dateFormat)
```

**Extension Type** Function

**Description** Returns the time stamp of the specified `dateValue` in milliseconds. The date format can be specified in the `dateFormat` parameter.

**Example**

```java
```

### UTC Timestamp function

**Syntax**

```
<string> time: utcTimestamp()
```

**Extension Type** Function

**Description** Returns the system time in the `yyyy-MM-dd HH:mm:ss` date format.

**Example**

```java
```

### Natural Language Processing Extension

This extension provides Natural Language Processing capabilities to Siddhi. Functions of the NLP extension are as follows.
Find Name Entity Type function

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; nlp:findNameEntityType(&lt;string&gt; entityType, &lt;bool&gt; groupSuccessiveMatch, &lt;string&gt; string-variable )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>This function uses the following input parameters.</td>
</tr>
<tr>
<td></td>
<td>* entityType: This is a user-specified string constant.</td>
</tr>
<tr>
<td></td>
<td>e.g., PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME</td>
</tr>
<tr>
<td></td>
<td>* groupSuccessiveMatch: This is a user-specified boolean constant used to group successive matches</td>
</tr>
<tr>
<td></td>
<td>of the specified entityType and a text stream.</td>
</tr>
<tr>
<td></td>
<td>* streamAttribute: A string or the stream attribute</td>
</tr>
<tr>
<td></td>
<td>in which text stream is included.</td>
</tr>
</tbody>
</table>

This function returns the entities in the text. If you specify group successive matches as true, the result aggregates successive words of the same entity type.

Example

```plaintext
findNameEntityType("PERSON",true,text)
```

In the above example, if the text attribute contains "Bill Gates donates £31million to fight Ebola", the result is Bill Gates. If the group successive match is set to false, two events are generated as Bill and Gates.

Find Name Entity Type Via Dictionary function

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;string&gt; nlp:findNameEntityTypeViaDictionary(&lt;string&gt; entityType, &lt;string&gt; dictionary, &lt;string&gt; dictionaryFilePath, &lt;string&gt; string-variable )</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
</tbody>
</table>

Description

This function uses the following input parameters.

- **entityType**: This is a user-specified string constant. e.g., PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME
- **dictionaryFilePath**: The path to the dictionary in which the function searches for the specified entries. The relevant entries for the entity types should be available in the dictionary as shown in the example below.

  ```
  <dictionary>
    <entity id="PERSON">
      <entry>Bill</entry>
      <entry>Addison</entry>
    </entity>
    <entity id="LOCATION">
      <entry>Mississippi</entry>
      <entry>Independence Square</entry>
    </entity>
    <entity id="ORGANIZATION">
      <entry>WSO2</entry>
    </entity>
  </dictionary>
  ```

- **streamAttribute**: A string or the stream attribute in which text stream is included.

  This function returns the entities in the text. If you specify group successive matches as `true`, the result aggregates successive words of the same entity type.

Example

```python
findNameEntityTypeViaDictionary("PERSON",
    "dictionary.xml",
    text)
```

In the above example, if the text attribute contains "Bill Gates donates £31million to fight Ebola", and the dictionary consists of the above entries (i.e. entries of the example in the Description), the result is "Bill".

---

**Find Relationship By Verb function**

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;string&gt; text, &lt;string&gt; subject, &lt;string&gt; object, &lt;string&gt; verb nlp:findRelationshipByVerb (&lt;string&gt; verb, &lt;string-variable&gt;)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
</tbody>
</table>
### Description

`findRelationshipByVerb` takes in a user specified string constant as a verb and a text stream, and returns the whole text, subject, object and the verb based on the specified verb. This information can be extracted only if the verb specified exists in the text stream. However, the tense of the verb does not have to match.

The input parameters used are as follows.

- **verb**: This is a user specified string constant.
- **string-variable**: A string or the stream attribute which includes the text stream.

### Examples

```java
findRelationshipByVerb("say", "Information just reaching us says another Liberian With Ebola Arrested At Lagos Airport") returns the following.
```

- The whole text
- **Information** as the subject
- **Liberian** as the object.
- **says** as the verb.

### Find Relationship By Regex function

#### Syntax

```java
<string> text, <string> subject, <string> object, <string> verb nlp:findRelationshipByRegex (<string> regex, <string> string-variable)
```

#### Extension Type

Function

#### Description

This function returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrex pattern.

#### Example

```java
findRelationshipByRegex('{}=verb /nsubj|agent/ {}=subject >/dobj/ {}=object', "gates foundation donates $50M in support of #Ebola relief") returns the following.
```

- The whole text
- "foundation" as the subject
- "$" as the object
- "donates" as the verb

### Find Semgrex Pattern function
### findSemgrexPattern Function

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
</table>

**Description**

The `findSemgrexPattern` function returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrex pattern.

This function uses the following input parameters.

- **regex**: A user specified regular expression that matches the Semgrex pattern syntax.
- **string-variable**: A string or the stream attribute which includes the text stream.

**Example**

```plaintext
findSemgrexPattern('{\text{lemma:die}} >/.\text{subj|num.*/=reln } \{\text{diedsubject}}', "Sierra Leone doctor dies of Ebola after failed evacuation."")
```

In this example, the function searches for words with the lemmatization die that are governors on any subject or numeric relation. The dependent is marked as the diedsubject, and the relationship is marked as reln. Thus, the query returns an output stream that has the full match of this expression, i.e. the governing word with lemmatization for die. It also returns the name of the corresponding node for each match it finds.

The following is the list of elements in the output stream.

- The whole text
- dies as the match
- "nsubj" as reln
- doctor as diedsubject

---

### Find Tokens Regex Pattern function

<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
</table>

**Syntax**

```plaintext
<string > text, <string > match, < string > object, < string > verb nlp:findTokensRegexPattern (<string> regex, <string> string-variable )
```
findTokensRegexPattern returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrex pattern. The return also includes the corresponding node in the Semgrex pattern and the corresponding named relation defined in the regular expression for each word/phrase.

This function uses the following input parameters.

- **regex**: A user specified regular expression that matches the Semgrex pattern syntax.
- **string-variable**: A string or the stream attribute which includes the text stream.

**Example**

```java
findTokensRegexPattern('([ ner:/PERSON|ORGANIZATION|LOCATION/]+) (?:[]* [lemma:donate]) ([ ner:MONEY]+)', text) defines three groups:

- The first group looks for words that are entities of either PERSON, ORGANIZATION or LOCATION with one or more successive words matching same.
- The middle group is defined as the non capturing group.
- Third looks for one or more successive entities of type MONEY.

This function returns the following.

- The whole text
- "Paul Allen donates $ 9million" as the match.
- "Paul Allen", as group_1.
- "$ 9million" as group_2.
```

**Map Extension**

This extension provides the capability to send a map object inside Siddhi stream definitions and use it inside queries. The following are the functions of the map extension.

- Create function
- Get function
- Is Map function
- Put function
- Remove function
- Create from JSON function
- Create from XML function
- To JSON function
- To XML function

**Create function**
### Syntax

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Object&gt;</code> <strong>map:create</strong>()</td>
<td>or</td>
<td><code>&lt;Object&gt;</code> <strong>map:create</strong>(Object key1, Object value1, Object key2, Object value2,...Object keyN, Object valueN)</td>
</tr>
</tbody>
</table>

**Extension Type** Function

**Description** Returns the created map object.

**Examples**

- `create()` returns an empty map.
- `create(1, "one", 2, "two", 3, "three")` returns a map with keys 1, 2, 3 and corresponding values "one", "two", "three".

---

### Get function

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Object&gt;</code> <strong>map:get</strong>(Map map, Object key)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Extension Type** Function

**Description** Returns the value object from the map that is related to the given key.

**Example**

```
get(company,1) returns the value that is related to the key 1 from the map named company.
```

---

### Is Map function

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt;</code> <strong>map:isMap</strong>(Object object)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Extension Type** Function

**Description** Returns `true` if the object is a map or `false` otherwise.

**Example**

```
isMap(students) returns true if the students object is a map. It returns false if the students object is not a map.
```

---

### Put function

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Object&gt;</code> <strong>map:put</strong>(Object map, Object key, Object value)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Extension Type** Function
<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the updated map after adding the given key-value pair.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td><code>put(students, 1234, &quot;sam&quot;)</code> returns the updated map named <code>students</code> after adding the object &quot;sam&quot; with key 1234.</td>
</tr>
</tbody>
</table>

**Remove function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;Object&gt; map:remove(&lt;Object&gt; map, &lt;Object&gt; key)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the updated map after removing the element with key.</td>
</tr>
<tr>
<td>Example</td>
<td><code>remove(students, 1234)</code> returns the updated map <code>students</code> after removing the element with the key 1234.</td>
</tr>
</tbody>
</table>

**Create from JSON function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;Object&gt; map:createFromJSON(&lt;string&gt; JSONstring)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the map created with the key values pairs given in the <code>JSONstring</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>createFromJSON(&quot;{'symbol' : 'IBM' , 'price' : 200, 'volume' : 100}&quot;&quot;)</code> returns a map with the keys &quot;symbol&quot;, &quot;price&quot;, &quot;volume&quot;, and with the values &quot;IBM&quot;, 200 and 100 respectively.</td>
</tr>
</tbody>
</table>

**Create from XML function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>&lt;Object&gt; map:createFromXML(&lt;string&gt; XMLstring)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the map created with the key values pairs given in the <code>XMLstring</code>.</td>
</tr>
<tr>
<td>Example</td>
<td><code>createFromXML(&quot;&lt;company&gt; &lt;symbol&gt; wso2 &lt;/symbol&gt; &lt;price&gt; &lt;100&gt; &lt;/price&gt; &lt;volume&gt; 200 &lt;/volume&gt; &lt;/company&gt;&quot;</code>) returns a map with the keys &quot;symbol&quot;, &quot;price&quot;, &quot;volume&quot;, and with the values WSO2, 100 and 200 respectively.</td>
</tr>
</tbody>
</table>

**To JSON function**

<p>| Syntax | <code>&lt;String&gt; map:toJSON(&lt;Object&gt; map)</code> |</p>
<table>
<thead>
<tr>
<th>Extension Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Converts a map into a JSON object and returns the definition of that JSON object as a string.</td>
</tr>
<tr>
<td>Example</td>
<td>If &quot;company&quot; is a map with key value pairs (&quot;symbol&quot; : wso2), (&quot;volume&quot; : 100), and (&quot;price&quot;, 200), toJSON(company) returns the string &quot;{&quot;symbol&quot; : &quot;wso2&quot;, &quot;volume&quot; : 100, &quot;price&quot; : 200}&quot;.</td>
</tr>
</tbody>
</table>

**To XML function**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;String&gt; map:toXML(&lt;Object&gt; map) or &lt;String&gt; map:toXML(&lt;Object&gt; map, &lt;String&gt; rootElementName)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the map as an XML string.</td>
</tr>
</tbody>
</table>
| Example | If "company" is a map with key value pairs ("symbol" : wso2), ("volume" : 100), and ("price" : 200),

  toXML(company) returns the string "<symbol>wso2</symbol><volume><100></volume><price>200</price>".

  toXML(company, "abcCompany") returns the string "<abcCompany><symbol>wso2</symbol><volume><100></volume><price>200</price></abcCompany>". |

**Reorder Extension**

Reorder extension is implemented using the K-Slack algorithm. The K-Slack Siddhi extension is used for reordering events from an unordered event stream. The following is an example of a query with the reorder extension.

```sql
@info(name = 'query1')
from inputStream#reorder:kslack(eventTimestamp)
select eventTimestamp, price, volume
insert into outputStream;
```

There are several important variations of the K-slack API of which the details are described below.
## K-Slack function

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>reorder:kslack(&lt;long&gt; timestamp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>StreamProcessor</td>
</tr>
<tr>
<td>Description</td>
<td>This is the most basic version. The events are sorted by the timestamp parameter.</td>
</tr>
</tbody>
</table>

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>reorder:kslack(&lt;long&gt; timestamp, &lt;long&gt; timeOut )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>StreamProcessor</td>
</tr>
<tr>
<td>Description</td>
<td>The second argument shown in the above syntax corresponds to a fixed time-out value set at the beginning of the process. Once the time-out value expires, the extension drains all the events that are buffered within the reorder extension to outside. The time out has been implemented internally using a timer. The events buffered within the extension are released each time the timer ticks.</td>
</tr>
</tbody>
</table>

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>reorder:kslack(&lt;long&gt; timestamp, &lt;long&gt; timeOut, &lt;long&gt; maxValue )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>StreamProcessor</td>
</tr>
<tr>
<td>Description</td>
<td>The third argument in the above syntax is the maximum value for K. This is the amount to which the K value of the K-Slack algorithm will be increased.</td>
</tr>
</tbody>
</table>

### Example

- kslack(timestamp, -1l, 1000000)
  
  In the above example, the algorithm execution starts when K=0 and it gets increased up to 1000000. The value of the K-slack does not increase from that point onwards. Hence, this leads to lower latency compared to the version shown in the first (i.e., single parameter) example of this list. Note that the second argument is set to -1l which effectively disables the timer based draining of the internal buffer.

- kslack(timestamp, 10001, 1000000)
  
  The above is another variation of the third category. Here, a time-out value is specified for the second argument (i.e. 1000 ms). In this case, the K-slack algorithm buffers events until the 1000ms time period expires. The maximum K value is 1000000. The K-value cannot exceed the specified amount.

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>reorder:kslack (&lt;long&gt; timestamp, &lt;long&gt; timeOut, &lt;bool&gt; expireFlag )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>StreamProcessor</td>
</tr>
<tr>
<td>Description</td>
<td>The fourth argument in the above syntax is a flag that indicates whether the out-of-order events that appear after the expiration of the K-slack window should be discarded or not.</td>
</tr>
</tbody>
</table>

### Example

kslack(timestamp, -1l, true)
Markov Models Extension

The Markov Models extension allows abnormal patterns relating to user activity to be detected when carrying out real time analysis. There are two approaches for using this extension. Click on the relevant tab for detailed information about the required approach.

- **Using an existing matrix**
- **Building a new matrix**

You can input an existing Markov matrix as a csv file. It should be a N x N matrix, and the first row should include state names as shown in the following samples. The rows below that indicate the transition probabilities/transition counts for all the possible state transitions.

```
testState01,testState02,testState03
0.1,0.6,0.3
0.3,0.5,0.2
0.6,0.3,0.1
```

```
testState01,testState02,testState03
2,12,6
6,10,4
12,6,2
```

**Syntax**

The following is the syntax for a query with the Markov Models extension using an existing matrix.

```
markov:markovChain(<String> id, <String> state, <int|long|time> durationToKeep, <double> alertThreshold, <String> markovMatrixStorageLocation, <boolean> train)
```

**Input parameters**

The following are the input parameters for this extension.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Required</td>
<td>The ID of the particular user or object being analyzed.</td>
</tr>
<tr>
<td>state</td>
<td>Required</td>
<td>The current state of the ID.</td>
</tr>
<tr>
<td>durationToKeep</td>
<td>Required</td>
<td>The maximum time duration to be considered for a continuous state change of a particular ID.</td>
</tr>
<tr>
<td>alertThreshold</td>
<td>Required</td>
<td>The alert threshold probability.</td>
</tr>
<tr>
<td>markovMatrixStorageLocation</td>
<td>Required</td>
<td>The location of the CSV file that contains the existing Markov matrix to be used.</td>
</tr>
</tbody>
</table>
train | Optional | If this is set to true, event values are used to train the Markov matrix. If this is set to false, the Markov matrix values remain the same.

Output parameters

The following are the output parameters for this extension.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastState</td>
<td>Last state</td>
<td>The previous state of the particular ID.</td>
</tr>
<tr>
<td>transitionProbability</td>
<td>Transition probability</td>
<td>The transition probability between the previous state and the current state for a particular ID.</td>
</tr>
<tr>
<td>notify</td>
<td>notify</td>
<td>This signifies a notification that indicates that the transition probability is less than or equal to the alert threshold probability.</td>
</tr>
</tbody>
</table>

Example

The following returns notifications to indicate whether a transition probability is less than or equal to 0.2 according to the Markov matrix you have provided.

```scala
define stream InputStream (id string, state string);
from InputStream#markov:markovChain(id, state, 60 min, 0.2, "markovMatrixStorageLocation", false)
select id, lastState, state, transitionProbability, notify
insert into OutputStream;
```

This approach involves using a reasonable amount of incoming data to train a Markov matrix and then using it to create notifications.

Syntax

The following is the syntax for a query with the Markov Models extension using a matrix newly built using incoming data.

```scala
markov:markovChain(<String> id, <String> state, <int|long|time> durationToKeep, <double> alertThreshold, <int|long> notificationsHoldLimit, <boolean> train)
```

Input parameters

The following are the input parameters for this extension.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
id  Required  The ID of the particular user or object being analyzed.
state  Required  The current state of the ID.
durationToKeep  Required  The maximum time duration to be considered for a continuous state change of a particular ID.
alertThreshold  Required  The alert threshold probability.
notificationsHoldLimitor  Required  The number of events that should be received before the matrix starts triggering notifications.
train  Optional  If this is set to true, event values are used to train the Markov matrix. If this is set to false, the Markov matrix values remain the same.

Output parameters
The following are the output parameters for this extension.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastState</td>
<td>Last state</td>
<td>The previous state of the particular ID.</td>
</tr>
<tr>
<td>transitionProbability</td>
<td>Transition probability</td>
<td>The transition probability between the previous state and the current state for a particular ID.</td>
</tr>
<tr>
<td>notify</td>
<td>notify</td>
<td>This signifies a notification that indicates that the transition probability is less than or equal to the alert threshold probability.</td>
</tr>
</tbody>
</table>

Example
The following query returns notifications that indicate whether a transition probability is less than or equal to 0.1 according to the Markov matrix that is build using incoming data itself. This starts sending notifications after the first 500 events arrive.

```sql
define stream InputStream (id string, state string, train bool);
from InputStream#markov:markovChain(id, state, 60 min, 0.1, 500, train)
select id, lastState, state, transitionProbability, notify
insert into OutputStream;
```

PMML Based Predictive Analytics Extension
This extension adds PMML based predictive analytic model compliance to Siddhi. It allows you to make predictions based on a predictive analytic model. Supported functions of the PMML extension are as follows.

- Predict function

Predict function
### Syntax

```
< double | float | long | int | string | boolean > pmml:predict(<string> pathToPmmlFile)
```

### Extension Type

Stream Processor

### Description

Processes the input stream attributes according to the defined PMML standard model and outputs the processed results together with the input stream attributes.

This function uses the following input parameter.

- **pathToPmmlFile**: The path to the PMML model file.

The function returns the outputs defined in the output fields. The number of outputs can vary.

### Example

```
predict ('<CEP HOME>/samples/artifacts/0301/decision-tree.pmml')
```

This model is implemented to detect network intruders. The input event stream is processed by the execution plan that uses the pmml predictive model to detect whether a particular user is an intruder to the network or not. The output stream contains the processed query results that include the predicted responses together with the feature values extracted from the input event stream.

### Syntax

```
< double | float | long | int | string | boolean > pmml:predict(<string> pathToPmmlFile, <double|float|long|int|string|boolean> input)
```

### Extension Type

Stream Processor
### Description

Processes the input stream attributes according to the defined PMML standards model and outputs the processed results.

This function uses the following input parameters.

- **pathToPmmlFile**: The path to the PMML model file.
- **input**: An attribute of the input stream that is sent to the PMML standard model as a value to based on which the prediction is made. The `predict` function does not accept any constant values as input parameters. You can have multiple input parameters according to the input stream definition.

This function returns the processed outputs defined in the query. The number of outputs can vary depending on the query definition.

### Examples

```python
predict ('<CEP HOME>/samples/artifacts/0301/decision-tree.pmml', root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate)
```

This model is implemented to detect network intruders. The input event stream is processed by the execution plan that uses the pmml predictive model to detect whether a particular user is an intruder to the network or not. The output stream contains the processed query results that include the predicted responses.

### Machine Learning Extension

This extension provides Siddhi the capability to make predictions based on Machine Learning models. Supported functions of the ML extension are as follows.

- **Predict function**

**Predict function**

- **Syntax**
  ```java
  <double|float|long|int|string|boolean> ml : predict (<string> pathToMLModel, <string> dataT ype)
  ```

- **Extension Type**
  Stream Processor
### Description

Returns an output event with the additional attribute that has the response variable name of the model, set with the predicted value, using the feature values extracted from the input event.

This function uses the following input parameters.

- **pathToMLModel**: The file path or the registry path to the location of the ML model. If the model storage location is **registry**, the value of this parameter should have **registry** as the prefix.

- **dataType**: The data type of the predicted value (**double**, **float**, **long**, **integer/int**, **string**, **boolean/bool**).

### Example

```java
predict('registry:/_system/governance/mlmodels/indian-diabetes-model')
```

### Syntax

```java
<double|float|long|int|string|boolean> ml : predict(<string> pathToMLModel, <string> dataType, <double> input)
```

### Extension Type

Stream Processor

### Description

Returns an output event with the additional attribute that has the response variable name of the model, set with the predicted value, using the feature values extracted from the input event.

This function uses the following input parameters.

- **pathToMLModel**: The file path or the registry path to the location of the ML model. If the model storage location is **registry**, the value of this parameter should have **registry** as the prefix.

- **dataType**: The data type of the predicted value (**double**, **float**, **long**, **integer/int**, **string**, **boolean/bool**).

- **input**: An attribute of the input stream that is sent to the ML model as a value to based on which the prediction is made. The predict function does not accept any constant values as input parameters. You can have multiple input parameters.

### Example

```java
predict('registry:/_system/governance/mlmodels/indian-diabetes-model', NumPregnancies, TSFT, DPF, BMI, DBP, PG2, Age, SI2)
```

### Kalman Filter Extension

This extension provides Kalman filtering capabilities to Siddhi. This allows you to detect outliers of input data. Following are the functions of the Kalman Filter extension.

- **Kalman Filter function**
**Kalman Filter function**

This function uses measurements observed over time containing noise and other inaccuracies, and produces estimated values for the current measurement using Kalman algorithms. The parameters used are as follows.

- **measuredValue**: The sequential change in the observed measurement. e.g., 40.695881
- **measuredChangingRate**: The rate at which the measured change is taking place. e.g., The velocity with which the measured value is changed can be 0.003 meters per second.
- **measurementNoiseSD**: The standard deviation of the noise. e.g., 0.01
- **timestamp**: The time stamp of the time at which the measurement was carried out.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double, double&gt; kf:kalmanFilter(&lt;double&gt; measuredValue)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
</tbody>
</table>
| Example | 1st round: kf:kalmanFilter(-74.178444) returns an estimated value of -74.178444.  
2nd round: kf:kalmanFilter(-74.175703) returns an estimated value of -74.1770735  
006853.  
3rd round: kf:kalmanFilter(-74.177872) returns an estimated value of -74.1773396  
670348. |

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double, double&gt; kf:kalmanFilter(&lt;double&gt; measuredValue, &lt;double&gt; measurementNoiseSD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
</tbody>
</table>
| Example | 1st round: kf:kalmanFilter(-74.178444, 0.003) returns an estimated value of -74.178444.  
2nd round: kf:kalmanFilter(-74.175703, 0.003) returns an estimated value of -74.1770735  
0205573.  
3rd round: kf:kalmanFilter(-74.177872, 0.003) returns an estimated value of -74.1773396  
667771. |

<table>
<thead>
<tr>
<th>Syntax</th>
<th>&lt;double, double&gt; kf:kalmanFilter(&lt;double&gt; measuredValue, &lt;double&gt; measuredChangingRate, &lt;double&gt; measurementNoiseSD, &lt;long&gt; timestamp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Type</td>
<td>Function</td>
</tr>
</tbody>
</table>
Using Siddhi as a Library

This section explains how to embed WSO2 Siddhi 3.0 in a Java project. Embedding Siddhi in a Java project allows you to use the Siddhi query language to carry out real time processing on complex events without running a WSO2 CEP server. This is useful when you need to carry out complex event processing in embedded devices in which WSO2 CEP cannot be deployed.

Follow the procedure below to use Siddhi 3.0 as a library.

- **Step 1: Creating a Java project**
- **Step 2: Creating an execution plan runtime**
- **Step 3: Registering a callback**
- **Step 4: Sending events**

**Step 1: Creating a Java project**

- Create a Java project using Maven and include the following dependencies in its `pom.xml` file.

```xml
<dependency>
    <groupId>org.wso2.siddhi</groupId>
    <artifactId>siddhi-core</artifactId>
    <version>3.0.2</version>
</dependency>
<dependency>
    <groupId>org.wso2.siddhi</groupId>
    <artifactId>siddhi-query-api</artifactId>
    <version>3.0.2</version>
</dependency>
<dependency>
    <groupId>org.wso2.siddhi</groupId>
    <artifactId>siddhi-query-compiler</artifactId>
    <version>3.0.2</version>
</dependency>
```

Add the following repository configuration to the same file.
• Create a new Java class in the Maven project.
• Define a stream definition as follows. The stream definition defines the format of the incoming events.

```java
String definition = "@config(async = 'true') define stream cseEventStream (symbol string, price float, volume long);";
```

• Define a Siddhi query as follows.

```java
String query = "@info(name = 'query1') from cseEventStream#window.timeBatch(500) select symbol, sum(price) as price, sum(volume) as volume group by symbol insert into outputStream;";
```

This Siddhi query stores incoming events for 500 milliseconds, groups them by symbol and calculates the sum for price and volume. Then it inserts the results into a stream named outputStream.

**Step 2: Creating an execution plan runtime**

An execution plan is a self contained, valid set of stream definitions and queries. This step involves creating a runtime representation of an execution plan by combining the stream definition and the Siddhi query you created in Step 1.

```java
SiddhiManager siddhiManager = new SiddhiManager();
ExecutionPlanRuntime executionPlanRuntime = siddhiManager.createExecutionPlanRuntime(definition + query);
```

In the above example, definition + query forms the execution plan. The Siddhi Manager parses the
execution plan and provides you with an execution plan runtime. This execution plan runtime is used to add callbacks and input handlers to the execution plan.

**Step 3: Registering a callback**

You can register a callback to the execution plan runtime in order to receive the results once the events are processed. There are two types of callbacks.

- **Query callback**: This subscribes to a query.
- **Stream callback**: This subscribes to an event stream.

In this example, a query callback is added because the Maven project has only one query.

```java
executionPlanRuntime.addCallback("query1", new QueryCallback() {
    @Override
    public void receive(long timeStamp, Event[] inEvents, Event[] removeEvents) {
        EventPrinter.print(timeStamp, inEvents, removeEvents);
    }
});
```

Here, a new query callback is added to a query named `query1`. Once the results are generated, they are sent to the receive method of this callback. An event printer is added inside this callback to print the incoming events for demonstration purposes.

**Step 4: Sending events**

In order to send events from the event stream to the query, you need to obtain an input handler as follows.

```java
InputHandler inputHandler =
    executionPlanRuntime.getInputHandler("cseEventStream");
```

Use the following code to start the execution plan runtime and send events.

```java
executionPlanRuntime.start();

inputHandler.send(new Object[]{"ABC", 700f, 1001});
inputHandler.send(new Object[]{"WSO2", 60.5f, 2001});
inputHandler.send(new Object[]{"DEF", 700f, 1001});
inputHandler.send(new Object[]{"ABC", 700f, 1001});
inputHandler.send(new Object[]{"WSO2", 60.5f, 2001});
inputHandler.send(new Object[]{"DEF", 700f, 1001});
inputHandler.send(new Object[]{"ABC", 700f, 1001});
inputHandler.send(new Object[]{"WSO2", 60.5f, 2001});
inputHandler.send(new Object[]{"DEF", 700f, 1001});

executionPlanRuntime.shutdown();
```

When the events are sent, they are printed by the event printer.
Communicating Results

Once the data is collected and analysed to produce meaningful information, the final step in complex event processing is to communicate this information. The information produced is communicated by presenting it in the following interfaces supported for WSO2 CEP.

- Configuring CEP to Create Alerts
- Visualizing Results

Configuring CEP to Create Alerts

Events can be notified or published to external systems from WSO2 servers using event publishers. Event publishers enable you to manage event publishing and notifications. They allow publishing events via multiple transports in JSON, XML, Map, text, and WSO2Event formats to various endpoints and data stores.

- Event publisher types
- Event publisher configuration
- Creating event publishers
- Enabling statistics for event publishers
- Enabling tracing for event publishers
- Deleting event publishers
- Editing event publishers

Event publisher types

Event publishers publish events via various transport protocols. These transports are implemented as output event adapters. Following are the adapters that are available by default. You can write extensions to support other transports.

- Cassandra Event Publisher
- Email Event Publisher
- HTTP Event Publisher
- JMS Event Publisher
- Kafka Event Publisher
- Logger Event Publisher
- MQTT Event Publisher
- RDBMS Event Publisher
- SMS Event Publisher
- SOAP Event Publisher
- UI EventPublisher
- WebSocket Event Publisher
- WebSocket Local Event Publisher
- WSO2Event Event Publisher

Event publisher configuration

An event publisher configuration has four main sections as follows.
Event publisher configurations are stored in file system as hot deployable artifacts in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as shown in the example below.

```xml
<eventPublisher name="WebSocketEventPublisher" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    <from streamName="testEventStream" version="1.0.0"/>
    <mapping customMapping="disable" type="text"/>
    <to eventAdapterType="websocket">
        <property name="websocket.server.url">ws://localhost:9099</property>
    </to>
</eventPublisher>
```

The above sections of an event publisher configuration are described below.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>The event stream from which the event publisher will fetch the events for publishing.</td>
</tr>
</tbody>
</table>
### To

An output event adapter (transport) configuration that is used to send the events to.

### Adapter properties

Output event adapters contain three types of adapter properties in their configuration as explained below.

- **Static Adapter Properties**: You can add these properties via the management console. You cannot change them based on the event.
- **Dynamic Adapter Properties**: You can add these properties via the management console. You can change them for each event by adding event attributes as follows: `{{ attribute }}`. E.g. `http://localhost:8000/endpoint/{{endpointId}}`
- **Global Adapter Properties**: These properties come from the `<PRODUCT_HOME>/repository/conf/output-event-adapters.xml` file. They are common for all adapters on its kind which was defined during the event publisher creation.

### Mapping configuration

The format of the message that needs to be sent. You can configure custom mappings on the selected format via advanced settings.

---

**Creating event publishers**

You can create event publishers either **using the management console** or **using a configuration file** as explained below.

### Creating publishers using the management console

Follow the steps below to create an event publisher using the management console of WSO2 CEP/DAS.

1. Log in to the management console, and click **Main**.
2. Click **Publishers** in the **Event** menu, and then click **Add Event Publisher**.
3. Enter a name for **Event Publisher Name**. (Do not use spaces between the words in the name of the event publisher.)

4. Select the **Event Source** with the published events.

5. You view the **Stream Attributes** of the selected event source. You cannot edit the attributes of a created event stream in here.

6. Select the output transport to which you want to publish events for the **Output Event Adapter Type**, and enter the **Adapter Properties** accordingly. For instructions on the adapter properties of output transport types, see **Event Publisher Types**.

7. Select the **Message Format** which you want to apply on the published events. WSO2 servers allow users to configure events in XML, JSON, Text, Map, and WSO2Event event formats.

8. Click **Advanced** to define custom output mappings based on the message format you selected, if you want to publish events that do not adhere to the default event formats. For more information on custom input mapping types, see **Publishing Events in Various Event Formats**.

9. Click **Add Event Publisher**, to create the event publisher in the system. When you click **OK** in the pop-up message on successful addition of the event publisher, you view it in the **Available Event**
Creating publishers using a configuration file

Follow the steps below to create an event publisher using a configuration file.

1. Create an XML file with the following event publisher configurations. An event publisher implementation must start with `<eventPublisher>` as the root element.

```xml
<eventPublisher name="EVENT-PUBLISHER-NAME" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    <from streamName="Test Stream" version="1.0.0"/>
    <mapping customMapping="disable" type="text"/>
    <to eventAdapterType="EVENT-ADAPTER-TYPE">
        .................
    </to>
</eventPublisher>
```

The properties of the above configuration are described below.

<table>
<thead>
<tr>
<th>Adapter property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the event publisher</td>
</tr>
<tr>
<td>statistics</td>
<td>Whether monitoring event statistics is enabled for the publisher</td>
</tr>
<tr>
<td>trace</td>
<td>Whether tracing events is enabled for the publisher</td>
</tr>
<tr>
<td>xmlns</td>
<td>XML namespace for event receivers</td>
</tr>
<tr>
<td>streamName</td>
<td>Name of the event stream from which the publisher publishes events.</td>
</tr>
<tr>
<td>version</td>
<td>Version of the event stream from which the publisher publishes events.</td>
</tr>
<tr>
<td>customMapping</td>
<td>Whether a custom mapping is enabled on the receiver.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the enabled custom mapping.</td>
</tr>
</tbody>
</table>
2. Add the XML file to the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory. Since hot deployment is supported in the product, you can simply add/remove event publisher configuration files to deploy/undeploy event publishers to/from the server.

First define the stream to which the publisher is publishing data from to activate the publisher. When receiving WSO2Events, the outgoing stream definition that you select in the advanced input mappings must also be defined, to activate the event publisher. When you click **Inactive Event Publishers** in the **Available Event Publishers** screen, if an event publisher is in the inactive state due to some issue in the configurations, you view a short message specifying the reason why the event publisher is inactive as shown below. A similar message is also printed on the CLI.

After a publisher is successfully added, it gets added to the list of publisher displayed under **Event** in the **Main** menu of the product's management console. Click **Edit** to change its configuration and redeploy it. This opens an XML-based editor allowing you to edit the event adapter configurations from the UI. Do your modifications and click **Update**. You can also delete it, enable/disable statistics or enable/disable tracing on it using the provided options in the UI as described below.

**Enabling statistics for event publishers**

Follow the steps below to enable monitoring statistics of events published by an existing event publisher.

1. Log in to the management console, and click **Main**.
2. Click **Publishers** in the **Event** menu. You view the **Available Event Publishers** list.
3. Click the **Enable Statistics** button of the corresponding event publisher to enable monitoring event statistics for it.

**Enabling tracing for event publishers**

Follow the steps below to enable tracing on events published by an existing event publisher.

1. Log in to the management console, and click **Main**.
2. Click **Publishers** in the **Event** menu. You view the **Available Event Publishers** list.
3. Click the **Enable Tracing** button of the corresponding event publisher to enable event tracing for it.

**Deleting event publishers**

Follow the steps below to delete an existing event publisher.
1. Log in to the management console, and click **Main**.
2. Click **Publishers** in the **Event** menu. You view the **Available Event Publishers** list.
3. Click the **Delete** button of the corresponding event publisher to delete it.

**Editing event publishers**

Follow the steps below to edit an existing event publisher.

1. Log in to the management console, and click **Main**.
2. Click **Publishers** in the **Event** menu. You view the **Available Event Publishers** list.
3. Click the **Edit** button of the corresponding event publisher to edit it. This opens **Edit Event Publishers Configurations** XML editor.
4. After editing, click **Update**, to save the configuration, or click **Reset** to reset the configuration to its original state.

**Event Publisher Types**

Event publishers publish events via various transport protocols. These transports are implemented as output event adapters. Following are the adapters that comes with the server by default. You can write extensions to support other transports.

- Cassandra Event Publisher
- Email Event Publisher
- HTTP Event Publisher
- JMS Event Publisher
- Kafka Event Publisher
- Logger Event Publisher
- MQTT Event Publisher
- RDBMS Event Publisher
- SMS Event Publisher
- SOAP Event Publisher
- UI Event Publisher
- WebSocket Event Publisher
- WebSocket Local Event Publisher
- WSO2Event Event Publisher

**Cassandra Event Publisher**

Cassandra event publisher dumps events in the **map** format to a Cassandra database.

- Creating a Cassandra event publisher
- Related samples

**Creating a Cassandra event publisher**

For instructions on creating a Cassandra event publisher, see **Configuring CEP to Create Alerts**.

Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a Cassandra event publisher using the management console as shown below.
You can also define the respective adapter properties of the event publisher based on the transport type within the <to> element of the event publisher configuration in the <PRODUCT_HOME>/repository/deployment/server/eventpublishers/ directory as follows.

After entering the above adapter properties, select the Message Format which you want to apply on the published events. Also, click Advanced to define custom output mappings based on the Message Format you selected. For more information on custom output mapping types, see Output Mapping Types.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hosts</strong></td>
<td>Hostnames or IP addresses separated by commas</td>
<td>hosts</td>
<td>testhost1,testhost2</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>The Cassandra port. If you do not define this, the default port will be used</td>
<td>port</td>
<td>9160</td>
</tr>
<tr>
<td><strong>User Name</strong></td>
<td>Username for the database</td>
<td>user.name</td>
<td>admin</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Password for the database</td>
<td>password</td>
<td>admin</td>
</tr>
<tr>
<td><strong>Keyspace Name</strong></td>
<td>Cassandra keyspace name</td>
<td>key.space.name</td>
<td>CEP_KS</td>
</tr>
<tr>
<td><strong>Column Family Name</strong></td>
<td>Column family namespace under the defined keyspace</td>
<td>column.family.name</td>
<td>CF_Transactions</td>
</tr>
<tr>
<td><strong>Strategy Class</strong></td>
<td>The strategy of the keyspace. If you do not define this, <code>org.apache.cassandra.locator.SimpleStrategy</code> will be used</td>
<td>strategy.class</td>
<td>SimpleStrategy</td>
</tr>
<tr>
<td><strong>Replication Factor</strong></td>
<td>The replication factor of the keyspace. If you do not define this, 1 will be used.</td>
<td>replication.factor</td>
<td>3</td>
</tr>
<tr>
<td><strong>Indexed Columns</strong></td>
<td>Columns to be indexed, separated by commas. Index of type &quot;KEYS&quot; with the name <code>{keyspaceName}_{columnFamilyName}_{columnKey}_Index</code> will be applied to the columns.</td>
<td>indexed.columns</td>
<td>key</td>
</tr>
</tbody>
</table>

**Related samples**
For more information on *cassandra* event publisher type, see the following sample.

- Sample 0067 - Publishing Map Events via Cassandra Transport

**Email Event Publisher**

Email event publisher is used to publish events in *XML*, *JSON* or *text* formats via email transports.

- Prerequisites
- Creating an email event publisher
- Related samples

**Prerequisites**

Follow the steps below to complete the prerequisites before starting the event publisher configurations.

Edit the email address, username, password and other relevant properties in the `<PRODUCT_HOME>/repository/conf/output-event-adapters.xml` file, to point the mail transport sender which is enabled by default in the product, to a valid SMTP configuration as shown in the example below.

```xml
<adapterConfig type="email">
    <property key="mail.smtp.from">email-address</property>
    <property key="mail.smtp.user">user-name</property>
    <property key="mail.smtp.password">password</property>
    <property key="mail.smtp.host">smtp.gmail.com</property>
    <property key="mail.smtp.port">587</property>
    <property key="mail.smtp.starttls.enable">true</property>
    <property key="mail.smtp.auth">true</property>
</adapterConfig>
```

- In *gmail* account security settings you may have to enable "Allow less secure apps" option in order to connect account to WSO2 products.
- When SMTP is used with SSL, it is required to extract the certificate of the email server and add it to the trust store of WSO2 DAS. For detailed instructions, see Creating New Keystores - Adding the public key to client-truststore.jks.

**Creating an email event publisher**

For instructions on creating an email event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

The following global properties can be set for the Email event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the email type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.
<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail.smtp.from</td>
<td>The email address used by the publisher to publish events.</td>
<td>String</td>
<td><a href="mailto:abcd@gmail.com">abcd@gmail.com</a></td>
</tr>
<tr>
<td>mail.smtp.user</td>
<td>The username used by the publisher to publish events via email.</td>
<td>String</td>
<td>abcd</td>
</tr>
<tr>
<td>mail.smtp.password</td>
<td>The password used by the publisher to publish events via email.</td>
<td>String</td>
<td>xxxx</td>
</tr>
<tr>
<td>mail.smtp.host</td>
<td>The host of the email server.</td>
<td>String</td>
<td>smtp.gmail.com</td>
</tr>
<tr>
<td>mail.smtp.port</td>
<td>The port of the email server.</td>
<td>Integer</td>
<td>587</td>
</tr>
<tr>
<td>mail.smtp.starttls.enable</td>
<td>This property specifies whether STARTTLS encryption is enabled or not. STARTTLS is an extension which enables a plain text connection to be upgraded to an encrypted (SSL or TLS) connection.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>mail.smtp.auth</td>
<td>This property specifies whether SMTP authentication is enabled or not.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td>keepAliveTimeInMillis</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td>jobQueueSize</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

Configuring adapter properties

Specify the **Dynamic Adapter Properties**, when creating an email event publisher using the management console as shown below. For Dynamic properties user can choose to add placeholders({{placeholder}}) so that value of the dynamic properties will be based on event values in run-time. As an example, as the value of 'Email Address' property user can provide {{email_address_attribute}}.

**Dynamic Adapter Properties**
Create a New Event Publisher

After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.

```xml
<eventPublisher name="EmailOuputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">

..................
<to eventAdapterType="email">
  <property name="email.address">user@gmail.com</property>
  <property name="email.type">text/plain</property>
  <property name="email.subject">This is a test mail.</property>
</to>
</eventPublisher>
```
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email Address</td>
<td>Email address of the client. Register the publisher for multiple email IDs’ by separating them with commas.</td>
<td>email.address</td>
<td><a href="mailto:user@gmail.com">user@gmail.com</a></td>
</tr>
<tr>
<td>Subject</td>
<td>Subject of the email to be sent to the defined email address</td>
<td>email.subject</td>
<td>This is a test mail.</td>
</tr>
<tr>
<td>Email Type</td>
<td>The email format to be sent to the defined email address.</td>
<td>email.type</td>
<td>text/plain</td>
</tr>
</tbody>
</table>

If you select text/html for this parameter, the message body should be in valid HTML.

**Related samples**

For more information on email event publisher type, see the following sample.

- Sample 0064 - Publishing Text Events via Email Transport

**HTTP Event Publisher**

HTTP event publisher is used to publish events in XML, JSON or text formats via HTTP and HTTPS transports.

- Creating a HTTP event publisher
- Related samples

**Creating a HTTP event publisher**

For instructions on creating a HTTP event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

The following global properties can be set for the HTTP event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the http type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
</tbody>
</table>
### keepAliveTimeInMillis

The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>20000</td>
</tr>
</tbody>
</table>

### jobQueueSize

The size of the queue that is used to hold events before they are forwarded to the event stream.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

### defaultMaxConnectionsPerHost

The maximum number of connections allowed per host configuration.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>50</td>
</tr>
</tbody>
</table>

### maxTotalConnections

The maximum number of connections allowed overall.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a HTTP event publisher using the management console as shown below.
You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.

After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.
The above adapter properties are described below.

**Static adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy Host</td>
<td>The proxy server host</td>
<td>http.proxy.host</td>
<td>yourhost</td>
</tr>
<tr>
<td>Proxy Port</td>
<td>The proxy server port</td>
<td>http.proxy.port</td>
<td>8080</td>
</tr>
<tr>
<td>HTTP Client Method</td>
<td>The standard HTTP client method</td>
<td>http.client.method</td>
<td>HttpPost</td>
</tr>
</tbody>
</table>

**Dynamic adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>The target HTTP/HTTPS URL</td>
<td>http.url</td>
<td><a href="http://yourhost:8080/service">http://yourhost:8080/service</a> or <a href="https://yourhost:8080/service">https://yourhost:8080/service</a></td>
</tr>
<tr>
<td>Username</td>
<td>HTTP BasicAuth username</td>
<td>http.username</td>
<td>admin</td>
</tr>
<tr>
<td>Password</td>
<td>HTTP BasicAuth password</td>
<td>http.password</td>
<td>admin</td>
</tr>
<tr>
<td>Headers</td>
<td>Custom HTTP headers</td>
<td>http.headers</td>
<td>header1: value1, header2: value2</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on HTTP event publisher type, see the following sample.

- Sample 0062 - Publishing XML, JSON, and Custom Text Events via HTTP Transport

**JMS Event Publisher**
JMS event publishers are used to publish events in **XML**, **JSON**, **map**, and **text** formats via a JMS transport. You can configure any type of JMS event publisher to run with WSO2 CEP/DAS. This section discusses how to configure a few common JMS event publisher types as follows.

### Configuring global properties

The following global properties can be set for the JMS event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the `jms` type. If a global property available by default is removed, the default value of the property is considered.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td>keepAliveTimeInMillis</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td>jobQueueSize</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

Custom properties cannot be added as global properties.

- **ActiveMQ JMS Event Publisher**
- **IBM WebSphere MQ JMS Event Publisher**
- **Qpid JMS Event Publisher**
- **WSO2 Message Broker JMS Event Publisher**

**ActiveMQ JMS Event Publisher**

ActiveMQ JMS event publisher is used to publish events in **map**, **XML**, **JSON**, and **text** formats via JMS transport.

- **Prerequisites**
- **Creating an ActiveMQ JMS event publisher**
- **Related samples**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install **Apache ActiveMQ JMS**.

   This guide uses ActiveMQ versions 5.7.0 or below. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to **Apache ActiveMQ Documentation**.

2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.

3. Refer the `<PRODUCT_HOME>/repository/conf/jndi.properties` file to register a connection factory. For example, if the connection factory JNDI name is `TopicConnectionFactory`, it will point the default
ActiveMQ host to localhost and port to 5672 as shown below. Furthermore, add the topics to be sent to the ActiveMQ broker in the format: \texttt{topic.\{topicName\} = \{topicName\}}

```java
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/test?brokerlist='tcp://localhost:5672'

\texttt{topic.topicMap = topicMap}
\texttt{topic.topicText = topicText}
```

4. Start ActiveMQ, and then start the product.

Creating an ActiveMQ JMS event publisher

For instructions on creating an ActiveMQ JMS event publisher, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the \textbf{Static} and \textbf{Dynamic Adapter Properties}, when creating an ActiveMQ JMS event publisher using the management console as shown below.
Create a New Event Publisher

<table>
<thead>
<tr>
<th>Enter Event Publisher Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name*</td>
</tr>
<tr>
<td>From</td>
</tr>
<tr>
<td>Event Source*</td>
</tr>
<tr>
<td>Stream Attributes</td>
</tr>
<tr>
<td>To</td>
</tr>
<tr>
<td>Output Event Adapter Type*</td>
</tr>
</tbody>
</table>

### Static Adapter Properties
- **JNDI Initial Context Factory Class**
  - org.apache.activemq.jndi.ActiveMQInitialContextFactory
  - JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
- **JNDI Provider URL**
  - tcp://localhost:61616
  - URL of the JNDI provider.
- **Username**
  - jms-user
- **Password**
  - ************
- **Connection Factory JNDI Name**
  - TopicConnectionFactory
  - The JNDI name of the connection factory.
- **Destination Type**
  - topic
  - Type of the destination.
- **Destination**
  - topicMap
  - **Concurrent Publishers**
  - allow
  - Concurrent publishers can yield high throughput, but may result in out-of-order message delivery.

### Dynamic Adapter Properties
- **Header**
  - Define Transport Headers (eg header.name1:header.value1,header.name2:header.value2)

### Mapping Configuration
- **Message Format**
  - text
  - Select the output message format

---

After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
The above adapter properties are described below.

Static adapter properties

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JNDI Initial Context Factory Class</strong></td>
<td>The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface</td>
<td><code>java.naming.factory.initial</code></td>
</tr>
<tr>
<td><strong>JNDI Provider URL</strong></td>
<td>URL of the JNDI provider</td>
<td><code>java.naming.provider.url</code></td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Valid username for the JMS connection</td>
<td><code>transport.jms.UserName</code></td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Valid password for the JMS connection</td>
<td><code>transport.jms.Password</code></td>
</tr>
<tr>
<td><strong>Connection Factory JNDI Name</strong></td>
<td>The JNDI name of the connection factory</td>
<td><code>transport.jms.ConnectionFactoryJNDIN</code></td>
</tr>
<tr>
<td><strong>Destination Type</strong></td>
<td>The sort order for messages that arrive on a specific destination</td>
<td><code>transport.jms.DestinationType</code></td>
</tr>
</tbody>
</table>
1. Destination
The topic or queue to which WSO2 CEP sends messages by publishing.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Define Transport Headers as a valid header name in a header value pair format</td>
<td>transport.jms.Header</td>
<td>header_name1:header_value1,header_name2:header_value2</td>
</tr>
</tbody>
</table>

Related samples

For more information on ActiveMQ event publisher type, see the following sample.

- Sample 0059 - Publishing Map and Text Events via JMS Transport - ActiveMQ

**IBM WebSphere MQ JMS Event Publisher**

IBM WebSphere JMS event publisher is used to publish events in **map**, **XML**, **JSON**, and **text** formats via JMS transport.

- Prerequisites
- Configuring WebSphere MQ
- Configuring WSO2 CEP/DAS
- Creating an IBM WebSphere JMS event publisher

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Start WSO2 CEP/DAS.
2. Download and install WebSphere MQ pack with the latest fixes. For more information on installing, see the IBM documentation.

Configuring WebSphere MQ

Follow the instructions below to configure WebSphere MQ.

Configuring JMSAdmin.conf File

1. Go to the `<WebSphere_MQ_HOME>/java/bin` directory and open the JMSAdmin.config file in a text editor.
2. Comment out the existing `INITIAL_CONTEXT_FACTORY` and add a `INITIAL_CONTEXT_FACTORY` named `com.sun.jndi.fscontext.RefFSContextFactory`.
3. Comment out the default PROVIDER_URL and use a directory path instead. Ensure the directory is created in the file system (e.g., C:\JNDI-Directory).

If there are .bindings files of earlier versions already existing in this folder, delete them. It should typically be an empty folder.

Your JMSAdmin.config file should now look similar to this:

```java
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WMQInitialContextFactory
#
# The following line specifies the URL of the service provider's initial context. It currently refers to an LDAP root context. Examples of a file system URL and WebSphere's JNDI namespace are also shown, commented out.
#
#PROVIDER_URL=ldap://polaris/o=ibm,c=us
PROVIDER_URL=file:/C:/JNDI-Directory
#PROVIDER_URL=iiop://localhost/
#PROVIDER_URL=localhost:1414/SYSTEM.DEF.SVRCONN
......
```

4. Restart the WebSphere MQ service.

Creating the Queue in WebSphere MQ

1. Start IBM WebSphere MQ Explorer and create a new queue manager. Make sure you select make this the default queue manager check box. Leave default values on the other fields.
2. Select the options to **Start Queue Manager**, **Autostart Queue Manager**, and **Create server connection channel**, and then click **Next**.

3. Select the option to create a listener configuration for TCP/IP, and provide a port number (e.g., 1415).

4. Select the created Queue manager and expand its navigation tree. Click **Queues** in the tree and create a new local queue (e.g., localq).

5. Keep the default configurations and click **Finish**.

6. Click **Topics** in the tree view and create a new local topic (e.g., localt).

7. Right-click **Channels** under **Advanced** and select **New > Server-connection Channel**. Provide a name for the channel (e.g., myChannel), and click **Next**.
8. Set the transmission protocol as TCP and click **Finish**. A listener is created and is running on the given port (e.g., 1415). You should be able to view it by clicking the **listeners** icon.

**Generating the .bindings file**

1. Go to the `<WebSphere_MQ_HOME>/java/bin` directory and invoke the IVT app by running the following command:

   ```
   IVTRun.bat -nojndi -client -m mymgr -host localhost -channel mychannel
   ```

2. Create the default set of JNDI bindings by running the following command on the command prompt:

   ```
   IVTSetup.bat
   ```

3. Execute the **IVTRun** tool as follows.

   ```
   IVTRun.bat -url "file:/C:/JNDI-Directory" -icf
   com.sun.jndi.fscontext.RefFSContextFactory
   ```

4. You have now enabled and verified JNDI support. Now go to `C:/JNDI-Directory` to view the .bindings file there.

5. Start the **JMSAdmin** tool by running the jmsadmin.bat file.

6. Modify the JNDI bindings by executing the following commands:

   For queues:

   ```
   ALTER QCF(ivtQCF) TRANSPORT(CLIENT)
   ALTER QCF(ivtQCF) QMGR(mymgr)
   ```

   For topics:
7. In IBM WebSphere MQ Explorer, select JMS Administered Objects from the tree view on the left, and then select Add initial context. Once done, select File system and enter the JNDI directory path. This will bring up all created queues and topics.

You have now set up and configured IBM WebSphere MQ in your environment.

Configuring WSO2 CEP/DAS

Follow the instructions below to configure WSO2 CEP/DAS.

1. If you set up WSO2 CEP/DAS on a different machine from WebSphere MQ, copy C:/JNDI-Directory to that machine. The bindings file allows you to access WebSphere queues from any machine in the network.

2. Copy the following JAR files from the <WebSphere_MQ_HOME>/java/lib directory to the <PRODUCT_HOME>/repository/components/lib/ directory:
   - com.ibm.mq.jms.jar
   - fscontext.jar
   - providerutil.jar
   - com.ibm.mq.jmsqi.jar
   - dhbcore.jar

3. If you are using WebSphere MQ version 6.0 instead of version 7.0, add the following two JAR files. You might not find com.ibm.mq.jmsqi.jar in version 6.0.
   - com.ibm.mq.jar
   - connector.jar

   Optionally, you might have to add the following jars as well.
   - jms.jar
   - jndi.jar
   - jta.jar
   - ldap.jar

4. If you are using WebSphere MQ version 7.1 or later, add the following jars to the <PRODUCT_HOME>/repository/components/dropins/ directory:
   - com.ibm.mq_2.0.0.jar
   - fscontext_1.0.0.jar

   Add the following files to the <PRODUCT_HOME>/repository/components/lib/ directory.
   - jms.jar
   - jta.jar

5. Log in to the JMSAdmin tool and create a queue named bogusq by running the following commands in JMSAdmin shell.

```
DEFINE Q(bogusq) QMGR(mymgr)
ALTER Q(bogusq) QUEUE(localq)
```

`localq` is the queue we created earlier. We use two queues for the queue scenario, and the queue named bogusq is defined as the default destination since we need the default queue (ivtQ) for our proxy service only. If we use ivtQ here, all the services deployed in CEP (XKMS, echo, wso2carbon-sts etc.) will start listening on the same queue.
6. Repeat these steps for the topic scenarios. For example:

   DEFINE T(bogust)
   ALTER T(bogust) TOPIC(localt)

   localt is the topic we created earlier.

7. Configure the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file as follows:

   ```xml
   <transportReceiver name="jms"
   class="org.apache.axis2.transport.jms.JMSListener">
      <parameter name="myTopicConnectionFactory" locked="false">
         <parameter name="java.naming.factory.initial" locked="false">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
         <parameter name="java.naming.provider.url" locked="false">file:/C:/JNDI-Directory</parameter>
         <parameter name="transport.jms.ConnectionFactoryJNDIName" locked="false">ivtQCF</parameter>
         <parameter name="transport.jms.ConnectionFactoryType" locked="false">topic</parameter>
      </parameter>
      <!--parameter name="SQProxyCF" locked="false">
         <parameter name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
         <parameter name="java.naming.provider.url">file:/C:/JNDI-Directory</parameter>
         <parameter name="transport.jms.ConnectionFactoryJNDIName" locked="false">ivtQCF</parameter>
         <parameter name="transport.jms.ConnectionFactoryType" locked="false">queue</parameter>
      </parameter-->
      <parameter name="default" locked="false">
         <parameter name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
         <parameter name="java.naming.provider.url">file:/C:/JNDI-Directory</parameter>
         <parameter name="transport.jms.ConnectionFactoryJNDIName" locked="false">ivtQCF</parameter>
         <parameter name="transport.jms.ConnectionFactoryType" locked="false">queue</parameter>
      </parameter-->
   </transportReceiver>
```
You will comment and uncomment the non-default connection factories depending on which scenario you are running, as described in the next section.

For details on the JMS configuration parameters used in the code segments, see JMS Connection Factory Parameters.

Creating an IBM WebSphere JMS event publisher

For instructions on creating an IBM WebSphere JMS event publisher, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the Static and Dynamic Adapter Properties, when creating an IBM WebSphere JMS event publisher using the management console as shown below.
Create a New Event Publisher

<table>
<thead>
<tr>
<th>Enter Event Publisher Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name*</td>
</tr>
<tr>
<td>From</td>
</tr>
<tr>
<td>Event Source*</td>
</tr>
<tr>
<td>Stream Attributes</td>
</tr>
<tr>
<td>To</td>
</tr>
<tr>
<td>Output Event Adapter Type*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static Adapter Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Initial Context Factory Class*</td>
</tr>
<tr>
<td>JNDI Provider URL*</td>
</tr>
<tr>
<td>Username</td>
</tr>
<tr>
<td>Password</td>
</tr>
<tr>
<td>Connection Factory JNDI Name*</td>
</tr>
<tr>
<td>Destination Type*</td>
</tr>
<tr>
<td>Destination*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Adapter Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
</tr>
<tr>
<td>Mapping Configuration*</td>
</tr>
<tr>
<td>Message Format*</td>
</tr>
</tbody>
</table>

After entering the above adapter properties, select the Message Format that you want to apply on the published events. Also, click Advanced to define custom output mappings based on the Message Format you selected. For more information on custom output mapping types, see Output Mapping Types.

You can also define the respective adapter properties of the event publisher based on the transport type within the <to> element of the event publisher configuration in the <PRODUCT_HOME>/repository/deployment/server/eventpublishers/ directory as follows.
<eventPublisher name="IBMWebSphereOutputEventAdapter"
  statistics="disable" trace="disable"
  xmlns="http://wso2.org/carbon/eventpublisher">
  ................
  <to eventAdapterType="jms">
    <property name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</property>
    <property name="java.naming.provider.url">file:/C:/JNDI-Directory</property>
    <property name="transport.jms.UserName">jms-user</property>
    <property encrypted="true" name="transport.jms.Password">JP4yDiEh6HogOEjJzQQwHaJFIWZ1nJTzaERl4eYrwukNeyp36R+odMKaN9bq4H9jBQsRV+mhcT1wQVnBpEZn4a+SuFuLKh3NhDEgw6R1tZVo8p1D6TUKvSHYEpwSOGKrkOmdaPEOQ0jdhfK3Hnnjxz/MYPYQkrnLK5MIY=</property>
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.Destination">test_topic</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">ivtQCF</property>
  </to>
</eventPublisher>

Static adapter properties

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>The JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td>Username</td>
<td>Valid username for the JMS connection</td>
<td>transport.jms.UserName</td>
</tr>
<tr>
<td>Password</td>
<td>Valid password for the JMS connection</td>
<td>transport.jms.Password</td>
</tr>
<tr>
<td>Connection Factory JNDI Name</td>
<td>The JNDI name of the connection factory</td>
<td>transport.jms.ConnectionFactoryJNDIN</td>
</tr>
<tr>
<td>Destination Type</td>
<td>The sort order for messages that arrive on a specific destination</td>
<td>transport.jms.DestinationType</td>
</tr>
<tr>
<td>Destination</td>
<td>The topic or queue to which WSO2 CEP/DAS sends messages by publishing</td>
<td>transport.jms.Destination</td>
</tr>
</tbody>
</table>

Dynamic adapter properties
Adapter Property | Description | Configuration file property | Example
--- | --- | --- | ---
Header | Define transport headers as a valid header name in a header value pair format | transport.jms.Header | header_name1:header_value1,header_name2:header_value2

**Qpid JMS Event Publisher**

Qpid JMS event publisher is used to publish events in **map**, **XML**, **JSON**, and **text** formats via JMS transport.

- Prerequisites
- Creating a Qpid JMS event publisher
- Related samples

Prerequisites

Follow the steps below to set up the prerequisites before starting the configuration.

1. Install **Qpid JMS Broker** and **Qpid JMS Client**.
2. Configure WSO2 CEP by adding relevant jars to support JMS transport.
3. Register a connection factory in the `<PRODUCT_HOME>/repository/conf/jndi.properties`. For example, if the connection factory JNDI name is **TopicConnectionFactory**, it will point the default Qpid host to localhost and port to 5672 as shown below. Furthermore, add the topics to be sent to the Qpid broker in the format: **topic.{topicName} = {topicName}**

```
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/
default?brokerlist='tcp://localhost:5672'
topic.topicMap = topicMap
topic.topicJSON = topicJSON
```

4. Start Qpid Broker, and then start the product.

Creating a Qpid JMS event publisher

For instructions on creating a Qpid JMS event publisher, see **Configuring CEP to Create Alerts**.

Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a Qpid JMS event publisher using the management console as shown below.
Create a New Event Publisher

<table>
<thead>
<tr>
<th>Enter Event Publisher Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name*</td>
</tr>
<tr>
<td>From</td>
</tr>
<tr>
<td>Event Source*</td>
</tr>
<tr>
<td>Stream Attributes</td>
</tr>
<tr>
<td>To</td>
</tr>
<tr>
<td>Output Event Adapter Type*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static Adapter Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Initial Context Factory Class*</td>
</tr>
<tr>
<td>JNDI Provider URL*</td>
</tr>
<tr>
<td>Username</td>
</tr>
<tr>
<td>Password</td>
</tr>
<tr>
<td>Connection Factory JNDI Name*</td>
</tr>
<tr>
<td>Destination Type*</td>
</tr>
<tr>
<td>Destination*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Adapter Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Format*</td>
</tr>
</tbody>
</table>

After entering the above adapter properties, select the **Message Format** that you want to apply to the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
<eventPublisher name="QpidJMSOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
            .................
            <to eventAdapterType="jms">
                <property name="java.naming.factory.initial">org.apache.qpid.jndi.PropertiesFileInitialContextFactory</property>
                <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
                <property name="transport.jms.UserName">jms-user</property>
                <property name="transport.jms.DestinationType">topic</property>
                <property name="transport.jms.Destination">test_topic</property>
            </to>
        </eventPublisher>

The above adapter properties are described below.

Static adapter properties

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>The JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface</td>
<td>java.naming.factory.initial</td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider</td>
<td>java.naming.provider.url</td>
</tr>
<tr>
<td>Username</td>
<td>Valid username for the JMS connection</td>
<td>transport.jms.UserName</td>
</tr>
<tr>
<td>Password</td>
<td>Valid password for the JMS connection</td>
<td>transport.jms.Password</td>
</tr>
<tr>
<td>Connection Factory JNDI Name</td>
<td>The JNDI name of the connection factory</td>
<td>transport.jms.ConnectionFactoryJNDINName</td>
</tr>
<tr>
<td>Destination Type</td>
<td>The sort order for messages that arrive on a specific destination</td>
<td>transport.jms.DestinationType</td>
</tr>
</tbody>
</table>
### Destination
The topic or queue to which WSO2 CEP/DAS sends messages by publishing.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Define transport headers as a valid header name in a header value pair format</td>
<td>transport.jms.Header</td>
<td>header_name1:header_value1,header_name2:header_value2</td>
</tr>
</tbody>
</table>

**Dynamic adapter properties**

**Related samples**

For more information on Qpid event publisher type, see the following sample.

- Sample 0060 - Publishing Custom Map and JSON Events via JMS Transport - Qpid

**WSO2 Message Broker JMS Event Publisher**

WSO2 Message Broker (MB) JMS output event adapter is used to publish events in **map**, XML, JSON, and text for maps via JMS transport.

- **Prerequisites**
- Creating a WSO2 MB JMS event publisher
- Related samples

**Prerequisites**

Follow the steps below to set up the prerequisites before starting the configuration.

1. Download and install WSO2 Message Broker. For instructions on WSO2 MB, go to [Message Broker documentation](#).

2. Configure WSO2 CEP by adding relevant libraries to support JMS transport.

3. Register a connection factory in the `<PRODUCT_HOME>/repository/conf/jndi.properties` file. For example, if the connection factory JNDI name is `TopicConnectionFactory`, it will point the default WSO2 MB host to localhost and port to 5672 as shown below. Furthermore, add the topics to be sent to the WSO2 MB in the format: **topic.{topicName}** = **{topicName}**

   ```
   connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/carbon?brokerlist='tcp://localhost:5672'
   topic.topicMap = topicMap
   topic.topicXML = topicXML
   ```

4. Start WSO2 MB and start the WSO2 CEP/DAS server with an off-port since the WSO2 MB has started in the default port. For instructions, see [Starting sample CEP configurations and append - DportOffset=1 -Dqpid.dest_syntax=BURL to the command](#).
For instructions on creating a WSO2 MB JMS event publisher, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the **Static and Dynamic Adapter Properties**, when creating a WSO2 MB JMS event publisher using the management console as shown below.

After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.
You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.

```xml
<eventPublisher name="WSO2MBJMSOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
......
<to eventAdapterType="jms">
  <property name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
  <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
  <property name="transport.jms.UserName">jms-user</property>
  <property encrypted="true" name="transport.jms.Password"></property>
  <property name="transport.jms.DestinationType">topic</property>
  <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
</to>
</eventPublisher>
```

The above adapter properties are described below.

**Static adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Initial Context Factory Class</td>
<td>The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface</td>
<td><code>java.naming.factory.initial</code></td>
</tr>
<tr>
<td>JNDI Provider URL</td>
<td>URL of the JNDI provider</td>
<td><code>java.naming.provider.url</code></td>
</tr>
<tr>
<td>Username</td>
<td>Valid username for the JMS connection</td>
<td><code>transport.jms.UserName</code></td>
</tr>
<tr>
<td>Password</td>
<td>Valid password for the JMS connection</td>
<td><code>transport.jms.Password</code></td>
</tr>
<tr>
<td>Connection Factory JNDI Name</td>
<td>The JNDI name of the connection factory</td>
<td><code>transport.jms.ConnectionFactoryJNDIN</code></td>
</tr>
</tbody>
</table>
**Destination Type**
The sort order for messages that arrive on a specific destination

<table>
<thead>
<tr>
<th>Destination</th>
<th>The topic or queue to which WSO2 CEP sends messages by publishing.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>transport.jms.DestinationType</td>
</tr>
<tr>
<td></td>
<td>transport.jms.Destination</td>
</tr>
</tbody>
</table>

Dynamic adapter properties

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Define transport headers as a valid header name in a header value pair format</td>
<td>transport.jms.Header</td>
<td>header_name1:header_value1,header_name2:header_value2</td>
</tr>
</tbody>
</table>

Related samples

For more information on WSO2 Message Broker event publisher type, see the following sample.

- Sample 0061 - Publishing Map and XML Events via JMS Transport - WSO2 MB

**Kafka Event Publisher**

Kafka event publisher is used to send events in **XML**, **text**, and **JSON** formats to a specific Web service location using POST. This feature is donated by Andres Gomez Ferrer. For more information on Apache Kafka, go to Apache Kafka documentation.

- Prerequisites
- Creating a Kafka event publisher
- Related samples

**Prerequisites**

Set up the below prerequisites to start configuring an Apache Kafka event publisher.

1. Download Apache Kafka server.
2. Configure WSO2 CEP by adding relevant jars to support Kafka transport.
3. Start the Apache Kafka server. For more information, see Apache Kafka documentation.

**Creating a Kafka event publisher**

For instructions on creating a Kafka event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

The following global properties can be set for the Kafka event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the kafka type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.
### Property Key | Description | Data Type | Default Value
--- | --- | --- | ---
minThread | The minimum number of threads (including idle threads) that should be available in the thread pool at a given time. | Integer | 8
maxThread | The maximum number of threads (including idle threads) that should be available in the thread pool at a given time. | Integer | 100
keepAliveTimeInMillis | The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine. | Integer | 20000
jobQueueSize | The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine. | Integer | 10000

### Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a Kafka JMS event publisher using the management console as shown below.

#### Create a New Event Publisher

After entering the above adapter properties, select the **Message Format** that you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.
You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.

```xml
<eventPublisher name="KafkaOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    .....................
    <to eventAdapterType="kafka">
        <property name="topic">test_topic</property>
        <property name="optional.configuration">{property_name1:property_value1, property_name2:property_value2}</property>
        <property name="meta.broker.list">{host1:port1,host2:port2}</property>
    </to>
</eventPublisher>
```

The above adapter properties are described below.

**Static adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Broker List</td>
<td>This is for bootstrapping and the producer will only use it for getting metadata. The list can be a subset of brokers.</td>
<td><code>meta.broker.list</code></td>
<td><code>{host1:port1,host2:port2}</code></td>
</tr>
<tr>
<td>Optional Configuration Properties</td>
<td>Define optional configuration properties</td>
<td><code>optional.configuration</code></td>
<td><code>{property_name1:property_value1, property_name2:property_value2}</code></td>
</tr>
</tbody>
</table>

**Dynamic adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Possible Values</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
<td>sensorStream</td>
<td>Name of the Kafka topic to which, input messages are published</td>
<td><code>topic</code></td>
<td><code>test_topic</code></td>
</tr>
</tbody>
</table>

**Related samples**

For more information on kafka event publisher type, see the following sample.

- Sample 0068 - Publishing XML Events via Kafka Transport

**Logger Event Publisher**

The logger event publisher logs the output events in XML, text, and JSON formats.

- Creating a logger event publisher
• Related samples

_Creating a logger event publisher_

For instructions on creating a logger event publisher, see _Configuring CEP to Create Alerts_.

Configuring adapter properties

Specify the **Dynamic Adapter Properties**, when creating a logger event publisher using the management console as shown below.

**Create a New Event Publisher**

After entering the above adapter properties, select the **Message Format** that you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see _Output Mapping Types_.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Identifier</td>
<td>A string of characters to uniquely identify a log entry</td>
<td>uniqueId</td>
<td>log_id</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on logger event publisher type, see the following samples.

- Sample 0051 - Publishing JSON Events via Logger Transport
- Sample 0052 - Publishing Custom JSON Events via Logger Transport
- Sample 0053 - Publishing XML Events via Logger Transport
- Sample 0054 - Publishing Custom XML Events via Logger Transport
- Sample 0055 - Publishing Text Events via Logger Transport
- Sample 0056 - Publishing Custom Text Events via Logger Transport

**MQTT Event Publisher**

MQTT event publisher is used to send events to a MQTT broker based on the configurations you provide. You can configure it with XML, JSON, and text output mapping types.

**Prerequisites**

Follow the steps below before starting the output MQTT event adapter configuration.

1. Configure WSO2 CEP by adding relevant libraries to support MQTT transport.
2. Start the MQTT-supported server.

**Creating a MQTT event publisher**

For instructions on creating a MQTT event publisher, see Configuring CEP to Create Alerts.
Configuring global properties

The following global properties can be set for the MQTT event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the `mqtt` type. If a global property available by default is removed, the default value of the property is considered.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td>keepAliveTimeInMillis</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td>jobQueueSize</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
<tr>
<td>connectionKeepAliveInterval</td>
<td>The time interval in milliseconds at which a check should be carried out to identify inactive threads.</td>
<td>Integer</td>
<td>60</td>
</tr>
</tbody>
</table>

Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a MQTT event publisher using the management console as shown below.
After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Publishing Events in Various Event Formats**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
<eventPublisher name="MQTTOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
  ......................
  <to eventAdapterType="mqtt">
    <property name="topic">sensordata</property>
    <property name="username">mqtt-user</property>
    <property name="qos">1</property>
    <property name="password">mqtt-password</property>
    <property name="clientId">test-client</property>
    <property name="url">tcp://localhost:1883</property>
    <property name="cleanSession">true</property>
  </to>
</eventPublisher>

The above adapter properties are described below.

**Static adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Id</td>
<td>Client identifier is used by the server to identify a client when it reconnects. It is used for durable subscriptions or reliable delivery of messages is required.</td>
<td>clientId</td>
<td>test-client</td>
</tr>
<tr>
<td>Broker Url</td>
<td>MQTT broker URL. The same URL can be used for WSO2 MB when offset=0</td>
<td>url</td>
<td>tcp://localhost:1883</td>
</tr>
<tr>
<td>Username</td>
<td>Username of the broker</td>
<td>username</td>
<td>mqtt-user</td>
</tr>
<tr>
<td>Password</td>
<td>Password of the broker</td>
<td>password</td>
<td>mqtt-password</td>
</tr>
<tr>
<td>Clean Session</td>
<td>Whether to persist topic subscriptions and acknowledge positions across client sessions.</td>
<td>cleanSession</td>
<td>true/false</td>
</tr>
<tr>
<td>Quality of Service</td>
<td>Quality of service for delivering messages between clients and servers. There are three QoS levels in MQTT as follows.</td>
<td>qos</td>
<td>0,1,2</td>
</tr>
<tr>
<td></td>
<td>• At most once (0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• At least once (1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Exactly once (2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dynamic adapter properties**

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
</table>
**Topic**
The topic that will be used to send messages to MQTT broker.

| topic | sensordata |

**Related samples**
For more information on mqtt event publisher type, see the following sample.

- Sample 0066 - Publishing JSON Events via MQTT Transport

**RDBMS Event Publisher**
RDBMS event publisher is used to publish events in **map** format to a RDBMS in two execution modes, which are insert and update-insert.

- Prerequisites
- Creating a RDBMS event publisher
- Related samples

**Prerequisites**
Follow the steps below to set up the prerequisites before starting the configurations.

1. Create a datasource to connect to the selected database. For instructions on creating a datasource, see Adding Datasources.

   If selected database is H2, uncomment the following H2 database configurations in the `<PRODUCT_HOME>/repository/config/carbon.xml` file as follows, to browse through the database and see the changes. Keep the other properties of the `H2DatabaseConfiguration` element uncommented.

   ```xml
   <H2DatabaseConfiguration>
   <property name="web"/>
   <property name="webPort">8082</property>
   <property name="webAllowOthers"/>
   </H2DatabaseConfiguration>
   ```

**Creating a RDBMS event publisher**
For instructions on creating a RDBMS event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

You can change the queries used to perform the standard database operations by adding the customised queries in the `<CEP_HOME>/repository/conf/output-event-adapters.xml` file. This enables you to use RDBMS database types that use different queries. Customised values can be defined for the following used in standard queries.

Custom properties cannot be added as global properties.
<table>
<thead>
<tr>
<th>Attribute/activity</th>
<th>Current query</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>VARCHAR(255)</td>
</tr>
<tr>
<td>double</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>integer</td>
<td>INT</td>
</tr>
<tr>
<td>long</td>
<td>BIGINT</td>
</tr>
<tr>
<td>float</td>
<td>FLOAT</td>
</tr>
<tr>
<td>createTable</td>
<td>CREATE TABLE $TABLE_NAME ($COLUMN_TYPES)</td>
</tr>
<tr>
<td>insertDataToTable</td>
<td>INSERT INTO $TABLE_NAME ($COLUMNS) VALUES ($VALUES)</td>
</tr>
<tr>
<td>isTableExist</td>
<td>SELECT * FROM $TABLE_NAME limit 1</td>
</tr>
<tr>
<td>updateTableRow</td>
<td>UPDATE $TABLE_NAME SET $COLUMN_VALUES WHERE $CONDITION</td>
</tr>
<tr>
<td>comma</td>
<td>,</td>
</tr>
<tr>
<td>questionMark</td>
<td>?</td>
</tr>
<tr>
<td>equal</td>
<td>=</td>
</tr>
<tr>
<td>and</td>
<td>AND</td>
</tr>
<tr>
<td>selectAllColumnsDataTypeInTable</td>
<td>SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '$TABLE_NAME'</td>
</tr>
<tr>
<td>selectFromTable</td>
<td>SELECT $COLUMNS FROM $TABLE_NAME</td>
</tr>
<tr>
<td>oracle.string</td>
<td>varchar2(255)</td>
</tr>
<tr>
<td>oracle.long</td>
<td>CLOB</td>
</tr>
<tr>
<td>oracle.double</td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>oracle.isTableExist</td>
<td>SELECT * FROM $TABLE_NAME WHERE ROWNUM = 1</td>
</tr>
<tr>
<td>oracle.selectAllColumnsDataTypeInTable</td>
<td>SELECT COLUMN_NAME, DATA_TYPE FROM USER_TAB_COLS WHERE TABLE_NAME = '$TABLE_NAME'</td>
</tr>
<tr>
<td>mysql.string</td>
<td>varchar2(255)</td>
</tr>
<tr>
<td>mysql.isTableExist</td>
<td>SELECT TOP 1 * FROM $TABLE_NAME</td>
</tr>
<tr>
<td>mysql.selectAllColumnsDataTypeInTable</td>
<td>SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '$TABLE_NAME'</td>
</tr>
<tr>
<td>h2.integer</td>
<td>varchar2(255)</td>
</tr>
<tr>
<td>h2.long</td>
<td>REAL</td>
</tr>
</tbody>
</table>
Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a RDBMS event publisher using the management console as shown below.

### Create a New Event Publisher

<table>
<thead>
<tr>
<th>Enter Event Publisher Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name*</td>
</tr>
<tr>
<td>From</td>
</tr>
<tr>
<td>Event Source*</td>
</tr>
<tr>
<td>Stream Attributes</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>To</td>
</tr>
<tr>
<td>Output Event Adapter Type*</td>
</tr>
</tbody>
</table>

#### Static Adapter Properties

| Data Source Name*              | WSO2_CARBON_DB |
| Table Name*                   | test_table |
| Execution Mode*               | insert |
| Composite key columns         | key/group |

#### Mapping Configuration

| Message Format*                | map |
| Advanced                       |     |

After entering the above adapter properties, select the **Message Format** that you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file Property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source Name</td>
<td>Name of the datasource</td>
<td>datasource.name</td>
<td>WSO2_CARBON_DB</td>
</tr>
<tr>
<td>Table Name</td>
<td>Name of the table</td>
<td>table.name</td>
<td>sensordata</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Type of the execution mode</td>
<td>execution.mode</td>
<td>insert/update or insert</td>
</tr>
<tr>
<td>Composite key columns</td>
<td>Attributes used for uniqueness checks for updates. Use commas to separate if you enter more than one attribute.</td>
<td>update.keys</td>
<td>sensor-key,sensor-group</td>
</tr>
</tbody>
</table>

It is required to enter one or more attributes as composite key columns if you select update-or-insert for the Execution Mode property.

Related samples

For more information on rdbms event publisher type, see the following sample.

- Sample 0072 - Publishing Map Events via RDBMS Transport

SMS Event Publisher

SMS event publisher is used to send message notifications via Short Message Peer-to-Peer Protocol (SMPP). It uses Axis2 SMS events when sending SMSs from WSO2 products. SMPP allows Axis2 to connect to a Short Messaging Service Center (SMSC) and send/receive SMSs. SMS event publisher can be configured with XML, text, and JSON output mappings.
Prerequisites

Follow the steps below to complete the prerequisites before starting the event publisher configurations.

1. Add the following configuration under transport senders section in the `<PRODUCT_HOME>/repository/conf/axis2/axis2_client.xml` file, to enable SMS Transport.

```xml
<axisconfigname="AxisJava2.0">
  ...
  <transportSender class="org.apache.axis2.transport.sms.SMSSender" name="sms">
    <parameter name="systemType"></parameter>
    <parameter name="systemId">cep1</parameter>
    <parameter name="password">cep123</parameter>
    <parameter name="host">localhost</parameter>
    <parameter name="port">2775</parameter>
    <parameter name="phoneNumber">CEP1</parameter>
  </transportSender>
  ...
</axisconfig>
```

2. Configure WSO2 CEP by adding relevant jars to support SMS transport.

Creating a SMS event publisher

For instructions on creating a SMS event publisher, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the Dynamic Adapter Properties, when creating a SMS event publisher using the management console as shown below.
Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name*  
SMSOutputEventAdapter  
1. Enter a unique name to identify Event Publisher

From
Event Source*  
Test Stream: 1.0.0
1. The stream of events that need to be published

Stream Attributes

To
Output Event Adapter Type*  
sms
1. Select the type of Adapter to publish events

Dynamic Adapter Properties

Phone No*  
0716453453
1. Phone No where SMS needs to be send (eg: (country-code)(number))

Mapping Configuration

Message Format*  
text
1. Select the output message format

Advanced

Add Event Publisher

After entering the above adapter properties, select the Message Format that you want to apply on the published events. Also, click Advanced to define custom output mappings based on the Message Format you selected. For more information on custom output mapping types, see Output Mapping Types.

You can also define the respective adapter properties of the event publisher based on the transport type within the <to> element of the event publisher configuration in the <PRODUCT_HOME>/repository/deployment/server/eventpublishers/ directory as follows.
<eventPublisher name="SMSOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">

........
<to eventAdapterType="sms">
   <property name="sms.no">0716453453</property>
</to>
</eventPublisher>

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone No</td>
<td>Phone number of the SMS receiver in the following format: [country-code][number]</td>
<td>sms.no</td>
<td>0716453453</td>
</tr>
</tbody>
</table>

**Other post configurations that use SMS event publisher**

Follow the instructions below to set up and configure a SMSC Simulator to receive messages. This guide uses Logica SMSC simulator.

1. Navigate to SMSC Simulator directory. The folder must contain following three files.
   - smpp.jar
   - smscsim.jar
   - users.txt

2. Add the following name-value pairs to users.txt file.

   Enter the value of the **systemId** parameter defined in the above SMS transport sender configuration as the value of the **name** parameter in the below list.

   
   ```
   name=cep1
   password=cep123
   timeout=unlimited
   ```

3. Start SMSC Simulator by executing the following command:
   ```
   java -cp smpp.jar:smcsim.jar com.logica.smscsim.Simulator
   ```

4. In the console where the command runs:
   - Enter 1 for the prompt to start simulation.
   - Enter 2775 as the port number (this port is equal to the port defined in the SMS transport sender configuration.)

When the Starting listener... started log is displayed on the console, the SMSC simulator is ready to accept messages as shown below.
Related samples

For more information on `sms` event publisher type, see the following sample.

- Sample 0065 - Publishing JSON Events via SMS Transport

**SOAP Event Publisher**

SOAP event publisher sends SOAP events in the `XML` format via HTTP, HTTPS and local transports.

- Creating a SOAP event publisher
- Related samples

**Creating a SOAP event publisher**

For instructions on creating a SOAP event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

The following global properties can be set for the SOAP event publisher type in the `<CEP_HOME>/repository/config/input-event-adapters.xml` file. These properties apply to all the publishers of the `soap` type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
</tbody>
</table>
### keepAliveTimeInMillis
The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>20000</td>
</tr>
</tbody>
</table>

### jobQueueSize
The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

### axis2ClientConnectionTimeout
The number of milliseconds allowed to elapse before the Axis2 client connection times out.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

### reuseHTTPClient
If this property is set to true, it is allowed to reuse the connection to the HTTP client for subsequent requests.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
</tr>
</tbody>
</table>

### autoReleaseConnection
If this property is set to true, inactive connections are automatically killed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
</tr>
</tbody>
</table>

### maxConnectionsPerHost
The maximum number of connections allowed per host configuration.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>50</td>
</tr>
</tbody>
</table>

### Configuring adapter properties

Specify the **Dynamic Adapter Properties**, when creating a SOAP event publisher using the management console as shown below.
After entering the above adapter properties, select the **Message Format** which you want to apply on the published events. Also, click **Advanced** to define custom output mappings based on the **Message Format** you selected. For more information on custom output mapping types, see **Output Mapping Types**.

You can also define the respective adapter properties of the event publisher based on the transport type within the `<to>` element of the event publisher configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Url</td>
<td>Destination web service URL</td>
<td>url</td>
<td><a href="http://localhost:9763/services/Axis2LogService/log">http://localhost:9763/services/Axis2LogService/log</a></td>
</tr>
<tr>
<td>User Name</td>
<td>Username token which is required to send event to a HTTPS endpoint.</td>
<td>username</td>
<td>soap-user</td>
</tr>
<tr>
<td>Password</td>
<td>Password token which is required to send event to a HTTPS endpoint.</td>
<td>password</td>
<td>soap-password</td>
</tr>
<tr>
<td>SOAP Headers</td>
<td>Necessary SOAP headers.</td>
<td>soapHeaders</td>
<td>header1: value1, header2: value2</td>
</tr>
<tr>
<td>HTTP Headers</td>
<td>Necessary HTTP headers.</td>
<td>httpHeaders</td>
<td>header1: value1, header2: value2</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on soap event publisher type, see the following sample.

- Sample 0063 - Publishing XML Events via SOAP Transport

**UI Event Publisher**

UI event publisher is an internal event publisher that comes with WSO2 products by default. You can configure it with WSO2Event output mapping types.

- Configuring global properties
Configuring global properties

The following global properties can be set for the UI event publisher type in the `<CEP_HOME>/repository/conf/input-event-adapters.xml` file. These properties apply to all the publishers of the `ui` type. If a global property available by default is removed, the default value of the property is considered.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eventQueueSize</code></td>
<td>The maximum number of events allowed in the adapter queue when the rate at which a UI publisher receives events to be published higher than the rate at which the relevant UI is accepting the events. When the number of events received by the publisher exceeds the value specified for this property, the publisher stops accepting events until the events that are already in the queue get published. Therefore, if you want to reduce system latency, a higher queue size should be specified.</td>
<td>Integer</td>
<td>30</td>
</tr>
<tr>
<td><code>minThread</code></td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td><code>maxThread</code></td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td><code>keepAliveTimeInMillis</code></td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td><code>jobQueueSize</code></td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

Configuring a UI event publisher

A UI event publisher can be configured using the Management Console or a configuration file.

- **Using the Management Console**
- **Using a configuration file**

Follow the procedure below to add a UI event publisher using the Management Console. For detailed information on creating publishers, see [Configuring CEP to Create Alerts](#).

1. Log into the CEP Management Console. In the **Main** tab, click **Publishers** to open the **Available Event Publishers** page.
2. Click **Add Event Publisher** to open the **Create a New Event Publisher** page.
3. Enter a unique name for the UI publisher in the **Event Publisher Name** parameter.
4. Select the relevant stream for the **Event Source** parameter.

An event stream should be defined before you define the publisher. Chart rendering problems could occur in browser if event streams receive events at high throughput.
5. Select **ui** for the **Output Event Adapter Type** parameter.

The following image is an example of a UI event publisher configuration.

### Create a New Event Publisher

**Enter Event Publisher Details**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name*</td>
<td>UIOutputEventAdapter</td>
</tr>
<tr>
<td>From</td>
<td>Test Stream:1.0.0</td>
</tr>
<tr>
<td>Event Source*</td>
<td>sensor id int</td>
</tr>
<tr>
<td>Stream Attributes</td>
<td></td>
</tr>
<tr>
<td>To</td>
<td></td>
</tr>
<tr>
<td>Output Event Adapter Type*</td>
<td>ui</td>
</tr>
<tr>
<td>Mapping Configuration</td>
<td>wso2event</td>
</tr>
<tr>
<td>Message Format*</td>
<td></td>
</tr>
</tbody>
</table>

Continue to add properties as instructed in the **Configuring adapter properties** section.

**Configuring adapter properties**

Follow the procedure below to add properties to a UI event publisher.

1. In the **Mapping Configuration** section, select the relevant message format for the **Message Format** parameter.
2. In the **Output Event Stream** parameter, enter the name of the stream which will be exposed to an external system by the publisher.
3. In the **Output Event Version** parameter, enter the version of the stream you entered in the **Output Event Stream** parameter.
4. Click **Add Event Publisher** to save the changes.

Follow the steps below to configure output UI event publisher using a configuration file.

1. Create an XML file with the following output UI event adapter configurations. Note that `<eventPublisher>` should be the root element.
2. Add the configuration file to the `<CEP_HOME>/repository/deployment/server/eventpublishers/` directory.

Since hot deployment is supported, you can also add/remove output event adapter configuration files to deploy/undeploy output event adapters to/from the server.

Once the publisher is successfully added, it will be displayed in the **Available Event Publishers** page. You can click **Edit** to change its configuration and redeploy it. This opens an XML-based editor allowing you to edit the event adapter configurations from the UI. Make your modifications and click **Update**. You can also delete it, enable/disable statistics or enable/disable tracing on it using the provided options in the UI.

**Related samples**

For more information on UI event publisher type, see the following sample.

- Sample 0071 - Publishing WSO2Event Events via UI Transport

**WebSocket Event Publisher**

The WebSocket event publisher can be configured with **XML**, **JSON**, and **text** output mapping types.

The `websocket` event publisher should be used when the CEP publishes to a websocket server to which the CEP server needs to connect in order to publish events. However, if the events are to be published in a websocket client, that websocket client needs to connect to the inbuilt websocket server of the CEP. In such scenarios, use the **WebSocket Local Event Publisher**.

**Prerequisites**

Start the WebSocket server, before starting the event publisher configurations.

**Creating a WebSocket event publisher**
For instructions on creating a WebSocket event publisher, see Configuring CEP to Create Alerts.

Configuring global properties

The following global properties can be set for the WebSocket event publisher type in the `<CEP_HOME>/repository/conf/output-event-adapters.xml` file. These properties apply to all the publishers of the websocket type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minThread</td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>maxThread</td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td>keepAliveTimeInMillis</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td>jobQueueSize</td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a WebSocket event publisher using the management console as shown below.
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory as follows.
<eventPublisher name="WebSocketOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    ........
    <to eventAdapterType="websocket">
        <property name="websocket.server.url">ws://localhost:9099</property>
    </to>
</eventPublisher>

### Adapter Property

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Socket Server URL</td>
<td>URL of the WebSocket server to which you want to connect</td>
<td>websocket.server.url</td>
<td>ws://localhost:9099</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on websocket event publisher type, see the following sample.

- Sample 0069 - Publishing JSON Events via WebSocket Transport

**WebSocket Local Event Publisher**

WebSocket local event publisher is an internal event publisher that comes with WSO2 products by default. You can configure it with XML, text, and JSON output mapping types.

The websocket-local event publisher should be used when the CEP publishes to a websocket client. WebSocket clients need to connect to the inbuilt websocket server of WSO2 CEP for the events to be published. However, when the CEP publishes to a websocket server, the CEP should connect to the websocket server in order to publish events. In such scenarios, use the WebSocket Event Publisher.

- Creating a WebSocket local event publisher
- Related samples

**Creating a WebSocket local event publisher**

For instructions on creating a WebSocket local event publisher, see Configuring CEP to Create Alerts.

**Configuring global properties**

The following global properties can be set for WebSocket local event publisher type in the `<CEP_HOME>/repository/conf/output-event-adapters.xml` file. These properties apply to all the publishers of the websocket-local type. If a global property available by default is removed, the default value of the property is considered.

Custom properties cannot be added as global properties.

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Description</th>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
</table>
### Configuring adapter properties

There are no adapter-specific properties for the WebSocket local event publisher as shown below.

#### Create a New Event Publisher

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>minThread</strong></td>
<td>The minimum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td><strong>maxThread</strong></td>
<td>The maximum number of threads (including idle threads) that should be available in the thread pool at a given time.</td>
<td>Integer</td>
<td>100</td>
</tr>
<tr>
<td><strong>keepAliveTimeInMillis</strong></td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>20000</td>
</tr>
<tr>
<td><strong>jobQueueSize</strong></td>
<td>The maximum number of milliseconds that idle threads should be kept alive when the total number of threads in the pool exceeds the number of cores in the machine.</td>
<td>Integer</td>
<td>10000</td>
</tr>
</tbody>
</table>

When multi-tenancy is used, the URL formats used to publish events are as follows.

<table>
<thead>
<tr>
<th>Super-tenant/Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WSO2 Event Publisher

WSO2Event event publisher handles WSO2 events. It sends WSO2 events over Thrift using TCP, SSL/TCP, HTTP, and HTTPS protocols to any external server, which can receive them.

- Creating a WSO2Event event publisher
- Related samples

Creating a WSO2Event event publisher

For instructions on creating a WSO2Event event publisher, see Configuring CEP to Create Alerts.

Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a WSO2Event event publisher using the management console as shown below.

### Related samples

For more information on websocket-local event publisher type, see the following sample.

- **Sample 0070 - Publishing JSON Events via Websocket-Local Output Event Adapter**

<table>
<thead>
<tr>
<th>Super-tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;&lt;CEP_Server_Port&gt;&quot;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;&lt;CEP_SSL_Server_Port&gt;&quot;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;&lt;CEP_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;&lt;CEP_SSL_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
</tbody>
</table>

Super-tenant/Tenant URL Formats

<table>
<thead>
<tr>
<th>Super-tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;9763&quot;/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;9443&quot;/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;9763&quot;/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;9443&quot;/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
</tbody>
</table>

e.g., If the publisher name is `WebSocketLocalOutputEventPublisher` and the tenant domain is `mycompany.com`, the URL would be as follows when you use the default server ports.

**Related samples**

For more information on websocket-local event publisher type, see the following sample.

- **Sample 0070 - Publishing JSON Events via Websocket-Local Output Event Adapter**

<table>
<thead>
<tr>
<th>Super-tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;&lt;CEP_Server_Port&gt;&quot;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;&lt;CEP_SSL_Server_Port&gt;&quot;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;&lt;CEP_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;&lt;CEP_SSL_Server_Port&gt;&quot;/t/&lt;tenant_domain&gt;/outputwebsocket/&lt;publisher_name&gt;</td>
</tr>
</tbody>
</table>

**e.g.,** If the publisher name is `WebSocketLocalOutputEventPublisher` and the tenant domain is `mycompany.com`, the URL would be as follows when you use the default server ports.

<table>
<thead>
<tr>
<th>Super-tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;9763&quot;/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;9443&quot;/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenant</th>
<th>URL Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ws://localhost:&quot;;9763&quot;/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
<tr>
<td></td>
<td>wss://localhost:&quot;;9443&quot;/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher</td>
</tr>
</tbody>
</table>
Create a New Event Publisher

<table>
<thead>
<tr>
<th><strong>Enter Event Publisher Details</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event Publisher Name</strong></td>
<td>WS2EventOutputEventAdapter</td>
</tr>
<tr>
<td>0. Enter a unique name to identify Event Publisher</td>
<td></td>
</tr>
<tr>
<td><strong>From</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Event Source</strong></td>
<td>Test Stream: 1.0.0</td>
</tr>
<tr>
<td>0. The stream of events that need to be published</td>
<td></td>
</tr>
<tr>
<td><strong>Stream Attributes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>To</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Output Event Adapter Type</strong></td>
<td>wso2event</td>
</tr>
<tr>
<td>0. Select the type of Adapter to publish events</td>
<td></td>
</tr>
</tbody>
</table>

| **Static Adapter Properties**    |  |
| **Receiver URL**                 | tcp://localhost:7661 |
| 0. Enter the Receiver Url        |  |
| **Authenticator URL**            | tcp://auth-host:7661 |
| 0. Enter the Authenticator Url   |  |
| **User Name**                    | wso2event-user |
| 0. Enter the UserName            |  |
| **Password**                     | ************** |
| 0. Enter the Password            |  |
| **Protocol**                     | thrift |
| 0. The communication protocol that will be used to published events |  |
| **Publishing Mode**              | non-blocking |
| 0. Select the how events should be published |  |
| **Publishing Timeout**           | 0 |
| 0. Timeout for the non-blocking Publishing Mode, default is '0' (fail immediately) |  |

| **Mapping Configuration**        |  |
| **Message Format**               | wso2event |
| 0. Select the output message format |  |

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see **Input Mapping Types**.
You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```xml
<eventPublisher name="WSO2EventOutputEventAdapter" statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    ............
    <to eventAdapterType="wso2event">
        <property name="username">wso2event-user</property>
        <property name="protocol">thrift</property>
        <property name="publishingMode">non-blocking</property>
        <property name="publishTimeout">0</property>
        <property name="receiverURL">tcp://localhost:7661</property>
        <property name="authenticatorURL">tcp://auth-host:7661</property>
        <property encrypted="true" name="password">jkFhzj2US/jSokI/gYjdMpBaGIoacV/XgamNwSPsLqlQ1ALTAlyBUTexgZ8JEiZoz/WL9H5Ncas1Dq/wMbV1Ll0ueUTXoLk63kEf1YWIkoD9ySk0PCFVFWgCsGhH8cAVabECeEP/qh0bFoXTfqYTKjoP2+F1B4EjhDsuc7M=</property>
    </to>
</eventPublisher>
```

The above adapter properties are described below.

<table>
<thead>
<tr>
<th>Adapter Property</th>
<th>Description</th>
<th>Configuration file property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver URL</td>
<td>URL of the target receiver</td>
<td>receiverURL</td>
<td>tcp://localhost:7661</td>
</tr>
<tr>
<td>Authenticator URL</td>
<td>URL of the authenticator</td>
<td>authenticatorURL</td>
<td>tcp://auth-host:7661</td>
</tr>
<tr>
<td>User Name</td>
<td>Username for the listener</td>
<td>username</td>
<td>wso2event-user</td>
</tr>
<tr>
<td>Password</td>
<td>Password for the listener</td>
<td>password</td>
<td>wso2event-password</td>
</tr>
<tr>
<td>Protocol</td>
<td>The communication protocol that will be used to publish events</td>
<td>protocol</td>
<td>thrift/binary</td>
</tr>
<tr>
<td>Publishing Mode</td>
<td>Events publishing mode. Non-blocking refers to asynchronous publishing, and blocking refers to synchronous publishing</td>
<td>publishingMode</td>
<td>non-blocking/blocking</td>
</tr>
<tr>
<td>Publishing Timeout</td>
<td>Positive integer to denote the timeout for the non-blocking publishing mode</td>
<td>publishTimeout</td>
<td>0</td>
</tr>
</tbody>
</table>

**Related samples**

For more information on `wso2event` event publisher type, see the following samples.

- Sample 0501 - Processing a Simple Filter Query with Apache Storm Deployment
- Sample 0057 - Publishing WSO2 Events via WSO2Event Transport
Output Mapping Types

By default, event publishers publish events in XML, JSON, Text, Map (Key-value pairs), and WSO2Event formats. Thereby, if the remote endpoint can process a default format, select the supported default format for Message Format property under Mapping Configuration when creating event publishers.

However, if the remote endpoint cannot process default formats, when creating event publishers select the supported format for Message Format, click the Advanced section and provide output mappings to convert the event to a supported format which the remote endpoint could process.

This section covers the following types of output event publisher mappings that WSO2 CEP/DAS supports and how to configure them:

- WSO2Event output mapping
- XML output mapping
- JSON output mapping
- Text output mapping
- Map output mapping

WSO2Event output mapping

WSO2Event output mapping converts events from one WSO2Event format to another. You need to define both the event stream to retrieve events for publishing and the mapped outgoing event stream for it. If the input event has any arbitrary data, they are mapped to the arbitrary data of the output event.

A sample mapping configuration is shown below.
The configuration XML file of the above sample mapping is as follows.

```xml
<wsotevent:wsoteventprocessor version="1.0">
  <outputEventStream>sensor.stream</outputEventStream>
  <outputEventVersion>1.0.7</outputEventVersion>
  <metaData>
    <name id="id" valueOf="meta_sensorId" />
    <name name="name" valueOf="meta_sensorName" />
  </metaData>
  <correlationData/>
  <payloadData>
    <name humidity="humidity" valueOf="sensorValue" />
    <name value="sensorValue" valueOf="sensorValue" />
  </payloadData>
</wsotevent:wsoteventprocessor>
```
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher xmlns="http://wso2.org/carbon/eventpublisher">
    <mapping customMapping="enable" type="wso2event">
        <to streamName="sensor.stream" version="1.0.7"/>
        <metaData>
            <property>
                <from name="meta_sensorId"/>
                <to name="id" type="int"/>
            </property>
            <property>
                <from name="meta_sensorName"/>
                <to name="name" type="string"/>
            </property>
        </metaData>
        <payloadData>
            <property>
                <from name="humidity"/>
                <to name="humidity" type="float"/>
            </property>
            <property>
                <from name="sensorValue"/>
                <to name="value" type="double"/>
            </property>
        </payloadData>
    </mapping>
</eventPublisher>

Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for WSO2Event Output mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - `null` - CEP will send this event as a WSO2Event with a `null` value for the particular attribute

- **String Attributes**
  - `null` - It will be sent out as a `null` value
  - Empty Attribute - It will be published as an empty string attribute

XML output mapping

XML output mapping converts canonical events of the server in the WSO2Event format to any XML message format that an endpoint can support. Attributes without a `meta_` or a `correlation_` prefix are matched with payload attributes. If they do not match with existing payload attributes, are considered arbitrary data attributes. If you want to use a registry resource for output mapping, see Using Registry Resources for Output Mapping.

A sample mapping configuration is shown below.
The configuration XML file of the above sample mapping is as follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
  <from ... />
  <mapping customMapping="enable" type="xml">
    <inline>
      <sensorData xmlns="">
        <id>{{meta_sensorId}}</id>
        <sensorValue>{{sensorValue}}</sensorValue>
        <humidity>{{humidity}}</humidity>
      </sensorData>
    </inline>
  </mapping>
  <to ... />
</eventPublisher>
```

### Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for XML Output mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
  - null - CEP will convert this to an empty value and send out
  - Empty attribute - CEP will send this as it is as an empty value

- **String Attributes**
  - null - It will be converted to an empty string and sent out
  - Empty Attribute - It will be published as an empty string attribute

### JSON output mapping

JSON output mapping converts canonical event s of the server in the WSO2Event format to any JSON message
WSO2 Complex Event Processor, version 3.1.0

format that an endpoint can support. Attributes without a meta_ or a correlation_ prefix are matched with
payload attributes. If they do not match with existing payload attributes, are considered arbitrary data attributes. If
you want to use a registry resource for output mapping, see Using Registry Resources for Output Mapping.
A sample mapping configuration is shown below.

The configuration XML file of the above sample mapping is as follows.
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
<from ... />
<mapping customMapping="enable" type="json">
<inline>{"sensorData": {
"id": {{meta_sensorId}},
"value": {{sensorValue}},
"humidity": {{humidity}},
"correlation": {"long": {{correlation_longitude}}, "lat":
{{correlation_latitude}}}
}}</inline>
</mapping>
<to ... />
</eventPublisher>

Events with null or empty attributes
In case of an event is being published with null or empty attributes the behaviour for JSON Output mapping
is as follows.
Non-String Attributes (e.g. int, double, long etc.)
null - CEP will send this as a null value
Empty attribute - CEP will send this as it is as an empty value
String Attributes
null - It will be sent out as a null value

Copyright © WSO2 Inc. 2005-2016

569


Text output mapping

Text output mapping converts canonical events of the server in the WSO2Event format to any text message format. Attributes without a `meta_` or a `correlation_` prefix are matched with payload attributes. If they do not match with existing payload attributes, are considered arbitrary data attributes. If you want to use a registry resource for output mapping, see Using Registry Resources for Output Mapping.

A sample mapping configuration is shown below.

The configuration XML file of the above sample mapping is as follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ...
  xmlns="http://wso2.org/carbon/eventpublisher">
  <from ... />
  <mapping 
    customMapping="enable" type="text">
    <inline>
      Sensor Data
      Sensor ID : {{meta_sensorId}}
      Sensor Name : {{meta_sensorName}}
      Sensor located at ({{correlation_longitude}}, {{correlation_latitude}})
      Value : {{sensorValue}}
      Humidity : {{humidity}}
    </inline>
  </mapping>
  <to ... />
</eventPublisher>
```

Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for Text Output mapping is as follows.
**Map output mapping**

Map output mapping converts canonical events of the server in the WSO2Event format to Map message format. A sample mapping configuration is shown below.

**Mapping Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Of</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>meta_sensorId</td>
<td>Delete</td>
</tr>
<tr>
<td>name</td>
<td>meta_sensorName</td>
<td>Delete</td>
</tr>
<tr>
<td>value</td>
<td>sensorValue</td>
<td>Delete</td>
</tr>
</tbody>
</table>

The configuration XML file of the above sample mapping is as follows.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
  <from ... />
  <mapping customMapping="enable" type="map">
    <property>
      <from name="meta_sensorId"/>
      <to name="id"/>
    </property>
    <property>
      <from name="meta_sensorName"/>
      <to name="name"/>
    </property>
    <property>
      <from name="sensorValue"/>
      <to name="value"/>
    </property>
  </mapping>
  <to ... />
</eventPublisher>
```
Using Registry Resources for Output Mapping

WSO2 CEP allows you to map custom content from registry for the XML, JSON and Text mapping types. Custom mapping is stored as a registry resource to be used by the publisher instead of being specified as inline input. WSO2 CEP caches this resource at runtime, and the cache is updated after every cache timeout specified in minutes. If caching is not required, specify the cache timeout as 0.

The following tutorial illustrates how to carry out custom mapping using a registry resource.

1. Log into the WSO2 CEP Management Console.
2. Define an event stream as follows. For detailed instructions to create event streams, see Understanding Event Streams.
1. Enter basic information for the stream as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Stream Name</td>
<td>sensorsteam</td>
</tr>
<tr>
<td>Event Stream Version</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

Add attributes as follows.

<table>
<thead>
<tr>
<th>Attribute Category</th>
<th>Attribute Name</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Data</td>
<td>sensorId</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>sensorName</td>
<td>string</td>
</tr>
</tbody>
</table>
3. Add an output mapping template as a registry resource as follows. For more information about the registry, see Registry.
   1. In the Main tab expand the Registry section and click Browse.
   2. Under Tree View, navigate to /_system/config directory and click Detail View. Click Add Collection. In the Name field, enter Template and click Add. This creates a new sub directory named template in the /_system/config directory.
   3. Navigate to the /_system/config/template sub directory and click Add Resource. Then enter information as shown in the table below, and click Add.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Create Text Content</td>
</tr>
<tr>
<td>Name</td>
<td>message</td>
</tr>
<tr>
<td>Media Type</td>
<td>text/plain</td>
</tr>
<tr>
<td>Content</td>
<td>Welcome Sensor {{meta_sensorName}} reports {{sensorValue}} from</td>
</tr>
<tr>
<td></td>
<td>{{correlation_longitude}}:{{correlation_latitude}}</td>
</tr>
</tbody>
</table>

4. Create a new publisher as follows. For more information, see Configuring CEP to Create Alerts - Creating event publishers.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>samplelogger</td>
</tr>
<tr>
<td>Event Source</td>
<td>sensorstream</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>logger</td>
</tr>
</tbody>
</table>

5. Under Mapping Configuration, select Text for the Message Format field. Then click Advanced. This expands the Create a New Event Publisher page to display the Text Mapping section. Select the Pick from Registry option. In the Registry Path field, navigate to the registry location where you saved the registry resource created in step 3.

In this example, the default value of 15 is left unchanged for the Cache Timeout (in minutes) field.
6. Click **Add Event Publisher**.
7. In the **Tools** tab, click **Event Simulator**. In the **Event Stream Name** field, select `sensorstream`. Enter the following values for the parameters displayed, and then click **Send** to simulate a single event.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensorid</td>
<td>10</td>
</tr>
<tr>
<td>sensorName</td>
<td>Temperature</td>
</tr>
<tr>
<td>language</td>
<td>en</td>
</tr>
<tr>
<td>longitude</td>
<td>79.861256</td>
</tr>
<tr>
<td>latitude</td>
<td>6.927131</td>
</tr>
<tr>
<td>sensorValue</td>
<td>23.0</td>
</tr>
</tbody>
</table>

The output for this event is logged in the terminal as follows.

```
[2016-06-16 17:11:15,398] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: samplelogger,
Event: Welcome
Sensor Temperature reports 23.0 from 79.861256:6.927131
```

**Using custom registry paths**

The registry resource path itself can be parameterized using runtime attribute values. e.g., If the event stream to which the `logger` publisher in the example above is connected has an attribute named `meta_language`, a different registry source is selected depending on the runtime value of the `meta_language` attribute.

The following tutorial illustrates how to change the registry resource path from which a custom output mapping is picked based on the value of an attribute.

1. Log into the CEP Management Console.
2. Create two sub directories in the **Template** directory you created in the previous tutorial, and add a resource to each of these sub directories as described below. For more information about the registry, see [Registry](#).
   1. In the **Main** tab expand the **Registry** section and click **Browse**.
   2. Under **Tree View**, navigate to the `/_system/config/template` sub directory. Create two more sub directories in it named `en` and `fr`.
   3. Navigate to the `/_system/config/template/en` sub directory and click **Add Resource**. Then enter information as shown in the table below, and click **Add**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Create Text content</td>
</tr>
<tr>
<td>Name</td>
<td>message</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Media Type</td>
<td>text/plain</td>
</tr>
</tbody>
</table>
| Content | Welcome  
Sensor {{meta_sensorName}} reports {{sensorValue}} from 
{{correlation_longitude}}:{{correlation_latitude}} |

4. Navigate to the `/system/config/template/fr` sub directory and click Add Resource. Then enter information as shown in the table below, and click Add.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Create Text content</td>
</tr>
<tr>
<td>Name</td>
<td>message</td>
</tr>
<tr>
<td>Media Type</td>
<td>text/plain</td>
</tr>
</tbody>
</table>
| Content | Bienvenue  
Sensor {{meta_sensorName}} reports {{sensorValue}} from 
{{correlation_longitude}}:{{correlation_latitude}} |

After adding both the resources, the registry tree view should look as follows.
3. Delete the publisher named `sample logger` and redefine it as follows. For more information, see Configuring CEP to Create Alerts - Creating event publishers.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>samplelogger</td>
</tr>
<tr>
<td>Event Source</td>
<td>sensorstream</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>logger</td>
</tr>
</tbody>
</table>

4. Under Mapping Configuration, select Text for the Message Format field. Then click Advanced. This expands the Create a New Event Publisher page to display the Text Mapping section. Select the Pick from Registry option. Then enter information as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registry Path</td>
<td>conf:/template/{{meta_language}}/message</td>
</tr>
<tr>
<td>Cache Timeout (in minutes)</td>
<td>15</td>
</tr>
</tbody>
</table>

5. Log into the CEP Management Console.
6. In the **Tools** tab, click **Event Simulator**. In the **Event Stream Name** field, select `sensorstream`. Send three events with the following values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value for Event 1</th>
<th>Value for Event 2</th>
<th>Value for Event 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensorid</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>sensorName</td>
<td>Temperature</td>
<td>Temperature</td>
<td>Temperature</td>
</tr>
<tr>
<td>language</td>
<td>en</td>
<td>fr</td>
<td>ru</td>
</tr>
<tr>
<td>longitude</td>
<td>79.861256</td>
<td>79.861256</td>
<td>79.861256</td>
</tr>
<tr>
<td>latitude</td>
<td>6.927131</td>
<td>6.927131</td>
<td>6.927131</td>
</tr>
<tr>
<td>sensorValue</td>
<td>23.0</td>
<td>23.0</td>
<td>23.0</td>
</tr>
</tbody>
</table>

The three events are logged as follows:

```
2016-08-16 17:08:47,409 ERROR org.wso2.carbon.event.publisher.core.internal.EventPublisher - Cannot send Event: streamId="sensorstream:1.0.0", time=1471347574466, metadata=["Temperature, ru", correlationId=[79.861256, 6.927131], payload=[23.0], arbitraryDataMap=null],
```

Note that an error is returned for the third event. This is because no output mapping was done for the `ru` language.

### Building Custom Event Publishers

In addition to the default publisher types, you can define your own custom publisher. This provides more flexibility to publish events that are sent to WSO2 products. Each event publisher implementation is an OSGI bundle. Therefore, you can easily deploy it as well as undo its deployment on a WSO2 product. To create a custom event publisher, import the `org.wso2.carbon.event.output.adaptor.core` package that contains the skeleton classes/interfaces required for the custom publisher implementation.

#### Implementing the `OutputEventAdapter` interface

The `org.wso2.carbon.event.output.adapter.core.OutputEventAdapter` interface contains the event publisher logic that is used to publish events. You should override the methods given below when implementing your own custom publisher.

- `void init()` throws `OutputEventAdapterFactoryException`

  This method is called when initiating the event publisher bundle. Relevant code segments that are needed when loading OSGI bundle can be included in this method.
• **void testConnect()** throws TestConnectionNotSupportedException, ConnectionUnavailableException  

  This method is used to test the connection of the publishing server.

• **void connect()** throws ConnectionUnavailableException  

  This method can be called to connect to the backend before the events are published.

• **void publish(Object message, Map<String, String> dynamicProperties)** throws ConnectionUnavailableException  

  This method publishes events. It throws the ConnectionUnavailableException if it cannot connect to the backend.

• **void disconnect()**  

  This method is called after the publishing is done, or when the ConnectionUnavailableException is thrown.

• **void destroy()**  

  This method can be used to clean all the resources consumed.

• **boolean isPolled()**  

  This method checks whether events get accumulated at the adapter, and clients connect to it to collect events.

The following is a sample Email publisher implementation of the methods described above.

```java
public class EmailEventAdapter implements OutputEventAdapter {

  @Override
  public void init() throws OutputEventAdapterException {
    tenantId=
    PrivilegedCarbonContext.getThreadLocalCarbonContext().getTenantId();
    //ThreadPoolExecutor will be assigned if it is null.
    if (threadPoolExecutor == null) {
      int minThread;
      int maxThread;
      long defaultKeepAliveTime;
      int jobQueSize;
      //If global properties are available those will be assigned else constant values will be assigned
      if (globalProperties.get(EmailEventAdapterConstants.MIN_THREAD_NAME) != null) {
        minThread =
        Integer.parseInt(globalProperties.get(EmailEventAdapterConstants.MIN_THREAD_NAME));
      } else {
        minThread = EmailEventAdapterConstants.MIN_THREAD;
      }
      if (globalProperties.get(EmailEventAdapterConstants.MAX_THREAD_NAME) != null) {
        maxThread =
        Integer.parseInt(globalProperties.get(EmailEventAdapterConstants.MAX_THREAD_NAME));
      } else {
        maxThread = EmailEventAdapterConstants.MAX_THREAD;
      }
      if (globalProperties.get(EmailEventAdapterConstants.MIN_THREAD_NAME) != null) {
        defaultKeepAliveTime =
        Long.parseLong(globalProperties.get(EmailEventAdapterConstants.MIN_THREAD_NAME));
      } else {
        defaultKeepAliveTime = EmailEventAdapterConstants.MIN_THREAD;
      }
      if (globalProperties.get(EmailEventAdapterConstants.MAX_THREAD_NAME) != null) {
        jobQueSize =
        Integer.parseInt(globalProperties.get(EmailEventAdapterConstants.MAX_THREAD_NAME));
      } else {
        jobQueSize = EmailEventAdapterConstants.MAX_THREAD;
      }
      threadPoolExecutor =
      Executors.newThreadPoolBuilder(minThread, maxThread,
      defaultKeepAliveTime,
      jobQueSize)
```


```java
maxThread = Integer.parseInt(globalProperties.get(EmailEventAdapterConstants.MAX_THREAD_NAME));
} else {
    maxThread = EmailEventAdapterConstants.MAX_THREAD;
}

if (globalProperties.get(EmailEventAdapterConstants.ADAPTER_KEEP_ALIVE_TIME_NAME) != null) {
    defaultKeepAliveTime = Integer.parseInt(globalProperties.get(
        EmailEventAdapterConstants.ADAPTER_KEEP_ALIVE_TIME_NAME));

    if (globalProperties.get(EmailEventAdapterConstants.ADAPTER_EXECUTOR_JOB_QUEUE_SIZE_NAME) != null) {
        jobQueSize = Integer.parseInt(globalProperties.get(
            EmailEventAdapterConstants.ADAPTER_EXECUTOR_JOB_QUEUE_SIZE_NAME));
    }
}

threadPoolExecutor = new ThreadPoolExecutor(minThread,
    maxThread, defaultKeepAliveTime,
    TimeUnit.MILLISECONDS, new
    LinkedBlockingQueue<Runnable>(jobQueSize));

@Override
public void testConnect() throws TestConnectionNotSupportedException {
    throw new TestConnectionNotSupportedException("Test connection is not available");
}

@Override
public void connect() throws ConnectionUnavailableException {
    if (session == null) {
        /**
         * Default SMTP properties for outgoing messages.
         */
        String smtpFrom;
        String smtpHost;
        String smtpPort;
        /**
         * Default from username and password for outgoing messages.
         */
```
final String smtpUsername;
final String smtpPassword;
// initialize SMTP session.
Properties props = new Properties();
props.putAll(globalProperties);
// Verifying default SMTP properties of the SMTP server.
smtpFrom = props.getProperty(MailConstants.MAIL_SMTP_FROM);
smtpHost = props.getProperty(EmailEventAdapterConstants.MAIL_SMTP_HOST);
smtpPort =
props.getProperty(EmailEventAdapterConstants.MAIL_SMTP_PORT);
if (smtpFrom == null) {
    String msg = "failed to connect to the mail server due to null smtpFrom value";
    throw new ConnectionUnavailableException("The adapter " +
eventAdapterConfiguration.getName() + " " + msg);
}
if (smtpHost == null) {
    String msg = "failed to connect to the mail server due to null smtpHost value";
    throw new ConnectionUnavailableException
    ("The adapter " +
eventAdapterConfiguration.getName() + " " + msg);
}
if (smtpPort == null) {
    String msg = "failed to connect to the mail server due to null smtpPort value";
    throw new ConnectionUnavailableException
    ("The adapter " +
eventAdapterConfiguration.getName() + " " + msg);
}
try {
    smtpFromAddress = new InternetAddress(smtpFrom);
} catch (AddressException e) {
    log.error("Error in retrieving smtp address : " +
    smtpFrom, e);
    String msg = "failed to connect to the mail server due to error in retrieving " +
    "smtp from address";
    throw new ConnectionUnavailableException
    ("The adapter " +
eventAdapterConfiguration.getName() + " " + msg, e);
}
// Retrieving username and password of SMTP server.
smtpUsername =
props.getProperty(MailConstants.MAIL_SMTP_USERNAME);
smtpPassword =
props.getProperty(MailConstants.MAIL_SMTP_PASSWORD);
// Initializing SMTP server to create session object.
if (smtpUsername != null && smtpPassword != null) {
    session = Session.getInstance(props, new Authenticator() {
        public PasswordAuthentication
    getPasswordAuthentication() {

return new PasswordAuthentication(smtpUsername, smtpPassword);

}

else {
    log.error("Error in smtp username & password verification");
    String msg = "failed to connect to the mail server due to failed " +
                  "user password authorization";
    throw new ConnectionUnavailableException("The adapter " +
                  eventAdapterConfiguration.getName() + " " + msg);
}

}@Override
public void publish(Object message, Map<String, String> dynamicProperties) {
    //Get subject and emailIds from dynamic properties
    String subject =
            dynamicProperties.get(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_SUBJECT);
    String[] emailIds =
            dynamicProperties.get(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_ADDRESS)
            .replaceAll(" ", "").split(EmailEventAdapterConstants.EMAIL_SEPARATOR);
    String emailType =
            dynamicProperties.get(EmailEventAdapterConstants.APTATER_MESSAGE_EMAIL_TYPE);

    //Send email for each emailId
    for (String email : emailIds) {
        try {
            threadPoolExecutor.submit(new EmailSender(email, subject, message.toString(), emailType));
        } catch (RejectedExecutionException e) {
            EventAdapterUtil.logAndDrop(eventAdapterConfiguration.getName(), message, "Job queue is full", e, log, tenantId);
        }
    }

}@Override
public void disconnect() {
    //not required
}

}@Override
public void destroy() {
    //not required
}
@Override
    public boolean isPolled() {

Implementing the OutputEventAdapterFactory class

The `org.wso2.carbon.event.output.adapter.core.OutputEventAdapterFactory` class can be used as the factory to create your appropriate event publisher type. You should override the methods given below when extending your own custom publisher.

- **public String getType()**
  
  Here, the type needs to be specified. This string is displayed in the publisher interface in the adapter type drop down list.

- **public List<String> getSupportedMessageFormats()**
  
  Here, the supported message formats for the created publisher type need to be specified.

- **public List<Property> getStaticPropertyList()**
  
  Here static properties need to be specified. These properties use the values assigned when creating a publisher. For more information on adapter properties see [Event Publisher Configuration](#).

- **public abstract List<Property> getDynamicPropertyList()**
  
  You can define dynamic properties similar to static properties. The only difference is that dynamic property values can be derived from events handled by publisher. For more information on adapter properties see [Event Publisher Configuration](#).

- **public abstract String getUsageTips()**
  
  Specify any hints to be displayed in the Management Console.

- **public OutputEventAdapter createEventAdapter(OutputEventAdapterConfiguration eventAdapterConfiguration, Map<String, String> globalProperties)**
  
  This method creates the publisher by specifying event adapter configuration and global properties that are common to each adapter type.

The following is a sample Email publisher implementation of the `OutputEventAdapterFactory` class.

```java
public class EmailEventAdapterFactory extends OutputEventAdapterFactory {

    @Override
    public String getType() {
        return EmailEventAdapterConstants.ADAPTER_TYPE_EMAIL;
    }

    @Override
    public List<String> getSupportedMessageFormats() {
        List<String> supportedMessageFormats = new ArrayList<String>();
        supportedMessageFormats.add(MessageType.TEXT);
        supportedMessageFormats.add(MessageType.XML);
        supportedMessageFormats.add(MessageType.JSON);
    }
}
```
return supportedMessageFormats;
}

@Override
public List<Property> getStaticPropertyList() {
    return null;
}

@Override
public List<Property> getDynamicPropertyList() {
    List<Property> dynamicPropertyList = new ArrayList<Property>();
    // set email address
    Property emailAddress = new Property(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_ADDRESS);
    emailAddress.setDisplayName(resourceBundle.getString(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_ADDRESS));
    emailAddress.setRequired(true);
    emailAddress.setHint(resourceBundle.getString(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_ADDRESS_HINT));

    dynamicPropertyList.add(emailAddress);

    // set email subject
    Property subject = new Property(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_SUBJECT);
    subject.setDisplayName(resourceBundle.getString(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_SUBJECT));
    subject.setRequired(true);
    dynamicPropertyList.add(subject);

    // set format of the email
    Property format = new Property(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_TYPE);
    format.setDisplayName(resourceBundle.getString(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_TYPE));
    format.setRequired(false);
    format.setOptions(new String[]{EmailEventAdapterConstants.MAIL_TEXT_PLAIN, EmailEventAdapterConstants.MAIL_TEXT_HTML});
    format.setDefaultValue(EmailEventAdapterConstants.MAIL_TEXT_PLAIN);
    format.setHint(resourceBundle.getString(EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_TYPE_HINT));
    dynamicPropertyList.add(format);

    return dynamicPropertyList;
}

@Override
public String getUsageTips() {


return null;
}

@Override
public OutputEventAdapter
createEventAdapter(OutputEventAdapterConfiguration
eventAdapterConfiguration, Map<String,
String> globalProperties) {
    return new EmailEventAdapter(eventAdapterConfiguration,
Exposing the custom event publisher as an OSGI service

Apart from the above, you can maintain a service class under the `internal\ds\` directory to expose the custom event publisher implementation as an OSGI service. When exposing the service, it needs to be exposed as a service of the `OutputEventAdapterFactory` type. The following is a sample implementation of a service class for a custom defined publisher.

```java
/**
 * @scr.component component.name="output.Email.AdapterService.component"
 * immediate="true"
 */
public class EmailEventAdapterServiceDS {

    private static final Log log =
        LogFactory.getLog(EmailEventAdapterServiceDS.class);

    /**
     * initialize the email service here service here.
     * @param context
     */
    protected void activate(ComponentContext context) {
        try {
            OutputEventAdapterFactory emailEventAdaptorFactory = new
                EmailEventAdapterFactory();
            context.getBundleContext().registerService(OutputEventAdapterFactory.class
                .getName(),
                emailEventAdaptorFactory, null);
            if (log.isDebugEnabled()) {
                log.debug("Successfully deployed the output Email event
                    adaptor service");
            }
        } catch (RuntimeException e) {
            log.error("Can not create the output Email event adaptor
                service ", e);
        }
    }
}
```

Furthermore you can have a utility directory as `internal\util\` where you can place utility classes required for the custom publisher implementation.
Deploying the custom event publisher

Follow the procedure below to deploy a custom event publisher.

1. Implement the custom event publisher type.
2. Build the project.
3. Copy the OSGI bundle that is created inside the target directory into the `<CEP_HOME>/repository/components/dropins` directory.

When you start the CEP server, you can see the newly created event publisher type service in the service startup logs. The newly created custom event publisher type will also be visible in the UI with the relevant properties. Now you can create several instances of this event publisher type.

Visualizing Results

The following sections cover the methods available to visualize the results of data processed by WSO2 CEP.

- Visualizing Results in the Analytics Dashboard
- Geo Dashboard

Visualizing Results in the Analytics Dashboard

Note that this dashboard is not supported with IBM JDK.

Analytics dashboard application is the data visualization component of WSO2 Complex Event Processor. You can create dashboards and real time gadgets event streams. A gadget displays the real-time information processed by WSO2 CEP in a selected format. A dashboard serves as a container for a collection of gadgets organized in a selected layout. The following diagram summarizes how to set up the Analytics Dashboard for WSO2 CEP as per your requirement.

Instructions to set up a dashboard are covered in the following steps.

- Prerequisites
  - Step 1: Log into the Analytics Dashboard
  - Step 2: Create required gadgets
  - Step 3: Create a new dashboard
  - Step 4: View information

Prerequisites

The following table specifies the prerequisites for viewing information in a gadget depending on the service provider...
selected for it.

<table>
<thead>
<tr>
<th>Service Provider</th>
<th>Prerequisites</th>
</tr>
</thead>
</table>
| Relational Database Source | • A relational database with at least one table that has data should exist.  
• The appropriate JDBC driver should be downloaded and saved in the `<DAS_HOME>/repository/conf/lib` directory.                                      |
| Realtime Data Source    | • A real-time event stream should exist in order to get the real-time events based on which information is displayed in the gadget. To define an event stream, see Understanding Event Streams.  
• A UI publisher should exist in order to publish events from the real-time event stream to the Analytics Dashboard. To define a UI Publisher, see UI Event Publisher. |
| REST Data Source        | An accessible REST endpoint should exist.                                                                                                                                                                      |

**Step 1: Log into the Analytics Dashboard**

Follow the procedure below to log into the Analytics Dashboard.

1. Access the WSO2 CEP Analytics Dashboard using the following URL.  
   https://<HOST_NAME>:<PORT>/portal/dashboard
2. Enter your username and password, and click **Login**.

![](Login.png)

You can deploy a dashboard and/or its components (e.g. layouts, gadgets, and widgets) in the Analytics Dashboard of WSO2 CEP by bundling them as artifacts of a Carbon Application (cApp). For instructions on deploying cApps in WSO2 CEP, see Packaging Artifacts as C-App Archive.

**Step 2: Create required gadgets**

Follow the procedure below to create a gadget to display information processed by WSO2 CEP as required.
1. In the Analytics Dashboard, click the menu icon and then click **Gadgets** to open the **Gadgets** page as demonstrated below.

![Gadgets Menu]

2. Click **GENERATE GADGET** to open the **Generate a Gadget** wizard.
3. Enter information as follows to configure a gadget.
1. In the **Select Provider** field, select the relevant service provider based on the source from which data should be taken to update the gadget. The source of each provider type is given in the table below. Then click **Next**. In this example, **Realtime Data Source** is selected to publish data updated in a real-time event stream. Then click **Next**.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Source of Data Published in Gadget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Database Source</td>
<td>A datasource that can connect to an RDBMS database.</td>
</tr>
<tr>
<td>Real-time Data Source</td>
<td>Real-time events received in event streams.</td>
</tr>
<tr>
<td></td>
<td>Here, the events are directly obtained from the backend. In order to do this, a UI event publisher should be defined and connected to the event stream that receives the events.</td>
</tr>
</tbody>
</table>
REST Datasource | A REST endpoint

2. In the **Event Stream** field, select the event stream from which the events to be published are derived. In this example, a stream named `org.wso2.event.sensor.filtered.stream:1.0.0` is selected. Click **Next**.

3. Configure a chart as follows.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadget Name</td>
<td>The text to be displayed as the name of the gadget.</td>
<td>Sensor Value VS Timestamp</td>
</tr>
<tr>
<td>Select Chart Type</td>
<td>The type of chart that you want the gadget to display.</td>
<td>Line Chart</td>
</tr>
<tr>
<td>X-Axis</td>
<td>Select the attribute for the X axis of the line chart.</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>X type</td>
<td>The data type of the item represented on the X axis.</td>
<td>time</td>
</tr>
<tr>
<td>Y-Axis</td>
<td>Select the attribute for the Y axis of the line chart.</td>
<td>sensorValue</td>
</tr>
<tr>
<td>Y type</td>
<td>The data type of the item represented on the Y axis.</td>
<td>default</td>
</tr>
<tr>
<td>Color domain</td>
<td>The points mapped on the line chart will be differentiated using colour based on the value of the attribute selected for this parameter.</td>
<td>sensorName</td>
</tr>
<tr>
<td>Max length</td>
<td>The maximum length of the chart in terms of the number of records displayed at a given time.</td>
<td>30</td>
</tr>
</tbody>
</table>

4. Click **Preview** and then simulate a few events to see the preview of the gadget in the lower section of the page.

In this example, the event stream used receives events via an HTTP receiver. Three events were simulated by issuing the following cURL commands.

```bash
curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524120, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 120 } }}" http://localhost:9763/endpoints/httpReceive --header "Content-Type:application/json"
```

```bash
curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524120, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude": 20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 120 } }}" http://localhost:9763/endpoints/httpReceive --header "Content-Type:application/json"
```
5. Click **Add to Store** to save the gadget configuration, and then click **Go to Portal**. The **Dashboards** page appears again.

- Once a gadget is created and saved, its configuration is saved in JSON format in the `<CEP_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>/fs/gadget` directory.
- For more detailed information about creating and customising gadgets, see [WSO2 Dashboard Server Documentation - Gadget Author Guide](#).

---

**Step 3: Create a new dashboard**

Follow the procedure below to add a new dashboard to the Analytics Dashboard.

1. Log into the Analytics Dashboard if you are not already logged in. For detailed instructions, see **Logging into the Analytics Dashboard**. If there are any existing dashboards, they are displayed as shown in the example below.

2. Click **CREATE DASHBOARD** in the top navigator to open the **CREATE A DASHBOARD** page.

3. Enter information as follows.
1. Enter a name and a description for the dashboard. In this example, the name and the description is entered as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of your Dashboard</td>
<td>Sensor Statistics</td>
</tr>
<tr>
<td>Description</td>
<td>This dashboard indicates the sensor value at different times in a particular location.</td>
</tr>
</tbody>
</table>

2. Select a layout based on how you want the gadgets to be organized on your dashboard. In this example, the Single Column layout is selected. A message appears to indicate that the dashboard is successfully created.

3. Click the icon for gadgets. Drag and drop a gadget displayed into the required column of your dashboard. In this example, the Sensor Value VS Timestamp gadget that was created in Step 2: Create required gadgets is added to the dashboard.

Once information is entered as described above, the Dashboard saved successfully message appears. The new dashboard can be viewed in the Dashboards page.

- Once a dashboard is created and saved, its configuration is saved in JSON format in the registry. This can be accessed via the /_system/config/ues/dashboards/<Dashboard_Name> registry path. For more information about the registry, see Registry.
- For more detailed information about creating and customising dashboards, see WSO2 Dashboard Server Documentation - Editor Guide.
Step 4: View information

Follow the procedure below to view information in a dashboard in the WSO2 CEP Analytics Dashboard.

1. Access the WSO2 CEP Analytics Dashboard using the following URL, and log in using your credentials.
   
   https://<HOST_NAME>:<PORT>/portal/dashboard

2. Click View on the required dashboard. In this example, the dashboard created in Step 3: Create a new dashboard is selected.

3. Simulate some data for the event stream connected to your gadget.

   The dashboard is updated as follows.

   ```
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524120 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": humidity }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 138 } } } "http://localhost:9763/endpoints/httpReceiver
   --header "Content-Type:application/json"
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524121 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.4, "sensorValue": 144 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524125 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": humidity }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.7, "sensorValue": 145 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524131 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 156 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524135 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": humidity }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.9, "sensorValue": 170 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524140 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 165 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524150 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": temperature }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.3, "sensorValue": 168 } } } "http://localhost:9763/endpoints/httpReceiver
   
   curl -X POST -d "{"event": { "metaData": { "timestamp":1439467524155 , "isPowerSaverEnabled":false, "sensorId": 701, "sensorName": humidity }, "correlationData": { "longitude": 4.504343, "latitude":20.44345 }, "payloadData": { "humidity": 2.9, "sensorValue": 171 } } } "http://localhost:9763/endpoints/httpReceiver
   ```
Geo Dashboard

The Geo Dashboard of WSO2 CEP is a Jaggery application that provides realtime information about geo spatial objects. It processes spatial data from an external source of events and analyzes/manipulates this data to produce meaningful information to end users using the geo dashboard. You can interact with it to generate a variety of alerts and warnings as follows.

<table>
<thead>
<tr>
<th>Alert type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed alert</td>
<td>You can specify a maximum speed limit to all spatial objects. If an object exceeds the specified speed, an alert will be generated.</td>
</tr>
<tr>
<td>Proximity alert</td>
<td>You can specify a radius and a time. Thereby, a warning will be populated if two spatial objects arrive near each other within the specified limits.</td>
</tr>
<tr>
<td>Within alert</td>
<td>You can specify a geo area and if any spatial object comes in to the specified area an alert will be generated.</td>
</tr>
<tr>
<td>Stationery alert</td>
<td>You can specify a geo area and a fluctuation radius and a time, which will generate an alert if any spatial object is located in that area during the specified time. Fluctuation radius is used to minimize the fluctuation effect of a spatial object.</td>
</tr>
<tr>
<td>Congestion Alert</td>
<td>You can define a geo area. An alert is generated when the congestion level in that area changes.</td>
</tr>
</tbody>
</table>

- Prerequisites
- Running the Geo Dashboard
- Executing the producer
- Accessing the Geo Dashboard

You can use the Geo Dashboard of WSO2 CEP as described below.

Prerequisites

Set up the following prerequisites before starting the configurations.

1. Install the **GPL - Siddhi Geo Extension** feature. For detailed instructions to install GPL features, see Installing WSO2 GPL Features.
2. Copy the two extension JAR files in the `<CEP_HOME>/samples/cep/utils/geo-dashboard-extensions/` directory to the `<CEP_HOME>/repository/components/lib/` directory.

Running the Geo Dashboard

Navigate to the `<CEP_HOME>/repository/resources/geo-dashboard` directory and copy all the folders into the `<CEP_HOME>/repository/deployment/server` directory.
Executing the producer

Follow the steps below to run the tfl-feed producer client from the command line.

1. Download GeoTools GIS toolkit
2. Copy the following client JAR files from downloaded <GeoTools_HOME> directory to both <CEP_HOME>/repository/components/lib/ directory and <CEP_HOME>/samples/cep/lib directory.
   - gt-geojson-13.1.jar
   - gt-main-13.1.jar
   - gt-metadata-13.1.jar
   - jai_core-1.1.3.jar
   - jai_imageio-1.1.jar
4. Copy jts-1.8.jar from <JTS_HOME>/lib directory to both <CEP_HOME>/repository/components/lib directory and <CEP_HOME>/samples/cep/lib directory.
5. Navigate to <CEP_HOME>/samples/cep/ producers/tfl-feed/ directory, and execute the ant command in a new tab of your CEP console, to execute the producer of the Geo Dashboard application.

Accessing the Geo Dashboard

Follow the steps below to start and access the Geo Dashboard.

1. Start the WSO2 CEP server. For instructions, see Starting CEP.
2. Access the Geo Dashboard application in your Web browser using the following URL: https://localhost:9443/portal/dashboards/geo-dashboard

   ![Geo Dashboard Image]

   This dashboard can be used to define alerts, define geo fences, click on spatial objects to view information relating to them etc.

Packaging Artifacts as C-App Archive

Carbon applications (C-App) deployer is a specific feature of WSO2 CEP, which deploys WSO2 CEP-specific deployable artifacts using a single CAR file.
C-App deployable artifacts and types

You can deploy all deployable artifacts, and also the stream definitions using Carbon apps (C-Apps). Given below is a list of deployable artifacts and the associated types.

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>event receivers</td>
<td>event/receiver</td>
</tr>
<tr>
<td>event publishers</td>
<td>event/publisher</td>
</tr>
<tr>
<td>execution plans</td>
<td>event/execution-plan</td>
</tr>
<tr>
<td>event stream</td>
<td>event/stream</td>
</tr>
</tbody>
</table>

Specific UI restrictions

If the above mentioned artifacts are being deployed using a C-App, you are restricted on editing or deleting them. Therefore, if you need to edit the C-App, you need to re-pack and re-deploy it.

Creating a C-App

Follow the steps below to create a C-App with the following structure.

```
artifacts.xml
WSO2EventReceiver_1.0.0
  artifact.xml
  WSO2EventReceiver-1.0.0.xml
HighFrequentTweetsLogger_1.0.0
  artifact.xml
  HighFrequentTweetsLogger-1.0.0.xml
```

1. Create the top level artifacts.xml file of the C-App which defines the set of folders included in it as shown below.
2. Create separate directories in the top level of the C-App for the two dependencies (WSO2EventReceiver_1.0.0 and HighFrequentTweetsLogger_1.0.0) defined in the artifacts.xml file.

You can have multiple dependencies as required with a directory for each of them in the same level as the artifacts.xml file in the C-App. Include the name and the version of the artifact in the name of the directory.

Each dependency directory should contain an artifact.xml file.

3. Create an artifact.xml file inside all dependency directories. For example, the artifact.xml file of the WSO2EventReceiver_1.0.0 dependency directory is as follows.

4. Create the WSO2EventReceiver-1.0.0.xml file, which is the actual event receiver file that you deploy in WSO2 CEP inside the WSO2EventReceiver_1.0.0 directory.

5. Create the HighFrequentTweetsLogger-1.0.0.xml file, which is the actual event publisher file that you deploy in WSO2 CEP inside the HighFrequentTweetsLogger_1.0.0 directory.

You can also create a C-App using the WSO2 Developer Studio. For more information, see Creating an Analytics Project.

Deploying a C-App

Follow the steps below to deploy a C-App.

1. Log in to WSO2 CEP management console using admin/admin credentials.
2. Click Main, and then click Add in the Carbon Applications menu.
3. Click Choose File as shown below.
4. Select the CAR file you need to upload as shown below.

5. Click **Upload**.

6. Click **Main**, and then click **List** in the **Carbon Applications** menu. You see the file added to the list of deployed CAR files as shown below.

   If there is a deployment failure, the faulty Carbon applications and the deployment error will be displayed in the UI.
Undeploying a C-App

Follow the steps below to undeploy a C-App.

1. Log in to WSO2 CEP management console using admin/admin credentials.
2. Click Main, and then click List in the Carbon Applications menu.
3. Click the corresponding Delete link of the relevant Carbon application, which you want to delete as shown below.

Monitoring and Debugging

This section explains how you can debug WSO2 CEP.

- Troubleshooting and debugging guide
  - Step 1: Assess the Event Flow
  - Step 2: Inspecting statistics
  - Step 3: Investigating Artifacts
    - Investigating Event Receiver
    - Investigating Event Publisher
    - Investigating Event Processor
  - Step 4: Simulating Execution Environment
    - Collecting/recording events from streams
    - Replaying events to a Stream
- Available Tools

Troubleshooting and debugging guide

WSO2 CEP provides several options to understand the data flow and debug during issues.

If the system is not behaving as you expected then follow the below steps to find the route course of the issue.

Step 1: Assess the Event Flow

This will help you to visually see how the event flows through the system.

You might find some links been broken on the event flow and hence the system will not behave as expected. This should be fixed by inspecting inactive artifacts, if there are any, or by creating appropriate artifacts in order to construct the proper flow.

- Artifacts may have become inactive if there are errors during deployment of execution plans, event receivers or publishers.
- Execution plans can also become inactive when one of their connected event stream’s format is changed.
Step 2: Inspecting statistics

Inspecting events statistics will allow us to find the number of events processed via each of the event flow artifacts. Through this we can find whether the artifact is receiving and outputting events, this can give us hints like whether the artifact is dropping all events or no events have arrived to this artifact for processing.

- If events are dropped at the artifact then we have to inspect the artifact in detail (Step 3)
- If the events have not even arrived at the artifact then we can use the event flow to identify the artifacts that feed events to the current artifact (Step 1) and then inspect the statistics of those artifacts (Step 2) till we find the artifact causing the root cause.

Step 3: Investigating Artifacts

After identifying an artifact for investigation there are several options that you can try depending on the artifacts type you are investigating.

Investigating Event Receiver

- When we find events are not arrived at the receiver then
  - We have to investigate the client system to check whether it's sending to the correct endpoint or
  - If the receivers are polling an endpoint check whether it's configured correctly.
- If events arrive to the event receiver but no events are outputted by it
  - Then check the carbon logs or console for possible errors, else
  - Check event mapping configuration of the event receiver.
- If events are outputted by event receiver but the data is not in the expected format then
  - Enable event tracing to the event receiver and send some sample events to it.
  This will help you to find the incoming and outgoing messages of the event receiver. This will be very much helpful to reconfigure your event receiver to output the expected format or fix your client to send proper input.

After enabling event tracing, you can go to the monitoring section and see how messages have been received and outputted.
Investigating Event Publisher

- When we find events are not arrived at the publisher then we have to follow Step 1 to identity why no events are pushed to publishers.

- If events arrive to the event publisher but no events are outputted by it, then
  - Check the carbon logs or console for possible errors and
  - Use the event simulator feature to simulate some sample events to see how it behaves for different events.

- If events are outputted by event publisher but the data is not in the expected format, then
  - Enable event tracing on the event publisher and
  - Send some sample events to it using the event simulator feature

This will help you to find how the incoming message is formatted and sent out. This will be very helpful to reconfigure your event publisher to output the message in the expected format or the proper endpoint.

Investigating Event Processor

- When we find events are not arrived on one or more streams to the event processor then we have to follow Step 1 to identify why events have arrived on the stream to the to processor.

- If all expected events arrive to the event processor but if some of the expected output events are not outputted, then
  - Check the carbon logs or console for possible errors. If no errors are reported on the logs or console then
  - Use the event simulator feature to simulate some sample events to see how it behaves for different input events this will help you to give some hints on logical query errors.
  - You can also configure logger publisher to suspicious streams to find what events are transmitted by them.
  - If it does not help you to fix the issue use Siddhi Try-it to find the root cause why the queries are not working as expected.
• If all expected events arrive to the event processor and if some of the expected output events are not outputted on expected format, then
  
   • Enable event tracing on the event processor to identify if there are any issues in the incoming events, else
  
   • Send some sample events to it the event processor using the event simulator feature, this will help you to find why event processor behave in such style.
  
   • You can also configure logger publisher to suspicious streams to find what events are transmitted by them.
  
   • If it does not help you to fix the issue the use Siddhi Try-it to find the root course why the queries are not working as expected.

With Siddhi Try-it feature you can test the execution plan or query in detail. Here you can define the queries and their expected inputs, and this feature will produce the outputs of the queries at different levels helping you to find the root course of the issue.

**Step 4: Simulating Execution Environment**

In this step we can simulate the whole execution environment to understand its behaviour on different conditions and to fix issues if they exist. This can be done by constructing a sample dataset or recording the events and replying them.

**Collecting/recording events from streams**

To collect events from a stream create an RDBMS publisher for that stream and configure it to insert the events in the database by configuring the publisher similar to below.

![Create a New Event Publisher](image)

The configuration for the REDMS publisher is given below.
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher name="BusArrivalDepartureRDBMSPublisher"
    statistics="disable" trace="disable"
    xmlns="http://wso2.org/carbon/eventpublisher">
    <from streamName="busArrivalDepartureStream" version="1.0.0"/>
    <mapping customMapping="disable" type="map"/>
    <to eventAdapterType="rdbms">
        <property name="datasource.name">WSO2_GEO_EVENTS_DB</property>
        <property name="table.name">BusArrivalDepartures</property>
        <property name="execution.mode">insert</property>
    </to>
</eventPublisher>

Now you should be able to see table entries in the RDBMS (such as MySQL) for received events similar to below.

```
mysql> select * from BusArrivalDepartures;
```

<table>
<thead>
<tr>
<th>BUSID</th>
<th>ISARRIVED</th>
<th>STOPID</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1630</td>
<td></td>
<td>197</td>
<td>1450155600799</td>
</tr>
<tr>
<td>3119</td>
<td></td>
<td>30743</td>
<td>1450155600801</td>
</tr>
<tr>
<td>4178</td>
<td></td>
<td>BP4242</td>
<td>1450155600882</td>
</tr>
<tr>
<td>1550</td>
<td></td>
<td>1011</td>
<td>1450155600883</td>
</tr>
<tr>
<td>20223</td>
<td></td>
<td>2043</td>
<td>1450155600885</td>
</tr>
<tr>
<td>4201</td>
<td></td>
<td>37317</td>
<td>1450155600885</td>
</tr>
<tr>
<td>3482</td>
<td></td>
<td>BP2778</td>
<td>1450155600885</td>
</tr>
<tr>
<td>2718</td>
<td></td>
<td>34</td>
<td>1450155600886</td>
</tr>
<tr>
<td>944</td>
<td></td>
<td>BP5172</td>
<td>1450155600886</td>
</tr>
<tr>
<td>944</td>
<td></td>
<td>R0085</td>
<td>1450155600886</td>
</tr>
<tr>
<td>20240</td>
<td></td>
<td>1739</td>
<td>1450155600886</td>
</tr>
<tr>
<td>1548</td>
<td></td>
<td>7982</td>
<td>1450155600886</td>
</tr>
<tr>
<td>3662</td>
<td></td>
<td>33477</td>
<td>1450155600888</td>
</tr>
<tr>
<td>1313</td>
<td></td>
<td>33260</td>
<td>1450155600888</td>
</tr>
<tr>
<td>20216</td>
<td></td>
<td>1472</td>
<td>1450155600888</td>
</tr>
<tr>
<td>3488</td>
<td></td>
<td>4711</td>
<td>1450155600810</td>
</tr>
<tr>
<td>3145</td>
<td></td>
<td>1011</td>
<td>1450155600811</td>
</tr>
<tr>
<td>20215</td>
<td></td>
<td>FC41</td>
<td>1450155600813</td>
</tr>
<tr>
<td>3896</td>
<td></td>
<td>2640</td>
<td>1450155600814</td>
</tr>
<tr>
<td>20190</td>
<td></td>
<td>8419</td>
<td>1450155600814</td>
</tr>
<tr>
<td>20230</td>
<td></td>
<td>14581</td>
<td>1450155600815</td>
</tr>
<tr>
<td>942</td>
<td></td>
<td>30743</td>
<td>1450155600815</td>
</tr>
<tr>
<td>3891</td>
<td></td>
<td>BP5485</td>
<td>1450155600816</td>
</tr>
<tr>
<td>3118</td>
<td></td>
<td>29735</td>
<td>1450155600817</td>
</tr>
<tr>
<td>1880</td>
<td></td>
<td>7823</td>
<td>1450155600818</td>
</tr>
<tr>
<td>3101</td>
<td></td>
<td>143</td>
<td>1450155600818</td>
</tr>
<tr>
<td>20273</td>
<td></td>
<td>2046</td>
<td>1450155600820</td>
</tr>
</tbody>
</table>

Now the RDBMS is collecting events received for that particular stream. Whenever you want to stop collecting data,
you can remove the RDBMS publisher.

**Replaying events to a Stream**

If you want to replay the events collected for further debugging/troubleshooting, you can simply use the EventSimulator and configure it to replay for database. See screenshot below.

After the details are entered correctly and saved, that configuration will be saved and the table is available for replaying data. To resend the collected events, simply click the play button for the DB configuration.

Alternately you can also construct a .csv file with sample events and play that to simulate the event flow.

**Available Tools**

For more info refer following tools.

- Event Metrics
- Event Tracer
- Siddhi Try It Tool

**Event Metrics**

Event Metrics feature provides a drilled down visualisation of metrics relating to CEP specific artifacts to carry out
in-depth monitoring of throughput, latency, and memory usage of CEP. It is disabled by default to avoid the excessive processing of events and execution plan from causing a system overhead.

**Prerequisites**

Before you use event metrics, both JVM metrics and CEP component-specific metrics need to be enabled as follows.

- To enable metrics for the CEP, set the `Enabled` parameter under the Metrics element to `true` in the `<CEP_HOME>/repository/conf/metrics.xml` file. Alternatively, you can enable metrics at the time of starting the CEP server by using the following command:

  ```
  -Dmetrics.enabled=true
  ```

  For detailed instructions to enable/disable metrics and to configure metrics, see [Enabling Metrics and Storage Types](#).

- To generate metrics related statistics, set the `StatisticsReporterDisabled` property to `false` in the `<CEP_HOME>/repository/conf/carbon.xml` file as shown below.

  ```
  <StatisticsReporterDisabled>true</StatisticsReporterDisabled>
  ```

- To enable event metrics for specific event receivers, event publishers, and execution plans, enable statistics for the relevant components by clicking the **Enable Statistics** link. e.g., The following image shows how to enable statistics for an execution plan.

  ![Enable Statistics](image)

**Monitoring event metrics**

There are two methods to view event metrics:

- **Monitoring Metrics using the Management Console**
- **Monitoring Metrics using the JConsole**

Both methods generate the same metrics. The difference between the two methods is, when there are multiple tenants, the CEP Management Console allows you to view metrics for the currently logged tenant as well as other tenants whereas the JConsole only allows you to view metrics for the currently logged tenant.

**Monitoring Metrics using the Management Console**

Follow the procedure below to access event metrics via the Management Console.

1. Log into the CEP Management Console. For detailed instructions, see [Running the Product](#).
2. In the **Monitor** tab, click **Event Metrics**. The **View All Metrics** page is displayed as shown in the example
Alternatively, you can click **Metrics** to open the **View All Metrics** page.

If you click **Event Metrics**, the **View All Metrics** page displays only event metrics. If you click **Metrics**, the **View All Metrics** page displays both JVM metrics and event metrics.

Click on a link displayed under **Sub Levels** to view a specific metric as shown in the example below.
Monitoring Metrics using the JConsole

Follow the procedure below to monitor event metrics via the JConsole.

1. Start the WSO2 CEP server.
2. Start the JConsole. For detailed instructions to start the JConsole to monitor the CEP, see JMX-Based Monitoring.
3. Click the MBeans tab in the JConsole window. Then expand the org.wso2.carbon.metrics domain.
4. Click on the relevant MBeans under the `org.wso2.carbon.metrics` domain to view the required metrics.

**Event Tracer**

The CEP Event Tracer is an important tool to monitor events. This tool provides the functionality to trace the events that take place in each and every component. Event tracing is disabled for each configuration by default, and therefore, it is required to enable it manually.

**Note**

Please note that event tracer is supported only in super tenant mode. Event tracing capabilities are not available in tenant mode.

**Enabling/Disabling Event Tracer**

The above example shows how to enable/disable event tracing for an event receiver configuration. Event tracing can be enabled/disabled in the same way for Event Publishers and Execution Plans. However, note that enabling/disabling tracing for a configurations results in the redeployment of that configuration.
**Tracing the events**

Follow the instructions below to access the Event Tracer.

1. Start the CEP Management Console. For more information, see Running the Product.
2. Click the **Monitor** tab, and then click **Event Tracer** in the left navigator to open the **Event Message Tracer** page.

When event tracing is enabled for the required configurations, incoming events and any changes to events in CEP components are displayed as shown in the example below. You can click **Clear All** to permanently delete all the event related data.
You can use the **Search** parameter to refine the data displayed in the **Event Message Tracer** page as shown in the example below.

---

**Siddhi Try It Tool**

The Siddhi Try It is a tool used for experimenting event sequences through Siddhi Query Language (SiddhiQL) statements. You can define an execution plan to store the event processing logic and input an event stream to test the Siddhi query processing functionality.

Follow the steps below to use the Siddhi Try It tool.

1. Log in to the CEP management console, click **Tools**, and then click **Siddhi Try It** as shown below.

   ![Tools Menu](image)

   - Tools Menu
     - Try It
     - Event Simulator
     - Siddhi Try It

2. Enter the **Execution Plan** as shown in the below example.

   **Home** > **Tools** > **Siddhi Try It**

   ![Execution Plan](image)

   **Execution Plan**

   ```
   @plan: name('TestExecutionPlan')
   define stream sensorStream (sensorId string, temperature float);
   @info(name = 'query1')
   from sensorStream[temperature>90.6]
   select sensorId
   insert into outputStream;
   ```

   The main elements of an execution plan are described below.

   For more information on execution plans, see the **Working with Execution Plans**.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution plan name</td>
<td>The name of the execution plan. It can contain only alphanumeric characters and '_.' character.</td>
<td>TestExecutionPlan</td>
</tr>
</tbody>
</table>
### Input stream
The mappings between the available event stream and the input stream of the Siddhi runtime, which is defined inside the query expressions.

<table>
<thead>
<tr>
<th>Query expressions</th>
<th>The event processing logic written in Siddhi QL.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When defining more than one query, end each query with a semicolon. Defining a query name (e.g. query1) is optional.</td>
</tr>
<tr>
<td></td>
<td><code>@info(name = 'query1')</code></td>
</tr>
<tr>
<td></td>
<td>`from sensorStream[temp</td>
</tr>
<tr>
<td></td>
<td><code>select sensorId</code></td>
</tr>
<tr>
<td></td>
<td><code>insert into outputStream;</code></td>
</tr>
</tbody>
</table>

### Output stream
The mappings between the output stream of the Siddhi runtime and one of the available event streams, which is defined inside the query expressions.

<table>
<thead>
<tr>
<th></th>
<th>outputStream</th>
</tr>
</thead>
</table>

3. Enter a time stamp to begin the process of sending events for **Begin Time** if required.

4. Enter the **Event Stream**, which is a logical series of events ordered based on the time as shown in the below example.

   ![Event Stream](image)

   An event stream can contain a delay between events. When defining a delay, enter the delay time in milliseconds as shown in the example below. Furthermore, for scheduler related queries, you need to set up a delay with a necessary time in the event stream. For more information on event streams, see [Working with Event Streams](#).

5. Click **Submit**. You view the input stream and the results of the execution plan under the defined output stream, and separated query outputs as shown below.
Managing CEP Artifacts Via the Template Manager

In Data Analytics, there are common use cases for analyzing statistics that involve operations such as calculating the average, minimum, maximum etc., for different endpoints, and visualising them graphically using suitable charts. The Template Manager tool in WSO2 CEP allows you to define a template in order to generate the artifacts needed for different scenarios with common requirements in a convenient manner. Domain-specific parameter values (e.g., attribute name, time duration) can be customized and configured in a dashboard.

Once you create a template with required configurable parameters, WSO2 CEP generates new artifacts such as execution plans, event streams, gadgets etc. for each new scenario that you create for that template. This is done by modifying the pre-defined template with user-defined values and then deploying it.

For example, consider a scenario where you need the following three streams.

<table>
<thead>
<tr>
<th>Stream</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.event.temperature.stream</td>
<td>To calculate sum, average, maximum and minimum values for temperature over a period of time and display the results on a dashboard.</td>
</tr>
<tr>
<td>org.wso2.event.pressure.stream</td>
<td>To calculate sum, average, maximum and minimum values for pressure over a period of time and display the results on a dashboard.</td>
</tr>
<tr>
<td>org.wso2.event.humidity.stream</td>
<td>To calculate sum, average, maximum and minimum values for humidity over a period of time and display the results on a dashboard.</td>
</tr>
</tbody>
</table>

Instead of creating each artifact individually, you can create a template based on this scenario, and then use the Template Manager tool to generate artifacts representing each sensor type for a given time period. This allows configurations to be reused, and simplifies the creation of artifacts for business users with limited knowledge of CEP artifacts.

Currently, this feature supports following WSO2 CEP artifact types

- Eventstream
- Realtime - Execution Plan
- Event Receiver
- Event Publisher
- Gadget
- Dashboard

The following sections provide detailed instructions to create templates and to use them.

- Configuring a Template for Template Manager
- Using Templates
Configuring a Template for Template Manager

This section illustrates how to configure templates for WSO2 CEP template manager. We will be utilizing a single use case throughout this documentation based on which we will provide sample configurations.

- **Use case**
- **Step 1: Create template**
- **Step 2: Add an event stream to the template**
- **Step 3: Configure real time analytics**
- **Step 4: Configure an event receiver**
- **Step 5: Configure an event publisher**
- **Step 6: Configure a gadget**
- **Step 7: Configure a dashboard**
- **Step 8: Configure stream mapping**
- **Step 9: Configure parameters**
- **Step 10: Configure common artifacts**
- **Step 11: Configure Scripts**

**Use case**

In this use case we will develop a template representing a scenario where you can monitor a configurable sensor type (temperature, pressure, etc) throughout a configurable time period and calculate the number of sensor data items received, the number of sensors engaged and the sum, average, maximum and minimum values for the data-set. After configuring the template manager, you can generate artifacts for different scenarios using the user interface of template manager.

**Step 1: Create template**

To create the template, enter the following configuration in an XML file, and save this file in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory as `Sensor-Analytics.xml`. 
<domain name="SensorDataAnalysis">
    <description>Domain for sensor data analysis</description>
    <scenarios>
        <scenario type="SensorAnalytics">
            <description>Configure a sensor analytics scenario to display statistics for a given stream of your choice</description>
            <templates>
                <!--Note: These will be deployed in the order they appear here-->
                <template type="eventstream">
                    <!--A Stream Definition Template here-->
                </template>
                <template type="realtime">
                    <!--An Execution Plan Template here-->
                </template>
                <template type="eventreceiver">
                    <!--An Event Receiver Template here-->
                </template>
                <template type="eventpublisher">
                    <!--An Event Publisher Template here-->
                </template>
                <template type="gadget">
                    <!--A Gadget Template here-->
                </template>
                <template type="dashboard">
                    <!--A Dashboard Template here-->
                </template>
            </templates>
            <streamMappings>
                <!--Define stream mappings here-->
            </streamMappings>
            <parameters>
                <!--Define parameters here-->
            </parameters>
        </scenario>
    </scenarios>
    <commonArtifacts>
        <!--Define common artifacts here-->
    </commonArtifacts>
    <scripts>
        <!--Define JavaScript files and/or content here-->
    </scripts>
</domain>

This is the basic template. Each section of the xml configuration is explained below. Instructions to configure each section in this template are explained in the subsequent steps. File name, domain name and scenario type attributes can have custom names based on the use-case.
### Configuration Type

<table>
<thead>
<tr>
<th>Configuration Type</th>
<th>Purpose</th>
<th>Sub Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>This is a logical collection of Scenarios. One domain can be used to group related scenarios together for better organization.</td>
<td><code>&lt;domain name=&quot;SensorDataAnalysis&quot;&gt;</code></td>
</tr>
<tr>
<td>Scenario</td>
<td>A description of the scenario to be analyzed. e.g., Analyzing average and maximum temperature.</td>
<td><code>&lt;scenario type=&quot;SensorAnalytics&quot;&gt;</code></td>
</tr>
<tr>
<td>Template</td>
<td>A templated artifact.</td>
<td><code>&lt;template type=&quot;type&quot;&gt;</code></td>
</tr>
<tr>
<td>Event Stream</td>
<td>This defines the event stream definitions associated with the scenario.</td>
<td><code>&lt;template type=&quot;eventstream&quot;&gt;</code></td>
</tr>
<tr>
<td>Execution Plan</td>
<td>This defines the Siddhi execution plan based on which the events are processed.</td>
<td><code>&lt;template type=&quot;realtime&quot;&gt;</code></td>
</tr>
<tr>
<td>Gadget</td>
<td>This defines the gadgets required to view the information processed by the event flows created via the template in a specific format.</td>
<td><code>&lt;template type=&quot;gadget&quot;&gt;</code></td>
</tr>
<tr>
<td>Dashboard</td>
<td>This defines the dashboards to display the gadgets with the information processed by the event flow.</td>
<td><code>&lt;template type=&quot;dashboard&quot;&gt;</code></td>
</tr>
<tr>
<td>Stream mapping</td>
<td>This maps an existing input stream in your WSO2 CEP installation with the event stream defined in the template manager.</td>
<td><code>&lt;streamMappings&gt;</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>This defines configurable variables. The $ sign is used to indicate configurable fields.</td>
<td><code>&lt;parameters&gt;</code></td>
</tr>
<tr>
<td>Common artifacts</td>
<td>This allows you to include artifacts that are common to all the scenarios using the templates in the template manager.</td>
<td><code>&lt;commonArtifacts&gt;</code></td>
</tr>
<tr>
<td>Scripts</td>
<td>To include java script functions that can be used as parameters.</td>
<td><code>&lt;scripts&gt;</code></td>
</tr>
</tbody>
</table>

**Step 2: Add an event stream to the template**

This step involves adding event stream definitions to be included in the template. For detailed information about event streams, see [Understanding Event Streams](#).

To add required event streams for this use case, add the following configuration under the `<template type="event stream">` element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file you created, and save.
The configuration you are providing here is a JSON representation of the event stream definition. The $ sign is used for configurable fields. The name of the event stream defined in this example is org.wso2.event.$sensorType.stream.

This step introduces the $sensorType parameter included in the event stream template given above. This is a configurable parameter that can be configured in the Template Manager UI to create different CEP event stream artifacts as required for the use case.

The following procedure is an easy approach to create the configuration for an event stream to be added to the Template Manager.

1. Log into the WSO2 CEP Management Console and define the required event stream. For detailed instructions, see Understanding Event Streams.
2. In the Available Event Streams page, click Edit for the stream you created and click switch to source view where configuration is available in the form of text.
3. Copy this configuration in the source view and add it to your template under the <template type="event stream"> element. Add the required configurable parameters and save.

**Step 3: Configure real time analytics**

This step involves adding an execution plan to the template. For more information about execution plans, see Creating a Standalone Execution Plan.

To add required event streams for this use case, add the following configuration under the <template type="realtime"> element in the <CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml file you created, and save.
An execution plan configuration should always be added inside a CDATA element.

**Real-time Element**

```xml
<template type="realtime">
  <![CDATA[/* Enter a unique ExecutionPlan */
    @Plan:name('SensorAnalyticsPlan')

    /* Enter a unique description for ExecutionPlan */
    -- @Plan:description('ExecutionPlan')

    /* define streams/tables and write queries here ... */

    @Import('org.wso2.event.$sensorType.stream:1.0.0')
    define stream InputStream (sensor_id string, sensor_value double);

    @Export('org.wso2.event.$sensorType.statistics.stream:1.0.0')
    define stream OutputStream (sensor_type string, sensor_id_distinct_count long, count long, sum double, average double, max double, min double);

    from InputStream#window.time($timeInMins min)
    select '$sensorType' as sensor_type, distinctcount(sensor_id) as sensor_id_distinct_count, count() as count, sum(sensor_value) as sum, avg(sensor_value) as average, max(sensor_value) as max, min(sensor_value) as min
    insert into OutputStream;]]>
  </template>
```

In this example, the execution plan consumes the org.wso2.event.$sensorType.stream stream and calculates values for the count, sum, avg, max and etc. attributes over a time period specified for the $timeInMins configurable parameter. The results are published to another stream named org.wso2.event.$sensorType.statistics.stream.

This step introduces the $timeInMins parameter included in the execution plan template given above. This is a configurable parameter that can be configured in the Template Manager UI to create different CEP event stream artifacts as required for the use case.

The following procedure is an easy approach to create the configuration for an execution plan to be added to the Template Manager.

1. Log into the WSO2 CEP Management Console and create the required execution plan. For detailed instructions to create an execution plan, see Creating a Standalone Execution Plan.
2. Once you have completed and validated the execution plan, copy it and paste it under the <template type="realtime"> element of your template file in the <CEP_HOME>/repository/conf/tem plate-manager/domain-template directory.
3. Add the required configurable parameters and save.

*Step 4: Configure an event receiver*
This step involves adding an event receiver to the template. Event receivers accept events to WSO2 CEP from external sources. For more information about event receivers, see Configuring Event Receivers.

To add the event receiver configuration required for this use case, add the following configuration under the `<template type="eventReceiver">` element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

```
<template type="eventReceiver">
  <![CDATA[
    <eventReceiver name="$sensorType.statistics.stream.receiver"
                    statistics="disable" trace="disable"
                    xmlns="http://wso2.org/carbon/eventreceiver">
      <from eventAdapterType="http">
        <property name="basicAuthEnabled">true</property>
        <property name="transports">all</property>
      </from>
      <mapping customMapping="disable" type="xml"/>
      <to streamName="org.wso2.event.$sensorType.stream" version="1.0.0"/>
    </eventReceiver>
  ]]>}
</template>
```

In this example, an HTTP event receiver is added, and the events received are mapped to the `org.wso2.event.$sensorType.stream` stream.

The following procedure is an easy approach to create the configuration for an event receiver to be added to the Template Manager.

1. Log into the WSO2 CEP Management Console and create the required event receiver. For detailed instructions to create an event receiver, see Configuring Event Receivers - Creating event receivers.
2. In the Available Event Receivers page, click Edit for the event receiver you created. The source view for the event receiver configuration is displayed.
3. Copy the source view and paste it under the `<template type="eventReceiver">` element of your template file in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory. Make sure you include this configuration within a CDATA block.
4. Add the required configurable parameters and save.

**Step 5: Configure an event publisher**

This step involves adding an event publisher to the template. Event publishers publish events from WSO2 CEP to external consumers. For more information about event publishers, see Configuring CEP to Create Alerts.

To add the event publisher configuration required for this use case, add the following configuration under the `<template type="eventPublisher">` element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

```
<template type="eventPublisher">
  <![CDATA[
    <eventPublisher name="$sensorType.statistics.stream.publisher"/>
  ]]>}
</template>
```

An event receiver configuration should always be added inside a CDATA element.
In this example, a UI event publisher consumes the org.wso2.event.$sensorType.statistics.stream stream, and publishes the events from this stream to an internal UI adapter so that the processed events are displayed as statistics in the gadgets created for this scenario.

The following procedure is an easy approach to create the configuration for an event publisher to be added to the Template Manager.

1. Log into the WSO2 CEP Management Console and create the required event publisher. For detailed instructions to create an event publisher, see Configuring CEP to Create Alerts - Creating event publishers.
2. In the Available Event Publishers page, click Edit for the event receiver you created. The source view for the event publisher configuration is displayed.
3. Copy the source view and paste it under the <eventPublisher> element of your template file in the <CEP_HOME>/repository/conf/template-manager/domain-template directory. Make sure you include this configuration within a CDATA block.
4. Add the required configurable parameters and save.

**Step 6: Configure a gadget**

This step involves adding a gadget configuration to the template. For more information about gadgets, see Creating Gadgets.

To add the gadget configuration required for this use case, add the following configuration under the <template type="gadget"> element in the <CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml file.
Gadget Element

```xml
<template type="gadget">
  <config>
    <properties>
      <property name="directoryName">$sensorType-count-chart</property>
      <property name="templateDirectory">numberchart</property>
    </properties>
    <artifacts>
      <artifact file="gadget.json"><![CDATA[{ "id": "$sensorType-count-chart", "title": "$sensorType-count-chart", "type": "gadget", "thumbnail": "gadget/$sensorType-count-chart/thumbnail.png", "data": { "url": "gadget/$sensorType-count-chart/gadget.xml" } }]]></artifact>
      <artifact file="conf.json"><![CDATA[{"provider-conf": {"streamName": "org.wso2.event.$sensorType.statistics.stream:1.0.0", "provider-name": "realtime"}, "chart-conf": {"x": "count", "title": "Count $sensorType", "gadget-name": "$sensorType-count-chart", "chart-name": "number-chart"}}]]]></artifact>
      <artifact file="js/core/gadget-util.js"><![CDATA[function getGadgetLocation(){ return '/portal/store/carbon.super/fs/gadget/$sensorType-count-chart'; }]]></artifact>
    </artifacts>
  </config>
</template>
```

The above configuration represents a chart in which the `count` attribute of the `org.wso2.event.$sensorType.statistics.stream` stream is mapped.

The `directoryName` property element in the configuration specifies the name of the directory to which the gadget configurations are copied when the template is used at runtime. The `templateDirectory` property element is the directory from which the static gadget configurations are copied. There are many files associated with the gadget, and most of the time, you only need to template the few files that you add to the template. The other files are static. Therefore, you need to add all the static files (i.e. files that are not templates) to the `templateDirectory` directory and place it inside the `<CEP_HOME>/repository/conf/template-manager/gadget-templates` directory. E.g., if you add a gadget template configuration in the domain template file as given above together with a directory named `numberchart` including all the gadget related configurations (all the non-templated files), then all the gadget related configurations (templated files as well as non-templated files) are copied to the `$sensorType-count-chart` directory, replacing the input values passed by the user.

Note that there are 3 artifacts added under this template type: `gadget.json`, `conf.json` and `js/core/gadget-util.js`. Follow the steps below to to generate this content.
1. Create a new gadget as required for the scenario. For detailed instructions, see Creating Gadgets.
2. The artifacts mentioned above are located in the `<CEP_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>/fs/gadget` directory. Copy the content of the relevant files (where the file name is the same as the required gadget name) and insert them within CDATA blocks under the `<template type="gadget"> element of your template file in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory.
3. Add the required configurable parameters and save.

**Step 7: Configure a dashboard**

This step involves adding a dashboard configuration to the template. This dashboard serves as a container for the gadgets that were added in step 6. For more information about dashboards, see Visualizing Results in the Analytics Dashboard.

To add the dashboard configuration required for this use case, add the following configuration under the `<template type="dashboard"> element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

### Dashboard element

```xml
<template type="dashboard">
  <config>
    <property name="dashboardId">analytics-$sensorType-dashboard</property>
  </config>
  <content><![CDATA[
    "accessTokenUrl": "",
    "apiKey": "",
    "apiSecret": "",
    "banner": {
      "customBannerExists": false,
      "globalBannerExists": false
    },
    "defaultPriority": "5",
    "description": "",
    "hideAllMenuItems": false,
    "id": "analytics-$sensorType-dashboard",
    "identityServerUrl": "",
    "isEditorEnable": true,
    "isUserCustom": false,
    "isanon": false,
    "landing": "landing",
    "menu": [
      {
        "id": "landing",
        "isanon": false,
        "ishidden": false,
        "subordinates": [],
        "title": "Home"
      }
    ],
    "pages": [
```
{
  "content": {
    "anon": {},
    "default": {
      "a": {
        "content": {
          "data": {
            "url": "fs://gadget/$sensorType-count-chart/index.xml"
          },
          "id": "$sensorType-count-chart",
          "locale_titles": {},
          "options": {
            "dataSource": {
              "options": [],
              "required": false,
              "title": "Data Source",
              "type": "STRING",
              "value": "/portal/gadgets/bar-chart/datasource/dataFile4.jag"
            },
            "updateGraph": {
              "options": [],
              "required": false,
              "title": "Update Interval (s)",
              "type": "STRING",
              "value": "No"
            }
          },
          "styles": {
            "borders": true,
            "title": "Count $sensorType"
          },
          "thumbnail": "fs://gadget/$sensorType-average-chart/gadgetIcon.png",
          "title": "Count $sensorType",
          "type": "gadget"
        },
        "id": "$sensorType-count-chart-0"
      }
    }
  },
  "id": "landing",
  "isanon": false,
  "layout": {
    "id": "landing",
    "isanon": false,
    "layout": {
      ...
"content": {
    "loggedIn": {
        "blocks": [
            {
                "height": 3,
                "id": "a",
                "width": 4,
                "x": 0,
                "y": 0
            }
        ]
    }},
    "fluidLayout": false
},
    "title": "Home"
},
"permissions": {
    "editors": [
        "Internal/everyone"
    ],
    "viewers": [
        "Internal/everyone"
    ]
},
"theme": "Default Theme",
"title": "Analytics $sensorType Dashboard"
In the above configuration, a new page is created to add the previously created gadgets. Fields such as `id` are provided so that the created gadgets are added to the dashboard when generating artifacts using this template.

The content of the above dashboard template can be generated by following the procedure below.

1. Create a sample dashboard as required for your scenario. For detailed instructions, see Visualising Results in the Analytics Dashboard - Adding a new dashboard.
   The dashboard configuration is saved in the `/system/config/ues/dashboards/<Dashboard_Name>` registry path. To access the registry, log into the WSO2 CEP Management Console, and click `Main => Registry => Browse`. For more information about the registry, see Registry.
2. Copy the dashboard configuration, and add it within a CDATA block under the `<template type="dashboard">` element of your template file in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory.

Step 8: Configure stream mapping

This step involves configuring the stream mapping required for this scenario. The stream mapping maps an existing input stream in your WSO2 CEP installation with the event stream defined in the template manager.

For example, if you have a single event stream named `org.wso2.event.aggregate.stream` that carries all of your sensor data, you need to break it down at runtime to make use of the template you created. This is achieved via stream mapping. The stream mapping configuration given below allows you to map the `org.wso2.event.aggregate.stream` to the `org.wso2.event.$sensorType.stream` stream you added to this template in step 2. This mapping is done in the Template Manager UI as further explained in Using Templates.

Add the following configuration under the `<streamMappings>` element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

```
<streamMappings>
  <streamMapping to="org.wso2.event.$sensorType.stream:1.0.0"/>
</streamMappings>
```

Step 9: Configure parameters

This step involves configuring the parameters required for this scenario. Two configurable parameters named `$sensorType` and `$timeInMins` were introduced to the template in this scenario in the previous steps. In this section, the following are configured for these parameters.

- Type
  - The name and the description to be displayed in the Template Manager UI.
  - The default value and the options that you can select as the value of the parameter in the Template Manager UI.

Using these parameters to differentiate scenarios in the Template Manager is explained under Using Templates.

To add parameters to the template, add the following configuration under the `<parameters>` element in the `<CEP_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.
Step 10: Configure common artifacts

This step involves configuring the artifacts common to all scenarios. The Template Manager allows you to add multiple scenarios within a single base template file. This section explains how to share these artifacts across scenarios without creating conflicts. To configure common artifacts, add the following configuration under the `<com monArtifacts>` element to the `<CEP_HOME>/repository/conf/template-manager/domain-template/S ensor-Analytics.xml` file.

```
<parameters>
    <parameter name="timeInMins" type="int">
        <displayName>Time(Mins)</displayName>
        <description>The sliding time period for which the window should hold events</description>
        <defaultValue>1</defaultValue>
    </parameter>
    <parameter name="sensorType" type="string">
        <displayName>Sensor Type Name</displayName>
        <description>The name of the sensor type</description>
        <defaultValue>temperature</defaultValue>
        <options>temperature,humidity,pressure</options>
    </parameter>
</parameters>

Step 11: Configure Scripts
A script can be included either using an external file that is stored in the `<CEP_HOME>/repository/conf/template-manager/scripts` directory as a source attribute, or the actual content can be provided as a value. Following example shows how to configure the script both ways respectively.

```xml
<scripts>
  <!--This script points to the
  <CEP_HOME>/repository/conf/template-manager/scripts/wso2-commons.js
  file.-->
  <script src="wso2-commons.js"/>
  <script>
    var toId = function (name) {
      return name.toLowerCase().replace(/ /g, '');
    }
  </script>
</scripts>
```

The JavaScript functions defined in the scripts as well as any default JavaScript functions can be used in any place where parameters can be used. The function calls should be placed in between `${` and `}`. An example is given below in declaration of event publisher.

```xml
<template type="eventpublisher">
  <![CDATA[
    <eventPublisher name="${toId('"$sensorType"').statistics.stream.publisher"
      statistics="disable" trace="disable"
      xmlns="http://wso2.org/carbon/eventpublisher">
      <from
        streamName="org.wso2.event."${"$sensorType".toLowerCase()}.statistics.stream"
        version="1.0.0"/>
      <mapping customMapping="disable" type="wso2event"/>
      <to eventAdapterType="ui"/>
    </eventPublisher>
  ]]></template>
```

The `SensorAnalyticsDomain.xml` file with all the required configurations is available by default in the `<CEP_HOME>/repository/conf/template-manager/domain-template` directory. This template contains all the configurations given above except stream mapping, common artifacts and scripts. You can try out these templates as explained in **Using Templates**.

**Using Templates**

This section explains how to generate artifacts representing different scenarios based on previously created templates using the Template Manager dashboard.

**Prerequisites**

Before you try out this scenario, a template should be created as described in **Configuring a Template for Template**
Manager.

Using the template

Follow the procedure below to use the previously created SensorDataAnalysis template in different scenarios.

1. Log in to the management console using the following URL.
   https://<CEP_HOST>:<CEP_PORT>/carbon/
2. Click Main, and then click Template Manager under the Manage menu. The following dashboard home page appears with the available domains as shown below.

![Domains](image)

3. Select the relevant domain to configure the defined template (in this example it is SensorDataAnalysis). This opens the Deployed Scenarios page.
4. Click Create New Scenario to add a new scenario to the selected domain. This opens the Edit Scenario page.
1. Enter values in the **Edit Scenario** page as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario Type</td>
<td>SensorAnalytics</td>
<td>This is the template created by following the instructions in <strong>Configuring a Template for Template Manager</strong>.</td>
</tr>
<tr>
<td>Scenario Name</td>
<td>SensorAnalytics</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Calculate maximum and average temperature values.</td>
<td></td>
</tr>
</tbody>
</table>

Enter values for the configurable parameters as shown below.

<table>
<thead>
<tr>
<th>Configurable Parameters</th>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(Mins)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
5. Click Add Scenario. A pop-up message appears to inform you that the configuration is successfully saved. Close this message. The scenario you configured is displayed in the Deployed Scenarios page as follows.

You can edit or delete this scenario by clicking the relevant link under Actions.

In this example, the scenario gets deployed at this stage because no stream mapping is configured in the default SensorAnalytics template used. If you have configured stream mapping for your template, you are redirected to the Stream Mapping page when you click Add Scenario. Select an appropriate predefined stream from the list for the Mapped From: Stream Name field. This expands the page to display the Attribute Mapping section as demonstrated below. Map attributes as required and click Save Mapping.

Once this configuration is successfully completed, the artifacts included in the template are deployed in
WSO2 CEP. You can view the event flow for the template artifacts by logging into the WSO2 CEP Management Console, and clicking **Main => Flow** where it is displayed as follows.

6. In the WSO2 CEP Management Console, click **Main** and then click **Analytics Dashboard** to open the Analytics Dashboard. Log in with your credentials. The dashboard named `analytics-temperature-dashboard` that is included in the template used in this example is displayed as follows.

![Dashboard](image)

7. To simulate data in order to view statistics in the `analytics-temperature-dashboard` dashboard, do the following. For more information, see *Sending Multiple Events Using a File*.

   1. In the WSO2 CEP Management Console, click **Tools** and then click **Event Simulator** to open the **Event Stream Simulator** page.
   2. Click **Configure** for the `events.csv` file to open the **Event Mapping Configuration** dialog box. Enter the following values in this dialog box.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the target event stream</td>
<td><code>org.wso2.event.temperature.stream:1.0.0</code></td>
</tr>
<tr>
<td>Field delimiter</td>
<td>,</td>
</tr>
</tbody>
</table>

   Click **Configure** and then click **OK** to close the message that appears to confirm that the configuration is successful.

   3. Click **Play** to simulate the event flow.
8. Once you simulate the events, access the Analytics Dashboard and mentioned in step 6, and then click View on the analytics-temperature-dashboard dashboard. Data published to the dashboard is displayed as shown below.
Admin Guide

The following topics explore various product deployment scenarios and other topics useful for system administrators.

- Performance Tuning Recommendations
- Monitoring WSO2 CEP
- Deployment Automation Guide
- Clustered Deployment
- User Management
- Working with Databases
- Registry
- Using Asymmetric Encryption
- Fixing Security Vulnerabilities
- Logging
- Feature Management
- Transports
- Enabling Java Security Manager
- Multitenancy
- Integration

Performance Tuning Recommendations

This section describes some recommended performance tuning configurations to optimize WSO2 CEP. It assumes that you have set up WSO2 CEP on a server running Unix/Linux, which is recommended for a production deployment.

- OS-Level Settings
- JVM settings
- WSO2 Carbon platform-level settings
- JDBC Pool configuration
- CEP-Level settings

**Important**

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The parameter values we discuss below are just examples. They might not be the optimal values for the specific hardware configurations in your environment. We recommend that you carry out load tests on your environment to tune the product accordingly.

**OS-Level Settings**

1. To optimize network and OS performance, configure the following settings in `/etc/sysctl.conf` file of Linux. These settings specify a larger port range, a more effective TCP connection timeout value, and a number of other important parameters at the OS-level.
When we have the localhost port range configuration lower bound to 1024, there is a possibility that some processes may pick the ports which are already used by WSO2 servers. Therefore, it’s good to increase the lower bound as sufficient for production, e.g., 10,000.

2. To alter the number of allowed open files for system users, configure the following settings in /etc/security/limits.conf file of Linux.

```
* soft nofile 4096
* hard nofile 65535
```

Optimal values for these parameters depend on the environment.

3. To alter the maximum number of processes your user is allowed to run at a given time, configure the following settings in /etc/security/limits.conf file of Linux (be sure to include the leading * character). Each carbon server instance you run would require upto 1024 threads (with default thread pool configuration). Therefore, you need to increase the nproc value by 1024 per each carbon server (both hard and soft).

```
* soft nproc 20000
* hard nproc 20000
```

**JVM settings**

When an XML element has a large number of sub-elements and the system tries to process all the sub-elements, the system can become unstable due to a memory overhead. This is a security risk.

To avoid this issue, you can define a maximum level of entity substitutions that the XML parser allows in the system. You do this using the entity expansion limit attribute that is in the `<CEP_HOME>/bin/wso2server.bat` file (for Windows) or the `<CEP_HOME>/bin/wso2server.sh` file (for Linux/Solaris). The default entity expansion limit is 64000.

```
-DentityExpansionLimit=100000
```

In a clustered environment, the entity expansion limit has no dependency on the number of worker nodes.
**WSO2 Carbon platform-level settings**

In multitenant mode, the WSO2 Carbon runtime limits the thread execution time. That is, if a thread is stuck or taking a long time to process, Carbon detects such threads, interrupts and stops them. Note that Carbon prints the current stack trace before interrupting the thread. This mechanism is implemented as an Apache Tomcat valve. Therefore, it should be configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file as shown below.

```xml
<Valve
className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve" threshold="600"/>
```

- The `className` is the Java class used for the implementation. Set it to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. The default value is 600 seconds.

**JDBC Pool configuration**

Within the WSO2 platform, we use Tomcat JDBC pooling as the default pooling framework due to its production ready stability and high performance. The table below indicates some recommendations on how to configure the JDBC pool using the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. For more details about recommended JDBC configurations, see [The Tomcat JDBC Connection Pool](#).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 10.</td>
<td>This value should match the maximum number of requests that can be expected at a time in your production environment. This is to ensure that, whenever there is a sudden increase in the number of requests to the server, all of them can be connected successfully without causing any delays. Note that this value should not exceed the maximum number of requests allowed for your database.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of connections that can remain idle in the pool, without extra ones being created. The connection pool can shrink below this number if validation queries fail. Default value is 0.</td>
<td>This value should be similar or near to the average number of requests that will be received by the server at the same time. With this setting, you can avoid having to open and close new connections every time a request is received by the server.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether connection objects will be validated before they are borrowed from the pool. If the object validation fails, it will be dropped from the pool, and we will attempt to borrow another connection.</td>
<td>Setting this property to 'true' is recommended as it will avoid connection requests from failing. The validationQuery property should be used if testOnBorrow is set to true. To increase the efficiency of connection validation and to improve performance, validationInterval property should also be used.</td>
</tr>
</tbody>
</table>
validationInterval

To avoid excess validation, run validation at most at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).

This time out can be as high as the time it takes for your DBMS to declare a connection as stale. For example, MySQL will keep a connection open for as long as 8 hours, which requires the validation interval to be within that range. However, note that having a low value for validation interval will not incur a big performance penalty, specially when database requests have a high throughput. For example, a single extra validation query run every 30 seconds is usually negligible.

validationQuery

The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw an SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).

Specify an SQL query, which will validate the availability of a connection in the pool. This query is necessary when testOnBorrow property is true.

---

When it comes to web applications, users are free to experiment and package their own pooling framework such BoneCP.

**CEP-Level settings**

Performance tuning can be tried out in the following areas at the CEP level. The performance is considered in terms of throughput per second (TPS) and latency.

- Receiving events
- Publishing events
- Configurations for extreme latency scenarios

**Receiving events**

The following parameters which affect the performance relating to receiving events are configured in the `<CEP_HOME>/repository/conf/data-bridge/data-bridge-config.xml` file. These configurations are common for both thrift and binary protocols.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>workerThreads</td>
<td>The number of threads reserved to handle the load of events received.</td>
<td>10</td>
<td>This value should be increased if you want to increase the throughput by receiving a higher number of events at a given time. The number of available CPU cores should be considered when specifying this value. If the value specified exceeds the number of CPU cores, higher latency would occur as a result of context switching taking place more often.</td>
</tr>
</tbody>
</table>
**maxEventBufferCapacity**  
The maximum size allowed for the event receiving buffer in mega bytes. The event receiving buffer temporarily stores the events received before they are forwarded to an event stream.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxEventBufferCapacity</td>
<td>The maximum size allowed for the event receiving buffer in mega bytes. The event receiving buffer temporarily stores the events received before they are forwarded to an event stream.</td>
<td>10</td>
<td>This value should be increased when there is an increase in the receiving throughput. When increasing the value heap memory size also needs to be increased accordingly.</td>
</tr>
</tbody>
</table>

**eventBufferSize**  
The number of messages that is allowed in the receiving queue at a given time.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventBufferSize</td>
<td>The number of messages that is allowed in the receiving queue at a given time.</td>
<td>200</td>
<td>This value should be increased when there is an increase in the receiving throughput.</td>
</tr>
</tbody>
</table>

**Publishing events**

The following parameters which affect the performance relating to publishing events are configured in the `<CEP_HOME>/repository/conf/data-bridge/data-agent-config.xml` file. These configurations are common for both thrift and binary protocols.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
<th>Recommendation</th>
</tr>
</thead>
</table>
| PublishingStrategy | The strategy used by data-agent to publish messages. The possible strategies are as follows.  
• async: Events will be published in a non-blocking asynchronous manner.  
• sync: Events will be published in a blocking synchronous manner. | async         | For normal use-cases use the async publishing strategy which uses event queues and internal thread pools to process events. For extreme latency scenarios use the sync strategy. This disables internal queues and thread pools. The client thread carries the event through the execution flow and emits the result, decreasing the latency significantly. However this approach decreases the throughput. |
| QueueSize      | The size of the queue event disruptor which handles events before they are published to an application/data store. This is not applicable if publishing strategy is sync. | 32768         | The value specified should always be the result of an exponent with 2 as the base. (e.g., 32768 is $2^{15}$).  
A higher value should be specified when a higher throughput needs to be handled. However, the increase in the load handled at a given time can reduce the speed at which the events are processed. Therefore, a lower value should be specified if you want to reduce the latency. |
<p>| BatchSize      | The maximum number of events in a batch sent to the queue event disruptor at a given time. This is not applicable if publishing strategy is sync. | 200           | This value should be assigned proportionally to the throughput of events handled. Greater the batch size, higher will be the number of events sent to the queue event disruptor at a given time. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorePoolSize</td>
<td>The number of threads that will be reserved to handle events at the time you start the CEP server. This value will increase as throughput of events handled increases, but it will not exceed the value specified for the MaxPoolSize parameter. This is not applicable if publishing strategy is sync.</td>
<td>1</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>The maximum number of threads that should be reserved at any given time to handle events. This is not applicable if publishing strategy is sync.</td>
<td>1</td>
</tr>
</tbody>
</table>

For better throughput you can configure the parameters as follows.

```xml
<QueueSize>32768</QueueSize>
<BatchSize>200</BatchSize>
<CorePoolSize>1</CorePoolSize>
<MaxPoolSize>1</MaxPoolSize>
```

For reduced latency, you can configure the parameters as follows.

```xml
<QueueSize>256</QueueSize>
<BatchSize>200</BatchSize>
<CorePoolSize>1</CorePoolSize>
<MaxPoolSize>1</MaxPoolSize>
```

**Configurations for extreme latency scenarios**

The following configurations are recommended when the latency is extremely high.

- Do the following configurations in the `<CEP_HOME>/repository/conf/data-bridge/data-bridge-config.xml` file.

  ```xml
  <workerThreads>1</workerThreads>
  <eventBufferSize>1</eventBufferSize>
  ```

- Do the following configurations in the `<CEP_HOME>/repository/conf/data-bridge/data-agent-config.xml` file for both Thrift and Binary agents.
The same configurations should be done on the client side.

**Monitoring WSO2 CEP**

The following sections explain the administrative function of monitoring WSO2 CEP.

- JMX-Based Monitoring
- System Statistics
- Event Monitoring
- SOAP Tracer
- SOAP Message Flows
- Monitoring with WSO2 Carbon Metrics

**JMX-Based Monitoring**

Java Management Extensions (JMX) is a technology that lets you implement management interfaces for Java applications. A management interface, as defined by JMX, is composed of named objects called MBeans (Management Beans). MBeans are registered with a name (an ObjectName) in an MBeanServer. To manage a resource or many resources in your application, write an MBean that defines its management interface. Then register the MBean in your MBeanServer. The content of the MBeanServer can then be exposed through various protocols, implemented by protocol connectors or protocol adapters.

JMX is enabled in WSO2 Carbon by default. Therefore, the WSO2 Complex Event Processor facilitates JMX-based monitoring and provides statistics on the events using JMX technology.

You can change the JMX configuration by editing the `jmx.xml` file that is inside `<CEP_HOME>/repository/conf/etc/` directory. You can configure JMX ports, that are `RMIRegistryPort` and `RMIServerPort` from the `Ports` section of the `carbon.xml`, which is in `<CEP_HOME>/repository/conf/` directory as follows:

```xml
<JMX>
  <!-- The port RMI registry is exposed -->
  <RMIRegistryPort>9999</RMIRegistryPort>
  <!-- The port RMI server should be exposed -->
  <RMIServerPort>11111</RMIServerPort>
</JMX>
```

If JMX is enabled, when the server starts up, it will print the JMX Server URL on the console as follows:

```
```

**Using Jconsole to Manage and Monitor CEP**

Jconsole is a JMX-compliant monitoring tool. It is shipped with the Java Development Kit (JDK) 1.5 and newer versions and can be found in `<JDK_HOME>/bin/` directory.
Start the tool by simply typing `jconsole` in a command (shell) prompt from `<JDK_HOME>/bin/` directory.

Once the console is opened, connect to Carbon by providing the following URL as the Remote Process URL. Type in the default username and password combination `admin/admin` to log in. Note that this is the same administrator account that is used to log into the CEP Management Console. Any user having the admin role can log in to JMX.

The **Java Monitoring & Management Console** opens. The first window looks similar to the following:

Click **MBeans** tab to see the `org.wso2.carbon` domain that contains the MBeans as follows:
**ServerAdmin MBean**

The ServerAdmin MBean is used for administering the Complex Event Processor instance.

There are several Server attributes such as Alive, ServerStatus, ServerData and ServerVersion. The ServerStatus attribute can take any of the following values:

- RUNNING
- SHUTTING_DOWN
- RESTARTING
- IN_MAINTENANCE
The ServerAdmin MBean has the following operations:
- **shutdown** - forcefully shutdown the server.
- **restart** - forcefully restart the server.
- **restartGracefully** - wait till all current requests are served and then restart.
- **shutdownGracefully** - wait till all current requests are served and then shutdown.
- **startMaintenance** - switch the server to maintenance mode. No new requests will be accepted while the server is in maintenance.
- **endMaintenance** - switch the server to normal mode, if it was switched to maintenance mode earlier.

**StatisticsAdmin MBean**

The Statistics MBean is used for monitoring system and server statistics. The attributes of it are as follows:

- **AvgSystemResponseTime** - The average response time for all the services deployed in the system. The beginning of the measurement is the time on which the server started.
- **MaxSystemResponseTime** - The maximum response time for all the services deployed in the system. The beginning of the measurement is the time on which the server started.
- **MinSystemResponseTime** - The minimum time for all the services deployed in the system. The beginning of the measurement is the time on which the server started.
- **SystemFaultCount** - The total number of faults that occurred in the system since the server was started.
- **SystemRequestCount** - The total number of requests that has been served by the system since the server was started.
- **SystemResponseCount** - The total number of response that has been sent by the system since the server was started.
Operations available in the StatisticsAdmin MBean are as follows:

- **getServiceRequestCount**(p1:string) - The p1 parameter is the service name. You can get the total number of requests received by this service since it was deployed, using this operation.
- **getServiceResponseCount**(p1:string) - The p1 parameter is the service name. You can get the total number of responses sent by this service since the time it was deployed, using this operation.
- **getServiceFaultCount**(p1:string) - The p1 parameter is the service name. You can get the total number of fault responses sent by this service since the time it was deployed, using this operation.
- **getMaxServiceResponseTime**(p1:string) - The p1 parameter is the service name. You can get the maximum response time of this service since deployment.
- **getMinServiceResponseTime**(p1:string) - The p1 parameter is the service name. You can get the minimum response time of this service since deployment.
- **getAvgServiceResponseTime**(p1:string) - The p1 parameter is the service name. You can get the average response time of this service since deployment.
- **getOperationRequestCount**(p1:string, p2:string) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of requests received by this operation since the time its service was deployed, using this operation.
- **getOperationResponseCount**(p1:string, p2:string) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of responses sent by this operation since the time its service was deployed, using this operation.
- **getOperationFaultCount**(p1:string, p2:string) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of fault responses sent by this operation since the time its service was deployed, using this operation.
- **getMaxOperationResponseTime**(p1:string, p2:string) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the maximum response time of this operation since deployment.
- **getMinOperationResponseTime**(p1:string, p2:string) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the minimum response time of this operation since deployment.
• **getAvgOperationResponseTime**\((p1:\text{string}, p2:\text{string})\) - The p1 parameter is the service name. The p2 parameter is the operation name. You can get the average response time of this operation since deployment.

**ServiceAdmin MBean**

The ServiceAdmin MBean is used for administering services deployed in the Complex Event Processor. The attributes of it are as follows:

• **NumberOfActiveServices** - The number of services that can currently serve requests.
• **NumberOfInactiveServices** - The number of services that have been disabled by an administrator.
• **NumberOfFaultyServices** - The number of services that are faulty.
The operations available in the ServiceAdmin MBean are as follows:

- **startService**(p1:string) - The p1 parameter is the service name. You can activate a service using this operation.
- **stopService**(p1:string) - The p1 parameter is the service name. You can deactivate/disable a service using this operation.
System Statistics

The System Statistics page shows certain statistics related to the WSO2 Complex Event Processor instance. These include free memory, request count, server name, server start time, system up time, active services, total memory, average response time, minimum response time, and maximum response time.

Follow the instructions below to access the System Statistics.

1. Sign in. Enter your user name and password to log on to the Management Console.
2. Click on the Monitor tab and then click System Statistics. The System Statistics page appears.
See detailed information about the statistics panels by visiting the links below:

- Service Summary
- Server Information
- Response Time Graph
- Memory Graph
- Statistics Configuration Panel

**Service Summary**

This panel provides the following information:

- **Average Response Time** - The average amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Minimum Response Time** - The minimum amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Maximum Response Time** - The maximum amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Total Request Count** - The total number of messages received and mediated through the mediation channel.
- **Total Response Count** - The total number of messages sent and mediated through the mediation channel.
- **Total Fault Count** - The number of messages that triggered faults while being mediated through the channel.
- **Active Services** - The number of currently active services.

**Server Information**
This panel provides the following information:

- **Host** - Shows the host number of the server.
- **Server Start Time** - Shows the time when the server started.
- **System Up Time** - Shows the amount of time that the server has been working.
- **Memory Allocated** - Shows the memory capacity of the server.
- **Memory Usage** - Shows the memory capacity used by the server.

**Response Time graph**

This graph shows a temporal variation of the Average Response time.

**Memory graph**

This graph shows a temporal variation of the server Memory.

**Statistics Configuration panel**

Use the **Statistics Configuration** panel to configure the statistics view.

1. Enter values into the appropriate fields:
   - **Statistics Refresh Interval (ms)** - Allows you to specify the statistics refresh rate.
   - **Response Time Graph** - Allows you to specify the X and Y parameters of the Response Time graph.
     - X-Scale (units)
     - X-Width (px)
   - **Memory Graph** - Allows you to specify the X and Y parameters of the Memory graph.
     - X-Scale (units)
     - X-Width (px)

2. Click **Update**.

3. Refresh your page.

   If you want to restore to the previous values after editing, click **Reset**.

   If you want to restore to the default values, click the corresponding button.

**Event Monitoring**

WSO2 CEP provides the following tools to monitor CEP events.

- **Event Tracer**
- **Event Metrics**
SOAP Tracer

The **SOAP Tracer** shows the SOAP messages, the respective SOAP message requests and responses - when the services are deployed on the server, and the invoked operations. By default, this feature is turned off. You have to turn it on to enable tracing.

**Note**

Turning on the Tracer feature affects the performance. This is because SOAP messages are always completely built (deferred building is not possible) and stored in the database. Hence this option should be used with caution.

Follow the instructions below to access the SOAP Tracer.

1. Log into the CEP Management Console and click on the **Monitor** tab.
2. Click on **SOAP Tracer** to open the **SOAP Message Tracer** page.
3. Select **Yes** from the list for the **Enable SOAP Tracing?** parameter. The tracer will show the messages of the operations that were invoked as shown below.
4. The SOAP messages are shown with their time-stamps, service name, operation invoked and the number of requests to the server. The SOAP messages are displayed in the order in which they were received, and the most recent messages appear at the top. When a particular SOAP message is selected, its request and response can be viewed in the **Request** and **Response** panels as shown in the example below.

5. If you want to search for one or more messages containing a specific word, enter the relevant word in the **Filt**
parameter and click **Search**.

**SOAP Message Tracer**

6. Click **Refresh** if you want to refresh the page.
7. If you want to clear all the existing SOAP messages, click **Clear all SOAP messages**.

**SOAP Message Flows**

Message Flows provide graphical or textual views of the globally-engaged handlers in the system at a given time. This functionality is provided by the following feature:

**Name**: WSO2 Carbon - Message Flows Feature

**Identifier**: org.wso2.carbon.message.flows.feature.group

If the above feature is not included in your product by default, you install it using the instructions given in section Feature Management.

Modules use handlers to engage in different message flows at defined phases. You can observe the handlers invoked in each phase of each flow in real time. For example, the Apache/Rampart module defines handlers in the security phase of each flow, which handles the security aspects of the messages that are transferred through these flows. Therefore, if the Rampart module is engaged, you can see the Apache/Rampart handlers in the message flows in real time.

Follow the instructions below to access the Message Flows.

1. Log on to the product's management console and click on the **Monitor** tab.
2. Click Message Flows to open the Message Flows page. The graphical view of the message flows appears by default, as shown below.
This diagram shows the four different flows defined in the system which are as follows.

<table>
<thead>
<tr>
<th>Message Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inflow</td>
<td>A correct message coming into the system.</td>
</tr>
<tr>
<td>Outflow</td>
<td>A correct message going out of the system.</td>
</tr>
<tr>
<td>In Fault Flow</td>
<td>A faulty message coming into the system.</td>
</tr>
<tr>
<td>Out Fault Flow</td>
<td>A faulty message going out of the system.</td>
</tr>
</tbody>
</table>

In each flow, a message passes through a set of phases to reach the service. These phases vary according to the currently engaged modules within the system. The interface displays the current phases in each and every flow as shown in the diagram below.

3. Click the links to get a view of the engaged handlers in each phase. For example, the figure below shows the handlers engaged in the Addressing phase at system start up.

4. To see the text view of message flows, click **Show Text View**. The page with the text view of message flows appears. The textual view provides the name and the fully qualified classes of all handlers within each and every phase as shown below.
Monitoring with WSO2 Carbon Metrics

WSO2 CEP is shipped with JVM Metrics to monitor the CEP server performance using Java Metrics. The Java Metrics library consists of a variety of metrics that can be used for monitoring. With the WSO2 Carbon Metrics API, we have enabled all the metrics that are required to effectively monitor WSO2 products.

The WSO2 CEP 4.2.0 also provides the Event Metrics feature for the in-depth monitoring of throughput, latency, and memory usage of CEP.

The following sections provide a detailed description of how Carbon Metrics is used in CEP for monitoring.

- Enabling Metrics and Storage Types
- Configuring Metrics Properties
- Using JVM Metrics
- Using Event Metrics

Enabling Metrics and Storage Types

Given below are the configurations that should be in place for your CEP server in order to use the metrics feature. You need to first enable metrics for your server and then enable the required storage types (reporters) that are used for storing the metrics data. See the following topics for instructions:

- Enabling metrics
- Configuring the storage of metrics
- Sample configuration

Enabling metrics
To enable metrics for your product, set the Enabled parameter under the Metrics element to true in the <CEP_HOME>/repository/conf/metrics.xml file. Alternatively, you can enable metrics at the time of starting the CEP server by using the following command:

```
-Dmetrics.enabled=true
```

### Configuring the storage of metrics

WSO2 CEP is configured by default to store the information from metrics in the following reporters: JMX, CSV and JDBC. These reporters are configured in the metrics.xml file (stored in the <CEP_HOME>/repository/conf directory). You can disable metrics for individual reporters by setting the Enabled parameter to false.

If you set the the Enabled parameter under the metrics element to false in the metrics.xml file, metrics is disabled for all the reporters and it is not possible to enable metrics for individual reporters.

See the following topics for information on configuring each of the available storage types.

- JMX
- CSV
- JDBC

#### JMX

The following parameters in the metrics.xml file can be used to configure a JMX storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for JMX or not.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

#### CSV

The following parameters in the metrics.xml file can be used to configure a CSV storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for CSV or not.</td>
<td>Boolean</td>
<td>false</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Location</td>
<td>The location where the CSV files are stored.</td>
<td>String</td>
<td>${carbon.home}/repository/logs/metrics/</td>
<td></td>
</tr>
</tbody>
</table>
**PollingPeriod**  
The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 milliseconds.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for JDBC or not.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>DataSourceName</td>
<td>The name of the datasource used.</td>
<td>String</td>
<td>jdbc/WSO2MetricsDB</td>
</tr>
<tr>
<td>PollingPeriod</td>
<td>The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 milliseconds.</td>
<td>Integer</td>
<td>60</td>
</tr>
</tbody>
</table>

**JDBC**

The following parameters in the `metrics.xml` file can be used to configure a JDBC storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScheduledCleanup/Enabled</td>
<td>This parameter specifies whether scheduled cleanup is enabled or not.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>ScheduledCleanup/ScheduledCleanupPeriod</td>
<td>The number of milliseconds that should elapse after a cleanup task before the next clean-up task is carried out.</td>
<td>Integer</td>
<td>86400</td>
</tr>
</tbody>
</table>
### ScheduledCleanup/DaysToKeep

<table>
<thead>
<tr>
<th>The number of days during which the scheduled clean-up task should be carried out.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
</tr>
</tbody>
</table>

If you have enabled JDBC, then you also need to specify a datasource configuration to be used to create the connection between WSO2 CEP and the JDBC data storage system. The `metrics-datasources.xml` is used for configuring this datasource for metrics.

Parameters that can be configured for a datasource are as follows:

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datasources-configuration&gt;</code></td>
<td>xmlns</td>
<td>The root element. The namespace is specified as: <code>xmlns:svns: http://org.wso2.securevault/configuration&quot;</code></td>
</tr>
<tr>
<td><code>&lt;providers&gt;</code></td>
<td></td>
<td>The container element for the datasource providers.</td>
</tr>
<tr>
<td><code>&lt;provider&gt;</code></td>
<td></td>
<td>The datasource provider, which should implement <code>org.wso2.carbon.ndatasource.common.spi.DataSourceReader</code>. The datasources follow a pluggable model in providing datasource type implementations using this approach.</td>
</tr>
<tr>
<td><code>&lt;datasources&gt;</code></td>
<td></td>
<td>The container element for the datasources.</td>
</tr>
<tr>
<td><code>&lt;datasource&gt;</code></td>
<td></td>
<td>The root element of a datasource.</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td></td>
<td>Name of the datasource.</td>
</tr>
<tr>
<td><code>&lt;description&gt;</code></td>
<td></td>
<td>Description of the datasource.</td>
</tr>
<tr>
<td><code>&lt;jndiConfig&gt;</code></td>
<td></td>
<td>The container element that allows you to expose this datasource JNDI datasource.</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td></td>
<td>The JNDI resource name to which this datasource should be bound.</td>
</tr>
<tr>
<td><code>&lt;environment&gt;</code></td>
<td></td>
<td>The container element in which you specify the following properties:</td>
</tr>
<tr>
<td></td>
<td>java.naming.factory.initial: Selects the registry service provider as the initial context.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>java.naming.provider.url: Specifies the location of the registry when the registry is being used as the initial context.</td>
<td></td>
</tr>
<tr>
<td><code>&lt;definition&gt;</code></td>
<td>type</td>
<td>The container element for the data source definition. Set the type attribute to RDBMS, or to custom if you’re creating a custom type. &quot;RDBMS&quot; data source reader expects a “configuration” element with the sub-elements listed below.</td>
</tr>
<tr>
<td><code>&lt;configuration&gt;</code></td>
<td></td>
<td>The container element for the RDBMS properties.</td>
</tr>
<tr>
<td><code>&lt;url&gt;</code></td>
<td></td>
<td>The connection URL to pass to the JDBC driver to establish the connection.</td>
</tr>
</tbody>
</table>
<username> The connection user name to pass to the JDBC driver to establish connection.

$password> The connection password to pass to the JDBC driver to establish connection.

<driverClassName> The class name of the JDBC driver to use.

<maxActive> The maximum number of active connections that can be allocated from this pool at the same time.

<maxWait> Maximum number of milliseconds that the pool waits (when there no available connections) for a connection to be returned before throwing an exception.

<testOnBorrow> Specifies whether objects are validated before being borrowed from pool. If the object fails to validate, it is dropped from the pool, and an attempt to borrow another. When set to true, the validationQuery parameter must be set to non-null string.

<validationQuery> The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can’t throw a SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).

<validationInterval> To avoid excess validation, only run validation at most at this frequency (interval time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).

**Sample configuration**

Shown below is a sample metrics.xml file with the default configurations specifying the types of storages enabled for metrics data. See the above topics for instructions.

```xml
<Metrics xmlns="http://wso2.org/projects/carbon/metrics.xml">
  <!--
  This is the main configuration file for metrics
  -->
  <Metrics xmlns="http://wso2.org/projects/carbon/metrics.xml">
    <!--
    Enable Metrics
    -->
    <Enabled>false</Enabled>
    <!--
    Metrics reporting configurations
    -->
```

<Reporting>
   <JMX>
      <Enabled>true</Enabled>
   </JMX>
   <CSV>
      <Enabled>false</Enabled>
      <Location>${carbon.home}/repository/logs/metrics/</Location>
      <!-- Polling Period in seconds -->
      <PollingPeriod>60</PollingPeriod>
   </CSV>
   <JDBC>
      <Enabled>true</Enabled>
      <!-- Source of Metrics, which will be used to identify each metric in database -->
      <!-- Commented to use the hostname
      <Source>Carbon</Source>
      -->
      <!-- JNDI name of the data source to be used by the JDBC Reporter. This data source should be defined in a *-datasources.xml file in conf/datasources directory. -->
      <DataSourceName>jdbc/WSO2MetricsDB</DataSourceName>
      <!-- Polling Period in seconds -->
      <PollingPeriod>60</PollingPeriod>
      <ScheduledCleanup>
         <!-- Schedule regular deletion of metrics data older than a set number of days. It is strongly recommended that you enable this job to ensure your metrics tables do not get extremely large. Deleting data older than seven days should be sufficient. -->
         <Enabled>true</Enabled>
         <!-- This is the period for each cleanup operation in seconds -->
         <ScheduledCleanupPeriod>86400</ScheduledCleanupPeriod>
         <!-- The scheduled job will cleanup all data older than the specified days -->
         <DaysToKeep>7</DaysToKeep>
      </ScheduledCleanup>
   </JDBC>
</Reporting>
Configuring Metrics Properties

The `<CEP_HOME>/repository/conf/metrics.properties` file specifies properties that correspond to the gauges in the Metrics Dashboard. See the topic on using JVM metrics for details on using the metrics dashboard for JVM metrics. The level defined for a property in this file determines the extent to which the relevant gauge in the dashboard should be updated with information. The different levels that can be defined for properties are as follows:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Designates no informational events.</td>
</tr>
<tr>
<td>Info</td>
<td>Designates informational metric events that highlight the progress of the application at coarse-grained level.</td>
</tr>
<tr>
<td>Debug</td>
<td>Designates fine-grained informational events that are most useful to debug an application.</td>
</tr>
<tr>
<td>Trace</td>
<td>Designates finer-grained informational events than the DEBUG.</td>
</tr>
<tr>
<td>All</td>
<td>Designates all the informational events.</td>
</tr>
</tbody>
</table>

If no specific level is configured for a property in the `metrics.properties` file, the metrics root level applies. The root level is defined as shown in the following example in the `metrics.properties` file.

```
metrics.rootLevel=OFF
```

If you want to change the current root level, you can also use the following command.

```
-Dmetrics.rootLevel=INFO
```

The levels in `metrics.properties` file can be configured to any hierarchy. However, if the level defined for an individual property is different to the level defined for its parent in the hierarchy, the level defined for the individual property overrules that of the parent. For example, if we have `metric.level.jvm.memory=INFO` in the `<CEP_HOME>/repository/conf/metrics.properties` file, all metrics under `jvm.memory` have `INFO` as the configured level. However, if you have `metric.level.jvm.memory.heap=TRACE`, the `TRACE` level would apply for the `jvm.memory.heap` property even though it is a child property of `jvm.memory`.

The properties that are included in this file by default are as follows:

- JVM's direct and mapped buffer pools
- Class loading
- GC
- Memory
- Operating system load
- Threads

JVM's direct and mapped buffer pools
### Class loading

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.class-loading</td>
<td>INFO</td>
<td>The gauge showing the number of classes currently loaded for the JVM.</td>
</tr>
</tbody>
</table>

### GC

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.gc</td>
<td>DEBUG</td>
<td>The gauge for showing garbage collection and memory usage. Monitoring this allows you to identify memory leaks and memory thrash, which have a negative impact on performance.</td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.memory</td>
<td>INFO</td>
<td>The gauge for showing the used and committed memory in WSO2 CEP.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.heap</td>
<td>INFO</td>
<td>The gauge for showing the used and committed heap in WSO2 CEP.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.non-heap</td>
<td>INFO</td>
<td>The gauge for showing the used code cache and used CMS Perm Gen in WSO2 CEP.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.total</td>
<td>INFO</td>
<td>The gauge for showing the total memory currently available for the JVM.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.pools</td>
<td>OFF</td>
<td>The gauge for showing the used and available memory for JVM in the memory pool.</td>
</tr>
</tbody>
</table>

### Operating system load

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.os</td>
<td>INFO</td>
<td>The gauge for showing the current load imposed by the JVM on the operating system.</td>
</tr>
</tbody>
</table>

### Threads

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.threads</td>
<td>OFF</td>
<td>The parent property of all the gauges relating to the JVM thread pool. The metric level defined for this property applies to all the rest of the properties in this table. The metric level set via this property to a child property can be overruled if a different level is set for it.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Visibility</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>metric.level.jvm.threads.count</td>
<td>DEBUG</td>
<td>The gauge for showing the number of active and idle threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.daemon.count</td>
<td>DEBUG</td>
<td>The gauge for showing the number of active daemon threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.blocked.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads that are currently blocked in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.deadlock.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads that are currently deadlocked in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.new.count</td>
<td>OFF</td>
<td>The gauge for showing the number of new threads generated in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.runnable.count</td>
<td>OFF</td>
<td>The gauge for showing the number of runnable threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.terminated.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads terminated from the JVM thread pool since you started running the WSO2 CEP instance.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.timed_waiting.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads in the Timed_Waiting state.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.waiting.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads in the Waiting state in the JVM thread pool. One or more other threads are required to perform certain actions before these threads can proceed with their actions.</td>
</tr>
</tbody>
</table>

**Using JVM Metrics**

JVM metrics are Java metrics enabled by default in WSO2 products for the purpose of monitoring general statistics related to the server performance. Follow the procedure below to view the JVM metrics dashboard for a WSO2 product.

For detailed instructions to enable/disable JVM metrics and to configure metrics, see [Enabling Metrics and Storage Types](#).

1. Log into the Management Console of the WSO2 product. Click **Monitor -> Metrics -> JVM Metrics** to open the **View Metrics** page.
2. Specify the source for the JVM metrics by selecting a value from the drop-down list for the **Source** parameter in the top panel.
3. Specify the time interval for which the statistics should be displayed in the dashboard by selecting a value from the following drop-down list in the top panel.
4. Click the required buttons opposite **Views** in the top panel to select the types of information you want to view in the dashboard, and refresh the web page.
Statistics corresponding to each button can be viewed as follows:

- **CPU**
  Click this button to view statistics relating to the CPU as shown below.

![CPU Load Graph](image)

- **Memory**
  Click **Memory** to view statistics relating to the memory as shown below.

![Memory Graph](image)
• **Threading**
  Click **Threading** to view statistics relating to threading as shown below.

• **Class Loading**
  Click **Class Loading** to view statistics relating to class loading as shown below.

• **File Descriptor**
  Click **File Descriptor** to view information relating to the file descriptor count as shown below.
Using Event Metrics

Event Metrics feature provides a drilled down visualisation of metrics relating to CEP specific artifacts to carry out in-depth monitoring of throughput, latency, and memory usage of CEP. It is disabled by default to avoid the excessive processing of events and execution plan from causing a system overhead.

Prerequisites

Before you use event metrics, both JVM metrics and CEP component-specific metrics need to be enabled as follows.

- To enable metrics for the CEP, set the Enabled parameter under the Metrics element to true in the `<CEP_HOME>/repository/conf/metrics.xml` file. Alternatively, you can enable metrics at the time of starting the CEP server by using the following command:

  -Dmetrics.enabled=true

  For detailed instructions to enable/disable metrics and to configure metrics, see Enabling Metrics and Storage Types.

- To generate metrics related statistics, set the StatisticsReporterDisabled property to false in the `<CEP_HOME>/repository/conf/carbon.xml` file as shown below.

  `<StatisticsReporterDisabled>true</StatisticsReporterDisabled>`

- To enable event metrics for specific event receivers, event publishers, and execution plans, enable statistics for the relevant components by clicking the Enable Statistics link. e.g., The following image shows how to enable statistics for an execution plan.
Monitoring event metrics

There are two methods to view event metrics:

- Monitoring Metrics using the Management Console
- Monitoring Metrics using the JConsole

Both methods generate the same metrics. The difference between the two methods is, when there are multiple tenants, the CEP Management Console allows you to view metrics for the currently logged tenant as well as other tenants whereas the JConsole only allows you to view metrics for the currently logged tenant.

**Monitoring Metrics using the Management Console**

Follow the procedure below to access event metrics via the Management Console.

1. Log into the CEP Management Console. For detailed instructions, see Running the Product.
2. In the Monitor tab, click Event Metrics. The View All Metrics page is displayed as shown in the example below.

Alternatively, you can click Metrics to open the View All Metrics page.

If you click Event Metrics, the View All Metrics page displays only event metrics. If you click Metrics, the View All Metrics page displays both JVM metrics and event metrics.

Click on a link displayed under Sub Levels to view a specific metric as shown in the example below.
Monitoring Metrics using the JConsole

Follow the procedure below to monitor event metrics via the JConsole.

1. Start the WSO2 CEP server.
2. Start the JConsole. For detailed instructions to start the JConsole to monitor the CEP, see JMX-Based Monitoring.
3. Click the MBeans tab in the JConsole window. Then expand the org.wso2.carbon.metrics domain.
4. Click on the relevant MBeans under the `org.wso2.carbon.metrics` domain to view the required metrics.

**Deployment Automation Guide**

Automating the deployment process is a common practice in production and development environments. There are many tools to perform this task. Puppet is our recommended tool to perform deployment automation. You can find the necessary puppet modules that are relevant to WSO2 CEP 4.0.0 [here](#). The following sections explain how you can refer/use these puppet modules when developing your own puppet modules for WSO2 CEP.

**WSO2 CEP Puppet Modules**

These puppet modules can be used to install WSO2 CEP in all its deployment modes (i.e. standalone mode, high availability mode and distributed mode). Also, it contains the necessary modules to install an Apache Storm cluster that is required for the distributed deployment of CEP. It also has enabled deployment of 3 main deployment patterns of wso2 CEP easily using Hiera to store deployment parameters.

**Setup**

**CEP Module**

To install CEP in its normal mode (i.e. standalone mode) with default parameters:

```python
include 'cep'
```

To install CEP in a specific role with default parameters:

```python
include 'cep:hanode'
include 'cep:distributednode'
```
Storm Module

To install each server of a storm cluster with default parameters:

```plaintext
include 'storm::nimbus'
include 'storm::supervisor'
include 'storm::zookeeper'
include 'storm::stormui'
```

References

Classes

- cep: Installs and manages the CEP in standalone mode.
- cep:hanode: Installs and configures the CEP as a worker in high availability(HA) mode.
- cep:distributednode: Installs and configures the CEP as a worker or a manager in distributed mode.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of the WSO2 CEP server. These scripts can only be used for version 4.0.0 onwards. (e.g. 4.0.0)</td>
</tr>
<tr>
<td>owner</td>
<td>The owner of the CEP installation folder (e.g. root).</td>
</tr>
<tr>
<td>group</td>
<td>The group of the CEP installation folder (e.g. root).</td>
</tr>
<tr>
<td>maintenance_mode</td>
<td>The mode of the installation. Possible values are as follows.</td>
</tr>
<tr>
<td></td>
<td>refresh: Stops the server if running and restarts it.</td>
</tr>
<tr>
<td></td>
<td>new: Deletes the existing installation, and redeploy and starts the server.</td>
</tr>
<tr>
<td></td>
<td>zero: Downloads a fresh copy of the wso2 CEP pack and perform a fresh deployment.</td>
</tr>
<tr>
<td>offset</td>
<td>The port offset of the carbon server.</td>
</tr>
<tr>
<td>clustering</td>
<td>This indicates whether clustering is enabled or not. e.g., If the value is true, clustering is enabled.</td>
</tr>
<tr>
<td><strong>depsync</strong></td>
<td>This indicates whether deployment synchronisation is enabled or not. e.g., If the value is false, deployment synchronisation is disabled.</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>local_member_port</strong></td>
<td>The local member port to be used in the Hazelcast cluster.</td>
</tr>
<tr>
<td><strong>members</strong></td>
<td>The members of the Hazelcast cluster. The hostname of each member with the local member port of each host should be specified.</td>
</tr>
<tr>
<td><strong>membership_scheme</strong></td>
<td>The Hazelcast cluster membership schema.</td>
</tr>
<tr>
<td><strong>ha_eventsynch_port</strong></td>
<td>The port used for syncing events among two CEP nodes in HA mode.</td>
</tr>
<tr>
<td><strong>ha_mgt_port</strong></td>
<td>The port on which the HA management service is run when the CEP is deployed in HA mode.</td>
</tr>
<tr>
<td><strong>manager_port</strong></td>
<td>The port on which the HA management service is run when the CEP is deployed in distributed mode.</td>
</tr>
<tr>
<td><strong>managers</strong></td>
<td>CEP instances acting as managers in distributed mode. Hostname and the manager service port of each manager should be specified.</td>
</tr>
<tr>
<td><strong>worker</strong></td>
<td>This indicates whether the CEP instance is acting as a worker in HA or distributed mode.</td>
</tr>
<tr>
<td><strong>manager</strong></td>
<td>This indicates whether the CEP instance is acting as a manager in distributed mode.</td>
</tr>
<tr>
<td><strong>presenter</strong></td>
<td>This indicates whether the CEP instance is acting as a presenter in HA or distributed mode.</td>
</tr>
<tr>
<td><strong>presenter_port</strong></td>
<td>The port which presenter instances use to receive events.</td>
</tr>
<tr>
<td><strong>registry_db_connection_url</strong></td>
<td>The connection URL of the registry database (e.g., jdbc:mysql://192.168.5 7.170:3306/regdb?autoReconnect=true).</td>
</tr>
<tr>
<td><strong>registry_db_user</strong></td>
<td>The username to be used to connect to the registry database.</td>
</tr>
<tr>
<td><strong>registry_db_password</strong></td>
<td>The password to log into the registry database.</td>
</tr>
<tr>
<td><strong>registry_db_driver_name</strong></td>
<td>The driver class name to be used to connect to the registry database (e.g., com.mysql.jdbc.Driver).</td>
</tr>
<tr>
<td><strong>userstore_db_connection_url</strong></td>
<td>The connection URL of userstore database.</td>
</tr>
</tbody>
</table>
userstore\_db\_user | The username to be used to connect to the user store database.
--- | ---
userstore\_db\_password | The password to log into the user store database.
userstore\_db\_driver\_name | The driver class name to be used to connect to the user store database.

### Clustered Deployment

You can install multiple instances of WSO2 products in a cluster to ensure that if one instance becomes unavailable or is experiencing high traffic, another instance will seamlessly handle the requests. For complete information on the clustered deployment, go to [Clustering CEP 4.2.0](#).

### User Management

User kernel of Carbon has the following features:

- The concept of single user store, which is either external or internal.
- Apache LDAP is the default, embedded user store.
- Ability to configure multiple user stores.
- Ability to operate in read-only mode on your organization's LDAP and Active Directory userstores.
- Ability to operate in read-write mode on internal and external user stores.
- Supports any custom realm.
- Roles can contain users from external user stores.
- Improved configurability for external user stores.
- Capability to read/write roles from/to LDAP/Active Directory user stores.
- Implements management permission through the management console UI.

The user core in all WSO2 Carbon-based products is defined in `$PRODUCT\_HOME/repository/conf/user-mgt.xml` file.

Visit the following pages to learn more information about the user core kernel usage:

- Introduction to User Management
- Adding and Managing Users and Roles
- Realm Configuration
- Default JDBC User Store Configuration
- Changing the RDBMS
- Default LDAP User Store Configuration
- Configuring Secondary User Stores

### Introduction to User Management

User management is a mechanism which involves defining and managing users, roles and their access levels in a system. A user management dashboard or console provides system administrators a holistic view of a system's active user sessions, their log-in statuses, the privileges of each user and their activity in the system, enabling the system admins to make business-critical, real-time security decisions. A typical user management implementation involves a wide range of functionality such as adding/deleting users, controlling user activity through permissions, managing user roles, defining authentication policies, managing external user stores, manual/automatic log-out, resetting user passwords etc.

Any user management system has users, roles, user stores and user permissions as its basic components.

*Users*
Users are consumers who interact with your organizational applications, databases or any other systems. These users can be a person, a device or another application/program within or outside of the organization's network. Since these users interact with internal systems and access data, the need to define which user is allowed to do what is critical to most security-conscious organizations. This is how the concept of user management developed.

**User Stores**

A user store is the database where information of the users and/or user roles is stored. User information includes log-in name, password, first name, last name, e-mail etc.

The user stores of all WSO2 Carbon-based products are embedded H2 databases except for WSO2 Identity Server, which has an embedded LDAP as its user store. In Carbon, permission is stored in a separate database called the user management database, which by default is H2. However, users have the ability to connect to external user stores as well.

The user stores of Carbon products can be configured to operate in either one of the following modes.

- User store operates in read/write mode - In Read/Write mode, WSO2 Carbon reads/writes into the user store.
- User store operates in read only mode - In Read Only mode, WSO2 Carbon guarantees that it does not modify any data in the user store. Carbon maintains roles and permissions in the Carbon database but it can read users/roles from the configured user store.

**Permission**

A permission is a ‘delegation of authority’ or a ‘right’ assigned to a user or a group of users to perform an action on a system. Permissions can be granted to or revoked from a user/user group/user role automatically or by a system administrator. For example, if a user has the permission to log-in to a system, then the permission to log-out is automatically implied without the need of granting it specifically.

**User Roles**

A user role is a consolidation of several permissions. Instead of associating permissions with a user, admins can associate permissions with a user role and assign the role to users. User roles can be reused throughout the system and prevents the overhead of granting multiple permissions to each and every user individually.

**User Management in WSO2 Carbon**

User management comes bundled with the WSO2 Carbon platform and facilitates the management and control of user accounts and roles at different levels. Since it is integrated into the core Carbon platform, user management capability is available by default in all WSO2 Carbon-based products.

The user store of Carbon products can be configured to operate in either one of the following modes.

- User store operates in read/write mode - In Read/Write mode, WSO2 Carbon reads/writes into the user store.
- User store operates in read only mode - In Read Only mode, WSO2 Carbon guarantees that it does not modify any data in the user store. Carbon maintains roles and permissions in the Carbon database but it can read users/roles from the configured user store.

The user kernel of WSO2 Carbon has the following features:

- The concept of single user store which is either external or internal.
- Ability to operate in read-only/read-write mode on your company's LDAP user stores.
- Ability to work with Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS) in read write mode.
- Supports any custom realm.
- Roles can contain users from external user stores.
- Improved configuration capability for external user stores.
- Capability to read roles from LDAP/Active Directory user stores.
Adding and Managing Users and Roles

Before you begin, note the following:

- Only system administrators can add, modify and remove users and roles. To set up administrators, see Realm Configuration.
- Your product has a primary user store where the users/roles that you create using the management console are stored by default. It’s default RegEx configurations are as follows. RegEx configurations ensure that parameters like the length of a user name/password meet the requirements of the user store.

```plaintext
PasswordJavaRegEx-------- ^[\S]{5,30}$
PasswordJavaScriptRegEx-- ^[\S]{5,30}$
UsernameJavaRegEx-------- ^~!#$;%*+={}\{3,30}$
UsernameJavaScriptRegEx-- ^[\S]{3,30}$
RolenameJavaRegEx-------- ^~!#$;%*+={}\{3,30}$
RolenameJavaScriptRegEx-- ^[\S]{3,30}$
```

When creating users/roles, if you enter a username, password etc. that does not conform to the RegEx configurations, the system throws an exception. You can either change the RegEx configuration or enter values that conform to the RegEx. If you change the default user store or set up a secondary user store, configure the RegEx accordingly under the user store manager configurations in user-mgt.xml file.

Go to the relevant topic listed below for details:

- Adding a new user and assigning roles
- Importing users
- Adding a user role
- Changing the current user's password
- Deleting an existing user

Adding a new user and assigning roles

Follow the instructions below to add a new user account and configure its role.

1. Log on to the product's Management Console. In the Configure menu, click Users and Roles.
2. Click **Users** from the **User Management** page that opens.

   The **User** link is only visible to users with **Administrator** permission.

3. Click **Add New User**.
4. The **Add User** page opens. Enter the user name and password. The **Domain** drop-down list contains all user stores configured for this product instance. By default, you only have the PRIMARY user store. To configure secondary user stores, see **Configuring Secondary User Stores**.

5. If you want to add a user with the default **Everyone** role, click **Finish** now. Else, click **Next** to define a user role other than the default.
6. If you proceed to the next step, select the roles to be assigned to the user and **Finish**.
7. The new user appears on the **Users** list.
8. You can change the user's password, roles or delete using the links associated with it.

You cannot change the user name of an existing user.

**Importing users**

In addition to manually adding individual users, you can import multiple users in bulk if you have exported them to a comma-separated values (.csv) file or Microsoft Excel (.xls) file. It is possible to import the **username** and **password** directly from the CSV/Excel to the product. Other user attributes can be imported only if claim URLs are defined for such attributes in the product. For example, consider that you have claim URLs defined for your product as shown below. These will allow you to import the user's **email address**, **full name**, **last name**, **given name** and **role** in addition to the **username** and **password**.

- http://wso2.org/claims/emailaddress
- http://wso2.org/claims/fullname
- http://wso2.org/claims/givenname
- http://wso2.org/claims/lastname
- http://wso2.org/claims/role

The username, password and other attributes (claim URLs) that you import should be given in a CSV file as shown below. Note that the first line of the file will not be imported considering that it is not a username.

```
UserName,Password,Claim
name1,password1,http://wso2.org/claims/emailaddress=name1@gmail.com,http://wso2.org/claims/fullname=fullName1
name2,password2,http://wso2.org/claims/emailaddress=name2@gmail.com,http://wso2.org/claims/fullname=fullName2
name3,password3,http://wso2.org/claims/emailaddress=name3@gmail.com,http://wso2.org/claims/fullname=fullName3
```

This is only supported if you have configured your user store as JDBCUserStoreManager. See [here](#) for information on how to do this.

1. On the **Users** screen, click **Bulk Import Users**.
2. Browse and select the file that contains the user data.
3. Specify a default password to assign to all the users you are importing and click **Finish**. This password is valid for only 24 hours, so you should inform your users that they must log in and change their password within 24 hours.

**Adding a user role**

Roles contain permissions for users to manage the Server. You can create different roles with various combinations of permissions and assign them to a user or a group of users. Through the management console, you can also edit and delete an existing user role.
Follow the instructions below to add a user role.

1. Log on to the product's Management Console. In the **Configure** menu, click **Users and Roles**.
2. Click **Roles** from the **User Management** page that opens.
3. Click on **Add New Role**.
4. Enter the name for the role and click **Next**. The **Domain** drop-down list contains all user stores configured for this product instance. By default, you only have the PRIMARY user store. To configure secondary user stores, see Configuring Secondary User Stores.

**Step 1 : Enter role details**

- **Domain**: PRIMARY
- **Role Name**

You can also click **Finish**, in which case, the new role will be created with default permissions (none) and no assigned users.
5. If you proceed, select permissions for the new role and click **Next**.

**Step 2 : Select permissions to add to Role**

- **Expand all**
- **Collapse all**

6. Select the users to be assigned to the role. You can conduct a search by name, or view all users by entering "*" into the search field.
7. Click **Finish**.
8. The new role appears under roles. Using the links associated with it, you can rename, edit permissions, users and delete the role.

<table>
<thead>
<tr>
<th>Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td><img src="image" alt="Assign Users" /> <img src="image" alt="View Users" /></td>
</tr>
<tr>
<td>FullAccess</td>
<td><img src="image" alt="Rename" /> <img src="image" alt="Permissions" /> <img src="image" alt="Assign Users" /> <img src="image" alt="View Users" /> <img src="image" alt="Delete" /></td>
</tr>
</tbody>
</table>

**When adding roles to external user stores**

- Some external user stores do not allow you to create empty roles. In that case, selecting users who belong to a role is mandatory.
- If you connect to an external user store in read only mode, you can read existing roles from it but you cannot edit/delete the roles. In this case, you can still create new roles which are editable and can be
managed internally.

- If you connect to an external user store in read/write mode, you can edit the roles in the external user store as well.

**Changing the current user's password**

Follow the instructions below to change the password of the user currently logged in.

1. Log on to the product's Management Console. In the **Configure** menu, click **Users and Roles**.
2. The **User Management** page opens. Click on the **Change My Password**.

3. The **Change Password** page appears. Populate the required fields and click **Change**.

If a user has forgotten the current password, they need to contact the administrator who can reset it without the current password.

**Deleting an existing user**

Follow the instructions below to delete a user.

- Deleting a user cannot be undone.

1. On the **Configure** tab in the management console, click **Users and Roles**.
2. Click **Users**. This link is only visible to users with the Admin role.
3. In the **Users** list, click **Delete** next to the user you want to delete, and then click **Yes** to confirm the operation.

**Realm Configuration**

<PRODUCT_HOME>/repository/conf/user-mgt.xml file's main configuration section has the following information:
<Configuration>
  <AdminRole>admin</AdminRole>
  <AdminUser>
    <UserName>admin</UserName>
    <Password>admin</Password>
  </AdminUser>
  <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role sees the registry root -->
  <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
  <Property name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
</Configuration>

The main elements of the Realm Configuration are as follows.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;AdminRole&gt;</td>
<td>Admin's role name. This role has permission to carry out any action related to the Management Console. If the user store is read-only, then this role is added to the system as a special internal role where users are from an external user store.</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;&lt;UserName&gt;</td>
<td>Admin user's username. If the user store is read-only, the admin user must exist in the user store. Otherwise the system won't start.</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;&lt;Password&gt;</td>
<td>Admin user's password. If the user store is read-only, this element and its value are ignored.</td>
</tr>
<tr>
<td>&lt;EveryOneRoleName&gt;</td>
<td>Everyone role name. All users in the system belong to this role.</td>
</tr>
</tbody>
</table>

The main properties of the user Realm Configuration can be explained as follows. It mainly contains details for the database connection.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSource</td>
<td>Data sources are configured in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml</code> file. This property indicates the relevant data source configuration for the User Management Database.</td>
</tr>
<tr>
<td>MultiTenantRealmConfigBuilder</td>
<td>Tenant Manager specific realm config parameter. Can be used to build different types of realms for the tenant.</td>
</tr>
</tbody>
</table>

Default JDBC User Store Configuration
The default JDBC user store reads/writes into the internal database of the Carbon server. Internal JDBC user stores can be configured using `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file's `JDBCUserStoreManager` configuration section.

The default configuration is shown below. Change the values according to your requirements.

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="ReadOnly">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="IsEmailUserName">false</Property>
    <Property name="DomainCalculation">default</Property>
    <Property name="PasswordDigest">SHA-256</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordJavaRegEx">^[\S]{5,30}$</Property>
    <Property name="PasswordJavaScriptRegEx">^[\S]{5,30}</Property>
    <Property name="UsernameJavaRegEx">^[^-!#$;\^*+={}|\&lt;&gt;,\']{3,30}$</Property>
    <Property name="UsernameJavaScriptRegEx">^[\S]{3,30}</Property>
    <Property name="RolenameJavaRegEx">^[^-!#$;\^*+={}|\&lt;&gt;,\']{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">^[\S]{3,30}</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>
```

The main elements of the above configuration are described below.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TenantManager</td>
<td>Includes the location of the tenant manager.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Indicates whether the user store of this realm operates in the user read only mode or not.</td>
</tr>
<tr>
<td>MaxUserNameListLength</td>
<td>Maximum number of users retrieved at once by user real.</td>
</tr>
<tr>
<td>IsEmailUserName</td>
<td>Indicates whether Email is used as user name (apply when realm operates in read only mode).</td>
</tr>
<tr>
<td>DomainCalculation</td>
<td>Can be either default or custom (apply when realm operates in read only mode).</td>
</tr>
<tr>
<td>PasswordDigest</td>
<td>Digesting algorithm of the password. Has values such as PLAIN_TEXT, SHA etc.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StoreSaltedPassword</td>
<td>Indicates whether to salt the password.</td>
</tr>
<tr>
<td>UserNameUniqueAcrossTenants</td>
<td>An attribute used for multi-tenancy.</td>
</tr>
<tr>
<td>PasswordJavaRegEx</td>
<td>A regular expression to validate passwords. By default, strings having length 5 to 30 non-empty characters are allowed.</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>The regular expression used by the font-end components for password validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings having length 5 to 30 non-empty characters are allowed.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the font-end components for username validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression to validate rolenames. By default, strings having length 5 to 30 non-empty characters are allowed.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the font-end components for rolename validation.</td>
</tr>
<tr>
<td>UserRolesCacheEnabled</td>
<td>This is to indicate whether to cache the role list of a user. By default it is 'true'. Set it to 'false' if user-roles are changed by external means and those changes should be instantly reflected in the carbon instance.</td>
</tr>
</tbody>
</table>

In addition to the above properties, set the following also in `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

1. **MultiTenantRealmConfigBuilder** property should be set to `org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder`. For example:

   ```xml
   <Property name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
   
   This property is described in section Realm Configuration.

2. Add a property by the name **passwordHashMethod** to `JDBCUserStoreManager` default configuration shown above and set the value to SHA or PLAIN_TEXT. For example:

   ```xml
   <UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
   ...
   <Property name="passwordHashMethod">SHA</Property>
   ...
   </UserStoreManager>
   
   3. Go to `<PRODUCT_HOME>/repository/conf/tenant-mgt.xml` file and comment out the **CommonHybridLDAPTenantManager**, which is used by default and uncomment **JDBCCTenantManager**. For example:
Changing the RDBMS

The default database of user manager is the H2 database shipped by the WSO2 Carbon based products. You can configure it to point to databases by different vendors.

1. Add the JDBC driver to the classpath by dropping the JAR into `<PRODUCT_HOME>/repository/component/s/lib`.

2. Change values of properties given in on the Realm Configuration page appropriately.

3. Create the database by running the relevant script in `<PRODUCT_HOME> /dbscripts/` and start the server as:
   - For Linux:
     ```
     sh wso2server.sh
     ```
   - For Windows:
     ```
     wso2server.bat
     ```

Or start the server as:
   - For Linux:
     ```
     sh wso2server.sh -Dsetup
     ```
   - For Windows:
     ```
     wso2server.bat -Dsetup
     ```

Default LDAP User Store Configuration

`<PRODUCT_HOME>/repository/conf/user-mgt.xml` file has a default configuration for the internal LDAP user store, which is embedded ApacheDS LDAP.

1. Enable `<ApacheDSUserStoreManager>` element in user-mgt.xml file. When it is enabled, the user manager reads/writes into the default LDAP user store of Carbon.

2. The default configuration for the internal LDAP user store in user-mgt.xml file is as follows. Change the values according to your requirement.
Main elements of the configuration can be explained as follows.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionURL</td>
<td>Connection URL to the ldap server. In the case of default LDAP in carbon, port is mentioned in carbon.xml and a reference to that port is mentioned in the above configuration.</td>
</tr>
<tr>
<td>ConnectionName</td>
<td>This should be the DN (Distinguish Name) of the admin user in LDAP.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ConnectionPassword</td>
<td>Password of the admin user.</td>
</tr>
<tr>
<td>passwordHashMethod</td>
<td>Password Hash method when storing user entries in LDAP.</td>
</tr>
<tr>
<td>UserNameListFilter</td>
<td>Filtering criteria for listing all the user entries in LDAP.</td>
</tr>
<tr>
<td>UserEntryObjectClass</td>
<td>Object Class used to construct user entries. In the case of default LDAP in carbon, it is a custom object class defined with the name- ‘wso2Person’</td>
</tr>
<tr>
<td>UserSearchBase</td>
<td>DN of the context under which user entries are stored in LDAP.</td>
</tr>
<tr>
<td>UserNameSearchFilter</td>
<td>Filtering criteria for searching a particular user entry.</td>
</tr>
<tr>
<td>UserNameAttribute</td>
<td>Attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, uid and etc.</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>Policy that defines the password format.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the font-end components for username validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings having length 5 to 30 non-empty characters are allowed.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the font-end components for rolename validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression to validate rolenames. By default, strings having length 5 to 30 non-empty characters are allowed.</td>
</tr>
<tr>
<td>ReadLDAPGroups</td>
<td>Specifies whether groups should be read from LDAP.</td>
</tr>
<tr>
<td>WriteLDAPGroups</td>
<td>Specifies whether groups should be written to LDAP.</td>
</tr>
<tr>
<td>EmptyRolesAllowed</td>
<td>Specifies whether underlying LDAP user store allows empty groups to be created. In the case of Idap in carbon, the schema is modified such that empty groups are allowed to be created. Usually LDAP servers do not allow to create empty groups.</td>
</tr>
<tr>
<td>GroupSearchBase</td>
<td>DN of the context under which user entries are stored in LDAP.</td>
</tr>
<tr>
<td>GroupNameListFilter</td>
<td>Filtering criteria for listing all the group entries in LDAP.</td>
</tr>
<tr>
<td>GroupEntryObjectClass</td>
<td>Object Class used to construct user entries.</td>
</tr>
<tr>
<td>GroupNameSearchFilter</td>
<td>Filtering criteria for searching a particular group entry.</td>
</tr>
<tr>
<td>GroupNameAttribute</td>
<td>Attribute used for uniquely identifying a user entry.</td>
</tr>
<tr>
<td>MembershipAttribute</td>
<td>Attribute used to define members of LDAP groups.</td>
</tr>
<tr>
<td>UserRolesCacheEnabled</td>
<td>This is to indicate whether to cache the role list of a user. By default it is ‘true’. Set it to ‘false’ if user-roles are changed by external means and those changes should be instantly reflected in the carbon instance.</td>
</tr>
<tr>
<td>UserDNPattern</td>
<td>The pattern for user's DN. It can be defined to improve the LDAP search. When there are many user entries in the LDAP, defining a &quot;UserDNPattern&quot; provides more impact on performances as the LDAP does not have to travel through the entire tree to find users.</td>
</tr>
</tbody>
</table>
Configuring Secondary User Stores

The default configurations of WSO2 products have a single, embedded user store (primary user store). If required, you can configure WSO2 products to connect to several secondary user stores as well. After configuration, users from different stores can log in and perform operations depending on their roles/permissions. You can also configure your own customized user stores and connect them with the products as secondary stores.

The topics below explain how to configure secondary user stores manually or using the management console:

- Configuring using the management console
- Configuring manually
- Related topics

Before you begin:

If you are setting up a database other than the default H2 that comes with the product to store user information, select the script relevant to your database type from the \<PRODUCT_HOME>/dbscripts folder and run it on your database. It creates the necessary tables.

Configuring using the management console

1. Log in to the management console and click User Store Management sub menu under Configure menu.
2. The User Store Management page opens. Initially, there are no secondary user stores.

   Note: You cannot update the PRIMARY user store at run time, so it is not visible on this page.

3. Click Add Secondary User Store.

4. In the User Store Manager Class list, select the type of user store you are creating:

<table>
<thead>
<tr>
<th>User store manager</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</td>
<td>Use ReadOnlyLDAP user stores.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</td>
<td>Use ReadWriteLDAP read and write operations.</td>
</tr>
</tbody>
</table>
You can also populate this drop-down list with custom user store manager implementations by adding them to the server. A sample custom user store manager can be found in the repository.

5. Enter a unique domain name with no underscore (_) characters, and optionally enter a description for this user store.

6. Enter values for the properties, using the descriptions in the Descriptions column for guidance. The properties that appear vary based on the user store manager class you selected, and there may be additional properties in an Optional or Advanced section at the bottom of the screen. See the related topics for descriptions of user store properties.

7. Ensure that all the mandatory fields are filled and a valid domain name is given and click Add.

8. A message appears saying that the user stores are being added.
8. Refresh the page after a few seconds to check the status.

9. If the new user store is successfully added, it will appear in the User Store Management page.

10. After adding to the server, you can edit the properties of the new secondary user store and enable/disable it in a dynamic manner.

**Configuring manually**

By default, the configuration of the primary user store is saved in the `user-mgt.xml` file. When you create a secondary user store using the management console as explained above, its configuration is saved to an XML file with the same name as the domain name you specify. Alternatively, you can create this XML file manually and save it as follows:

- When you configure multiple user stores, you must give a unique domain name to each user store in the `<DomainName>` element. If you configure a user store without specifying a domain name, the server throws an exception at start up.
- If it is the configuration of a super tenant, save the secondary user store definitions in `<PRODUCT_HOME>/repository/deployment/server/userstores` directory.
- If it is a general tenant, save the configuration in `<PRODUCT_HOME>/repository/tenants/<tenantid>/userstores` directory.
- The secondary user store configuration file must have the same name as the domain with an underscore (_) in place of the period. For example, if the domain is `wso2.com`, name the file as `wso2_com.xml`.
- One file only contains the definition for one user store domain.

**Related topics**

**Configuring Primary User Stores**

Every WSO2 product comes with an embedded, internal user store, which is configured in `<PRODUCT_HOME>/repository/conf/user-mgt.xml`. In WSO2 Identity Server, the embedded user store is LDAP, and in other products it is JDBC. Because the domain name (unique identifier) of this default user store is set to PRIMARY by default, it is called the primary user store.

Instead of using the embedded user store, you can set your own user store as the primary user store. Since the user store you want to connect to might have different schemas from the ones available in the embedded user store, it needs to go through an adaptation process. WSO2 products provide the following adapters to enable you to authenticate users from different types of user stores and plug into LDAP, Active Directory, and JDBC to perform authentication:

<table>
<thead>
<tr>
<th>User store manager class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code></td>
<td>Use <code>ReadOnlyLDAPUserStoreManager</code> to do read-only operations for external LDAP user stores.</td>
</tr>
<tr>
<td><code>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</code></td>
<td>Use <code>ReadWriteLDAPUserStoreManager</code> to do both read and write operations. This is the user store configuration which is uncommented in the code in the <code>user-mgt.xml</code> file.</td>
</tr>
</tbody>
</table>

**Note:** The above message does not imply that the user store is added successfully. It simply means that the server is attempting to add the new user store to the end of the available chain of stores.
The `user-mgt.xml` file already has sample configurations for all of the above user stores. To enable these configurations, you must uncomment them in the code and comment out the ones that you do not need.

The following topics provide details on the various primary user stores you can configure.

- Configuring an external LDAP or Active Directory user store
- Configuring an internal/external JDBC user store

If you are using `ldaps` (secured) to connect to the Active Directory as shown below, you need to import the certificate of Active Directory to the `client-truststore.jks` of the WSO2 product. See the topic on configuring keystores for information on how to add certificates to the trust-store.

```xml
<Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
```

**Configuring an external LDAP or Active Directory user store**

All WSO2 products can read and write users and roles from external Active Directory or LDAP user stores. You can configure WSO2 products to access these user stores in one of the following modes:

- Read-only mode
- Read/write mode

**Read-only mode**

**Before you begin**

- If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.
When you configure a product to read users/roles from your company LDAP in read-only mode, it does not write any data into the LDAP.

1. Comment out the following user store which is enabled by default in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml`

```xml
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
  <Property name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
  <Property name="ReadOnly">true</Property>
  <Property name="Disabled">false</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="ConnectionURL">ldap://localhost:10389</Property>
  <Property name="ConnectionName">uid=admin,ou=system</Property>
  <Property name="ConnectionPassword">admin</Property>
  <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
  <Property name="UserSearchBase">ou=system</Property>
  <Property name="UserNameListFilter">(objectClass=person)</Property>
  <Property name="UserNameSearchFilter">(&(objectClass=person)(uid=?))</Property>
  <Property name="UserNameAttribute">uid</Property>
  <Property name="ReadGroups">true</Property>
  <Property name="GroupSearchBase">ou=system</Property>
  <Property name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
  <Property name="GroupNameSearchFilter">(&(objectClass=groupOfNames)(cn=?))</Property>
  <Property name="GroupNameAttribute">cn</Property>
</UserStoreManager>
```

2. Given below is a sample for the LDAP user store. This configuration is found in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file, however, you need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.

```xml
<UserManager>
  <Realm>
    ...
    <UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">
      <Property name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
      <Property name="ReadOnly">true</Property>
      <Property name="Disabled">false</Property>
      <Property name="MaxUserNameListLength">100</Property>
      <Property name="ConnectionURL">ldap://localhost:10389</Property>
      <Property name="ConnectionName">uid=admin,ou=system</Property>
      <Property name="ConnectionPassword">admin</Property>
      <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
      <Property name="UserSearchBase">ou=system</Property>
      <Property name="UserNameListFilter">(objectClass=person)</Property>
      <Property name="UserNameSearchFilter">(&(objectClass=person)(uid=?))</Property>
      <Property name="UserNameAttribute">uid</Property>
      <Property name="ReadGroups">true</Property>
      <Property name="GroupSearchBase">ou=system</Property>
      <Property name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
      <Property name="GroupNameSearchFilter">(&(objectClass=groupOfNames)(cn=?))</Property>
      <Property name="GroupNameAttribute">cn</Property>
    </UserStoreManager>
  </Realm>
</UserManager>
```
<Property name="SharedGroupNameListFilter">(objectClass=groupOfNames)</Property>

<Property name="SharedTenantNameListFilter">(objectClass=organizationalUnit)</Property>

<Property name="SharedTenantNameAttribute">ou</Property>

<Property name="SharedTenantObjectClass">organizationalUnit</Property>

<Property name="MembershipAttribute">member</Property>

<Property name="UserRolesCacheEnabled">true</Property>

<Property name="ReplaceEscapeCharactersAtUserLogin">true</Property>

<Property name="MaxRoleNameListLength">100</Property>

<Property name="MaxUserNameListLength">100</Property>

<Property name="SCIMEnabled">false</Property>
1. Update the connection details to match your user store. For example:

```xml
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

2. Obtain a user who has permission to read all users/attributes and perform searches on the user store from your LDAP/Active Directory administrator. For example, if the privileged user is "AdminLDAP" and the password is "2010#Avrudu", update the following sections of the realm configuration as follows:

```xml
<Property name="ConnectionName">uid=AdminLDAP,ou=system</Property>
<Property name="ConnectionPassword">2010#Avrudu</Property>
```

3. Update `<Property name="UserSearchBase">` with the directory name where the users are stored. When LDAP searches for users, it will start from this location of the directory.

```xml
<Property name="UserSearchBase">ou=system</Property>
```

4. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

```xml
<Property name="UserNameAttribute">uid</Property>
```

5. Set the `ReadGroups` property to 'true', if it should be allowed to read roles from this user store. When this property is 'true', you must also specify values for the `GroupSearchBase`, `GroupSearchFilter` and `GroupNameAttribute` properties as shown in the following example:

```xml
<Property name="ReadGroups">true</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
```

If the `ReadGroups` property is set to ‘false’, only Users can be read from the user store.

6. Optionally, configure the realm to read roles from the user store by reading the user/role mapping
based on a membership (user list) or backlink attribute. The following code snippet represents reading roles based on a membership attribute. This is used by the ApacheDirectory server and OpenLDAP.

```xml
<Property name="ReadLDAPGroups">false</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MembershipAttribute">member</Property>
```

7. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

3. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

**Read/write mode**

**Before you begin**

- To read and write to an Active Directory user store, set the `WriteGroups` property to `true` instead of `false`.
- To write user entries to an LDAP user store (roles are not written, just user entries), you follow the steps in the **Read-only mode** section but specify the following class instead:

```xml
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```

- Use the following class for Active Directory.

```xml
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
```

The `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file has commented-out configurations for external
LDAP/AD user stores.

1. Enable the `<ReadWriteLDAPUserStoreManager>` or the `<ActiveDirectoryUserStoreManager>` in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file by uncommenting the code. When it is enabled, the user manager reads/writes into the LDAP/AD user store. Note that these configurations already exist in the `user-mgt.xml` file so you only need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.

2. The default configuration for the external read/write user store in the `user-mgt.xml` file is as follows. Change the values according to your requirements.

- **LDAP User Store**
- **Active Directory User Store**

**LDAP user store sample:**

```xml
<UserStoreManager
    class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
    <Property
        name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPPTenantManager</Property>
    <Property
        name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
    <Property
        name="ConnectionName">uid=admin,ou=system</Property>
    <Property
        name="PasswordHashMethod">SHA</Property>
    <Property
        name="UserNameListFilter">(objectClass=person)</Property>
    <Property
        name="UserEntryObjectClass">wso2Person</Property>
    <Property
        name="UserNameSearchBase">ou=Users,dc=wso2,dc=org</Property>
    <Property
        name="UserNameSearchFilter">(&(objectClass=person)(uid=?))</Property>
    <Property
        name="UserNameAttribute">uid</Property>
    <Property
        name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property
        name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
        name="UsernameJavaRegEx">^[^~!@#$;%^*+={}\|\\<>\",\'\"]+</Property>
    <Property
        name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
        name="RolenameJavaRegEx">^[^~!@#$;%^*+={}\|\\<>\",\'\"]+</Property>
    <Property
        name="ReadLDAPGroups">true</Property>
    <Property
        name="WriteLDAPGroups">true</Property>
    <Property
        name="EmptyRolesAllowed">true</Property>
    <Property
        name="GroupSearchBase">ou=Groups,dc=wso2,dc=org</Property>
    <Property
        name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
    <Property
        name="GroupNameObjectClass">groupOfNames</Property>
    <Property
        name="GroupNameSearchFilter">(&(objectClass=groupOfNames)(cn=?))</Property>
    <Property
        name="GroupNameAttribute">cn</Property>
```
<Property name="SharedGroupNameAttribute">cn</Property>
<Property name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
<Property name="SharedGroupEntryObjectClass">groups</Property>
<Property name="SharedTenantNameListFilter">(object=organizationalUnit)</Property>
<Property name="SharedTenantNameAttribute">ou</Property>
<Property name="SharedTenantObjectClass">organizationalUnit</Property>
<Property name="MembershipAttribute">member</Property>
<Property name="UserRolesCacheEnabled">true</Property>
Active directory user store sample:

```xml
<UserStoreManager
    class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
    <Property
        name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDA
PTenantManager</Property>
    <Property name="defaultRealmName">WSO2.ORG</Property>
    <Property name="Disabled">false</Property>
    <Property name="kdcEnabled">false</Property>
    <Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
    <Property name="ConnectionName">CN=admin,CN=Users,DC=WSO2,DC=Com</Property>
    <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
    <Property name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
    <Property name="UserEntryObjectClass">user</Property>
    <Property name="UserNameAttribute">cn</Property>
    <Property name="isADLDSRole">false</Property>
    <Property name="userAccountControl">512</Property>
    <Property name="UserNameListFilter">(objectClass=user)</Property>
    <Property name="UserNameSearchFilter">(&(objectClass=user)(cn=?))</Property>
    <Property name="UsernameJavaRegEx">^[a-zA-Z0-9._-\/]\{3,30}$</Property>
    <Property name="UsernameJavaScriptRegEx">^[a-zA-Z0-9._-\/]\{3,30}$</Property>
    <Property name="PasswordJavaScriptRegEx">^[a-zA-Z0-9._-\/]\{5,30}$</Property>
    <Property name="RolenameJavaRegEx">^[a-zA-Z0-9._-\/]\{3,30}$</Property>
    <Property name="ReadGroups">true</Property>
    <Property name="WriteGroups">true</Property>
    <Property name="EmptyRolesAllowed">true</Property>
</UserStoreManager>
```

**Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up: `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`
<Property name="GroupSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
  <Property name="GroupEntryObjectClass">group</Property>
  <Property name="GroupNameAttribute">cn</Property>
  <Property name="SharedGroupNameAttribute">cn</Property>

<Property name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
  <Property name="SharedGroupEntryObjectClass">groups</Property>
  <Property name="SharedTenantNameListFilter">(object=organizationalUnit)</Property>
  <Property name="SharedTenantNameAttribute">ou</Property>
  <Property name="SharedTenantObjectClass">organizationalUnit</Property>
  <Property name="MembershipAttribute">member</Property>

<Property name="GroupNameListFilter">(objectcategory=group)</Property>
  <Property name="GroupNameSearchFilter">(&amp;(objectClass=group)(cn=?))</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="Referral">follow</Property>
  <Property name="BackLinksEnabled">true</Property>
  <Property name="MaxRoleNameListLength">100</Property>
<Property name="MaxUserNameListLength">100</Property>
(Property name="SCIMEnabled">false</Property>
</UserStoreManager>

**Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up- `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

When working with Active Directory it is best to enable the `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows.

```
<AuthorizationManager
   class="org.wso2.carbon.user.core.authorization.JDBCAuthorizatio
nManager">
   <Property
      name="AdminRoleManagementPermissions"/>permission</Property>
   <Property name="AuthorizationCacheEnabled">true</Property>
   <Property name="GetAllRolesOfUserEnabled">true</Property>
</AuthorizationManager>
```

While using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permissions stats.

If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.

The `class` attribute of the `UserStoreManager` element indicates whether it is an Active Directory or LDAP user store:

- **Active Directory:** `<UserStoreManager
   class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager ">
- **Read-only LDAP:** `<UserStoreManager
   class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">

3. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

- **LDAP**
- **Active Directory**
4. The following code snippet represents reading roles based on a backlink attribute. This is used by the Active Directory.

```xml
<Property name="UserNameAttribute">uid</Property>
<Property name="UserNameAttribute">sAMAccountName</Property>
<Property name="ReadLDAPGroups">true</Property>
<Property name="GroupSearchBase">cn=users,dc=wso2,dc=lk</Property>
<Property name="GroupSearchFilter">(objectcategory=group)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MemberOfAttribute">memberOf</Property>
```

5. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

6. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

When configuring an external LDAP for Governance Registry or API Manager, the user name and password for the default admin will change to the LDAP admin. As a result, the `<PRODUCT_HOME>/repository/conf/api-manager.xml` file must be updated with the new LDAP admin credentials.

### Configuring an internal/external JDBC user store

The default internal JDBC user store reads/writes into the internal database of the Carbon server. JDBC user stores can be configured using the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file's `JDBCUserStoreManager` configuration section. Additionally, all Carbon-based products can work with an external RDBMS. You can configure Carbon to read users/roles from your company RDBMS and even write to it. Therefore, in this scenario, the user core connects to two databases:

- The Carbon database where authorization information is stored internally.
- Your company database where users/roles reside.

Therefore, the `user-mgt.xml` file must contain details for two database connections. The connection details mentioned earlier are used by the authorization manager. If we specify another set of database connection details inside the UserStoreManager, it reads/writes users to that database. The following are step-by-step guidelines for connecting to an internal and external JDBC user store in read-only mode:

1. Uncomment the following section in `<PRODUCT_HOME>/repository/conf/user-mgmt.xml`:
The following are samples for the internal and external JDBC user store configuration:

- **Internal JDBC User Store**
- **External JDBC User Store**

**Internal JDBC user store configuration sample:**

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="ReadOnly">false</Property>
    <Property name="WriteGroups">true</Property>
    <Property name="UsernameJavaRegEx">^[\S]{3,30}$</Property>
    <Property name="UsernameJavaScriptRegEx">^[\S]{3,30}$</Property>
    <Property name="UsernameJavaRegExViolationErrorMsg">Username pattern policy violated</Property>
    <Property name="PasswordJavaRegEx">^[\S]{5,30}$</Property>
    <Property name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>
    <Property name="PasswordJavaRegExViolationErrorMsg">Password length should be within 5 to 30 characters</Property>
    <Property name="RolenameJavaRegEx">^[\S]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">^[\S]{3,30}$</Property>
    <Property name="CaseInsensitiveUsername">true</Property>
    <Property name="SCIMEnabled">false</Property>
    <Property name="IsBulkImportSupported">true</Property>
    <Property name="PasswordDigest">SHA-256</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="MultiAttributeSeparator">,</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
</UserStoreManager>
```

**External JDBC user store configuration sample:**

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="driverName">com.mysql.jdbc.Driver</Property>
    <Property name="url">jdbc:mysql://localhost:3306/tcsdev</Property>
    <Property name="username">shavantha</Property>
    <Property name="password">welcome</Property>
    <Property name="Disabled">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordDigest">SHA-256</Property>
</UserStoreManager>
```
<Property name="ReadGroups">true</Property>
<Property name="ReadOnly">false</Property>
<Property name="IsEmailUserName">false</Property>
<Property name="DomainCalculation">default</Property>
<Property name="StoreSaltedPassword">true</Property>
<Property name="WriteGroups">false</Property>
<Property name="UserNameUniqueAcrossTenants">false</Property>
<Property name="PasswordJavaRegEx">^[\S]{5,30}$</Property>
<Property name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>
<Property name="UsernameJavaRegEx">^[\S]{5,30}$</Property>
<Property name="UsernameJavaScriptRegEx">^[\S]{5,30}$</Property>
<Property name="RolenameJavaRegEx">^[\S]{5,30}$</Property>
<Property name="RolenameJavaScriptRegEx">^[\S]{5,30}$</Property>
<Property name="SCIMEnabled">false</Property>
<Property name="SelectUserSQL">SELECT * FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="GetRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_TENANT_ID=? AND UM_SHARED_ROLE = '0' ORDER BY UM_ROLE_NAME</Property>
<Property name="GetSharedRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_SHARED_ROLE = '1' ORDER BY UM_ROLE_NAME</Property>
<Property name="UserFilterSQL">SELECT UM_USER_NAME FROM UM_USER WHERE UM_USER_NAME LIKE ? AND UM_TENANT_ID=? ORDER BY UM_USER_NAME</Property>
<Property name="UserSharedRoleSQL">SELECT UM_ROLE_NAME, UM_ROLE.UM_TENANT_ID, UM_SHARED_ROLE FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID WHERE UM_ROLE.UM_ROLE_NAME= ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>
<Property name="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID WHERE UM_ROLE.UM_ROLE_NAME= ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_ROLE.UM_TENANT_ID</Property>
<Property name="IsUserExistingSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>


-Property name="GetProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?</Property>

-Property name="GetUserProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>

-Property name="GetUserIDFromUserNameSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>

-Property name="GetUserNameFromTenantIDSQL">SELECT UM_USER_NAME FROM UM_USER WHERE UM_TENANT_ID=?</Property>

-Property name="GetTenantIDFromUserNameSQL">SELECT UM_TENANT_ID FROM UM_USER WHERE UM_USER_NAME=?</Property>

-Property name="AddUserSQL">INSERT INTO UM_USER (UM_USER_NAME, UM_USER_PASSWORD, UM_SALT_VALUE, UM_REQUIRE_CHANGE, UM_CHANGED_TIME, UM_TENANT_ID) VALUES (?, ?, ?, ?, ?, ?)</Property>

-Property name="AddUserRoleToUserSQL">INSERT INTO UM_USER_ROLE (UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?),(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), ?)</Property>

-Property name="AddRoleSQL">INSERT INTO UM_ROLE (UM_ROLE_NAME, UM_TENANT_ID) VALUES (?, ?)</Property>

-Property name="AddSharedRoleSQL">UPDATE UM_ROLE SET UM_SHARED_ROLE = ? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID = ?</Property>

-Property name="AddRoleToUserSQL">INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?),(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?)</Property>

-Property name="AddSharedRoleToUserSQL">INSERT INTO UM_SHARED_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_USER_TENANT_ID,

<Property name="RemoveUserFromRoleSQL">DELETE FROM "UM_USER_ROLE" WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>

<Property name="RemoveRoleFromUserSQL">DELETE FROM "UM_USER_ROLE" WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>

<Property name="DeleteRoleSQL">DELETE FROM "UM_ROLE" WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>

<Property name="OnDeleteRoleRemoveUserRoleMappingSQL">DELETE FROM "UM_USER_ROLE" WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>

<Property name="DeleteUserSQL">DELETE FROM "UM_USER" WHERE UM_USER_NAME = ? AND UM_TENANT_ID=?</Property>


<Property name="UpdateRoleNameSQL">UPDATE "UM_ROLE" set UM_ROLE_NAME=? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>

<Property name="AddUserPropertySQL">INSERT INTO "UM_USER_ATTRIBUTE" (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?, ?, ?)</Property>


<Property name="UserNameUniqueAcrossTenantsSQL">SELECT UM_ID FROM "UM_USER" WHERE UM_USER_NAME=?</Property>

<Property name="IsDomainExistingSQL">SELECT UM_DOMAIN_ID FROM "UM_DOMAIN" WHERE UM_DOMAIN_NAME=? AND UM_TENANT_ID=?</Property>

<Property name="AddDomainSQL">INSERT INTO "UM_DOMAIN" (UM_DOMAIN_NAME, UM_TENANT_ID) VALUES (?, ?)</Property>

<Property name="AddUserRoleToRole-mssql">INSERT INTO "UM_USER_ROLE" (UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?)</Property>
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (?)</Property>

<Property name="AddRoleToUserSQL-mssql">INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (?)</Property>

<Property name="AddUserPropertySQL-mssql">INSERT INTO UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (?), (?), (?), (?)</Property>


The sample for the external JDBC user store consists of properties pertaining to various SQL statements. This is because the schema may be different for an external user store, and these adjustments need to be made in order to streamline the configurations with WSO2 products.

You can define a data source in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` and refer to it from the `user-mgt.xml` file. This takes the properties defined in the `master-datasources.xml` file and reuses them in the `user-mgt.xml` file. To do this, you need to define the following property:

```
<Property name = "dataSource">jdbc/WSO2CarbonDB</Property>
```

2. Find a valid user that resides in the RDBMS. For example, say a valid username is AdminSOA. Update the Admin user section of your configuration as follows. You do not have to update the password element; leave it as is.

```
<AdminUser>
    <UserName>AdminSOA</UserName>
    <Password>XXXXXX</Password>
</AdminUser>
```

3. Add the `PasswordHashMethod` property to the `UserStoreManager` configuration for `JDBCUserStoreManager`. For example:

```
<UserStoreManager
    class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name = "PasswordHashMethod">SHA</Property>
    ...
</UserStoreManager>
```

The `PasswordHashMethod` property specifies how the password should be stored. It usually has the following values:

- **SHA** - Uses SHA digest method.
- **MD5** - Uses MD 5 digest method.
- **PLAIN_TEXT** - Plain text passwords.

In addition, it also supports all digest methods in [http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html](http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html).
4. Update the connection details found within the `<UserStoreManager>` class based on your preferences.
5. In the realm configuration section, set the value of the `MultiTenantRealmConfigBuilder` property to `org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder`. For example:

   ```xml
   <Property
       name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
   ```

6. Add the JDBC driver to the classpath by copying its JAR file into the `<PRODUCT_HOME>/repository/components/lib` directory.
7. Edit the SQLs in the `user-mgt.xml` file according to your requirements, and then start the server.

**Working with Properties of Primary User Stores**

The following table provides descriptions of the key properties you use to configure primary user stores.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxUserNameListLength</td>
<td>Controls the number of users listed in the user store of a WSO2 product. Setting this property to 0 displays all users.</td>
</tr>
<tr>
<td>ConnectionURL</td>
<td>Connection URL to the user store server. In the case of default LDAP in Carbon, the port is specified in the <code>carbon.xml</code> file, and a reference to that port is included in this configuration.</td>
</tr>
<tr>
<td>ConnectionName</td>
<td>The username used to connect to the database and perform various operations. This user does not have to be an administrator in the user store or have an administrator role in the WSO2 product that you are using, but the user must have permissions to read the user list and users' attributes and to perform search operations on the user store. The value you specify is used as the DN attribute of the user. This property is mandatory.</td>
</tr>
<tr>
<td>ConnectionPassword</td>
<td>Password for the <code>ConnectionName</code> user.</td>
</tr>
<tr>
<td>PasswordHashMethod</td>
<td>Password hash method to use when storing user entries in the user store.</td>
</tr>
<tr>
<td>UserNameListFilter</td>
<td>Filtering criteria for listing all the user entries in the user store. This query or filter is used when doing search operations on users. In this case, the search operation only provides the objects created from the specified class.</td>
</tr>
<tr>
<td>UserEntryObjectClass</td>
<td>Object class used to construct user entries. By default, it is a custom object class defined with the name <code>wso2Person</code>.</td>
</tr>
<tr>
<td>UserSearchBase</td>
<td>DN of the context or object under which the user entries are stored in the user store. When the user store searches for users, it will start from this location of the directory. Different databases have different search bases.</td>
</tr>
<tr>
<td>UserNameSearchFilter</td>
<td>Filtering criteria used to search for a particular user entry.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UserNameAttribute</td>
<td>The attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, UID, etc.</td>
</tr>
<tr>
<td>UsernameWithEmailJavaScriptRegEx</td>
<td>This property defines the JavaScript regular expression pattern when you need to support both email as a user name and normal user names.</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>Policy that defines the password format.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for username validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings have a length of 5 to 30. Only non-empty characters are allowed.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for role name validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression used to validate role names. By default, strings have a length of 5 to 30. Only non-empty characters are allowed.</td>
</tr>
<tr>
<td>ReadGroups</td>
<td>Specifies whether groups should be read from the user store. If this is disabled by setting it to false, none of the groups in the user store can be read.</td>
</tr>
<tr>
<td>WriteGroups</td>
<td>Specifies whether groups should be written to the user store.</td>
</tr>
<tr>
<td>EmptyRolesAllowed</td>
<td>Specifies whether the underlying user store allows empty groups to be created. Usually LDAP servers do not allow you to create empty groups.</td>
</tr>
<tr>
<td>GroupSearchBase</td>
<td>DN of the context under which user entries are stored in the user store.</td>
</tr>
<tr>
<td>GroupSearchFilter</td>
<td>The query used to search for groups.</td>
</tr>
<tr>
<td>GroupNameListFilter</td>
<td>Filtering criteria for listing all the group entries in the user store. Grc returns objects created from this class.</td>
</tr>
<tr>
<td>GroupEntryObjectClass</td>
<td>Object class used to construct group entries.</td>
</tr>
<tr>
<td>GroupNameSearchFilter</td>
<td>Filtering criteria used to search for a particular group entry.</td>
</tr>
<tr>
<td>GroupNameAttribute</td>
<td>Attribute used for uniquely identifying a user entry. This attribute is considered as the username.</td>
</tr>
<tr>
<td>MembershipAttribute</td>
<td>Attribute used to define members of groups.</td>
</tr>
<tr>
<td>UserRolesCacheEnabled</td>
<td>This is to indicate whether to cache the role list of a user. By default, those changes should be instantly reflected in the Carbon instance.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UserDNPattern</td>
<td>(LDAP) The pattern for the user's DN, which can be defined to improve search performance and impact on performance as the LDAP does not have to travel through the entire tree to find users.</td>
</tr>
<tr>
<td>ReplaceEscapeCharactersAtUserLogin</td>
<td>(LDAP) If the user name has special characters it replaces it to validate user login. Only \ and \ are identified as escape characters.</td>
</tr>
<tr>
<td>TenantManager</td>
<td>Includes the location of the tenant manager.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>(LDAP and JDBC) Indicates whether the user store of this realm operates in read-only mode or not.</td>
</tr>
<tr>
<td>IsEmailUserName</td>
<td>(JDBC) Indicates whether the user's email is used as their username.</td>
</tr>
<tr>
<td>DomainCalculation</td>
<td>(JDBC) Can be either default or custom (this applies when the realm operates in read-only mode).</td>
</tr>
<tr>
<td>PasswordDigest</td>
<td>(JDBC) Digesting algorithm of the password. Has values such as, F, H2, SHA.</td>
</tr>
<tr>
<td>StoreSaltedPassword</td>
<td>(JDBC) Indicates whether to salt the password.</td>
</tr>
<tr>
<td>UserNameUniqueAcrossTenants</td>
<td>(JDBC) An attribute used for multi-tenancy.</td>
</tr>
<tr>
<td>PasswordJavaRegEx</td>
<td>(LDAP and JDBC) A regular expression to validate passwords. By default, strings having a length between 5 to 30 with non-empty characters are allowed.</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for password validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings have a length between 5 to 30 with non-empty characters.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for username validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression to validate role names. By default, strings have a length between 5 to 30 with non-empty characters.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for role name validation.</td>
</tr>
<tr>
<td>MultiTenantRealmConfigBuilder</td>
<td>Tenant Manager specific realm config parameter. Can be used to build different types of realms for the tenant.</td>
</tr>
<tr>
<td>SharedGroupEnabled</td>
<td>This property is used to enable/disable the shared role functionality.</td>
</tr>
<tr>
<td>SharedGroupSearchBase</td>
<td>Shared roles are created for other tenants to access under the mentioned DN.</td>
</tr>
<tr>
<td>SharedTenantObjectClass</td>
<td>Object class for the shared groups created.</td>
</tr>
<tr>
<td>SharedTenantNameAttribute</td>
<td>Name attribute for the shared group.</td>
</tr>
<tr>
<td>SharedTenantNameListFilter</td>
<td>This is currently not used.</td>
</tr>
</tbody>
</table>

**Working with Databases**

The default databases that WSO2 products uses to store registry, user manager and product-specific data are the H2 databases in `<PRODUCT_Home>/repository/database` as follows:
• **WSO2CARBON_DB.h2.db**: used to store registry and user manager data

These embedded H2 databases are suitable for development, testing, and some production environments. For most production environments, however, we recommend you to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc.

In addition to the above databases, if the product uses features of WSO2 Identity Server or WSO2 Enterprise Store (ES), data which are specific to these will by default reside in the embedded Carbon database. However, for production environments, we recommend you to use separate RDBMS databases to store identity-related and storage-related data.

You can use the scripts provided with WSO2 products to install and configure several other types of relational databases, including MySQL, IBM DB2, Oracle, and more.

The following sections explain how to change the default databases:

- Setting up the Physical Database
- Managing Datasources

### Setting up the Physical Database

The topics in this section describe how to use scripts in `<PRODUCT_HOME>/dbscripts/` folder to set up each type of physical database.

- Setting up IBM DB2
- Setting up Derby
- Setting up H2
- Setting up IBM Informix
- Setting up Microsoft SQL
- Setting up MySQL
- Setting up MySQL Cluster
- Setting up OpenEdge
- Setting up Oracle
- Setting up Oracle RAC
- Setting up PostgreSQL
- Setting up MariaDB

### Setting up IBM DB2

The following sections describe how to replace the default H2 databases with IBM DB2:

- Prerequisites
- Setting up the database and users
- Setting up DB2 JDBC drivers
- Setting up datasource configurations
- Creating database tables

#### Prerequisites

Download the latest version of **DB2 Express-C** and install it on your computer.

For instructions on installing DB2 Express-C, see this [ebook](#).
Setting up the database and users

Create the database using either DB2 command processor or DB2 control center as described below.

**Using the DB2 command processor**

1. Run DB2 console and execute the `db2start` command in CLI to open DB2.
2. Create the database using the following command:
   
   ```
   create database <DB_NAME>
   ```

3. Before issuing a SQL statement, establish the connection to the database using the following command:
   
   ```
   connect to <DB_NAME> user <USER_ID> using <PASSWORD>
   ```

4. Grant required permissions for users as follows:

   ```
   connect to DB_NAME
   grant <AUTHORITY> on database to user <USER_ID>
   ```

   For example:

   ![DB2 Command Process Example]

   For more information on DB2 commands, see the DB2 Express-C Guide.

**Using the DB2 control center**

1. Open the DB2 control center using the `db2cc` command as follows:

   ```
   greg@so2:~/sql/lib/bin$ ./db2cc
   ```

2. Right-click All Databases in the control center tree (inside the object browser), click Create Database, and then click Standard and follow the steps in the Create New Database wizard.
3. Click **User and Group Objects** in the control center tree to create users for the newly created database.

4. Give the required permissions to the newly created users.
4. Setting up DB2 JDBC drivers

Copy the DB2 JDBC drivers (db2jcc.jar and db2jcc_license_c0u.jar) from <DB2_HOME>/SQLLIB/java/ directory to the <PRODUCT_HOME>/repository/components/lib/ directory.

```bash
user@uso2:~/sqllib/java$ cp db2jcc.jar db2jcc_license_c0u.jar /home/user/uso2/greg/repository/components/lib/
user@uso2:~/sqllib/java$
```

<DB2_HOME> refers to the installation directory of DB2 Express-C, and <PRODUCT_HOME> refers to the directory where you run the WSO2 product instance.

5. Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM DB2 database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**
Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:db2://SERVER_NAME:PORT/DB_NAME</url>
         <username>regadmin</username>
         <password>regadmin</password>
         <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
         <maxActive>80</maxActive>
         <maxWait>360000</maxWait>
         <minIdle>5</minIdle>
         <testOnBorrow>true</testOnBorrow>
         <validationQuery>SELECT 1</validationQuery>
         <validationInterval>30000</validationInterval>
         <defaultAutoCommit>false</defaultAutoCommit>
       </configuration>
     </definition>
   </datasource>
   
   The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
</tbody>
</table>
### testOnBorrow
The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

### validationQuery
The SQL query that will be used to validate connections from this pool before returning them to the caller.

### validationInterval
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

### defaultAutoCommit
This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

---

**For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file, see Tomcat JDBC Connection Pool.**

---

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

---

**Creating database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts in the
DB2 Express-C command editor.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

   ```
   <PRODUCT_HOME>/dbscripts/db2.sql
   ```

2. Restart the server.

   You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

   - For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
   - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

---

**Setting up Derby**

You can set up either an embedded Derby database or a remote database as described in the following topics:

- Setting up Embedded Derby
- Setting up Remote Derby

**Setting up Embedded Derby**

The following sections describe how to replace the default H2 databases with embedded Derby:

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

**Setting up the database**

Follow the steps below to set up an embedded Derby database:

1. Download Apache Derby.
2. Install Apache Derby on your computer.

   For instructions on installing Apache Derby, see the Apache Derby documentation.

**Setting up the drivers**

Copy `derby.jar`, `derbyclient.jar`, and `derbynet.jar` from the `<DERBY_HOME>/lib/` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the WSO2 Carbon web application).

**Setting up datasource configurations**
A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Embedded Derby database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.apache.derby.jdbc.EmbeddedDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections</td>
</tr>
</tbody>
</table>
### Table: Parameters and Their Descriptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>maxWait</strong></td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td><strong>minIdle</strong></td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td><strong>testOnBorrow</strong></td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td><strong>validationQuery</strong></td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td><strong>validationInterval</strong></td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td><strong>defaultAutoCommit</strong></td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).

   ```xml
   <dbConfig name="wso2registry">
   <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
Creating database tables

You can create database tables by executing the database scripts as follows:

1. Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory as illustrated below:

   ```bash
   ij version 10.8
   ```

2. Create the database and connect to it using the following command inside the `ij` prompt:

   ```sql
   connect 'jdbc:derby:repository/database/WSO2CARBON_DB;create=true';
   ```

   Replace the database file path in the above command with the full path to your database.

3. Exit from the `ij` tool by typing the `exit` command.

   ```bash
   exit;
   ```

4. Log in to the `ij` tool with the username and password that you set in `registry.xml` and `user-mgt.xml`:

   ```sql
   connect 'jdbc:derby:repository/database/WSO2CARBON_DB' user 'regadmin' password 'regadmin';
   ```

5. Use the scripts given in the following locations to create the database tables:

   - To create tables for the **registry and user manager database (WSO2CARBON_DB)**, run the below command:

     ```bash
     run '<PRODUCT_HOME>/dbscripts/derby.sql';
     ```

     Now the product is running using the embedded Apache Derby database.

6. Restart the server.

   You can create database tables automatically **when starting the product for the first time** by using the `-D setup` parameter as follows:

   - For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
   - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

   The product is configured to run using an embedded Apache Derby database.
Setting up Remote Derby

The following sections describe how to replace the default H2 databases with a remote Derby database:

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

**Setting up the database**

Follow the steps below to set up a remote Derby database.

1. Download Apache Derby.
2. Install Apache Derby on your computer.

   For instructions on installing Apache Derby, see the Apache Derby documentation.

3. Go to the `<DERBY_HOME>/bin/` directory and run the Derby network server start script. Usually it is named `startNetworkServer`.

**Setting up the drivers**

Copy `derby.jar`, `derbyclient.jar`, and `derbynet.jar` from the `<DERBY_HOME>/lib/` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the Carbon web application).

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Remote Derby database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.apache.derby.jdbc.ClientDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file, see Tomcat JDBC Connection Pool.

In contrast to setting up with embedded Derby, in the remote registry you set the database driver name (the `driverName` element) to the value `org.apache.derby.jdbc.ClientDriver` and the database URL (the `url` element) to the database remote location.

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

**Creating database tables**

You can create database tables by executing the following script(s):

1. Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory.
2. Create the database and connect to it using the following command inside the ij prompt:

```sql
connect 'jdbc:derby://localhost:1527/db;user=regadmin;password=regadmin;create=true';
```

Replace the database file path, user name, and password in the above command to suit your requirements.

3. Exit from the ij tool by typing the exit command as follows:

```sql
exit;
```

4. Log in to the ij tool with the username and password you just used to create the database.

```sql
connect 'jdbc:derby://localhost:1527/db' user 'regadmin' password 'regadmin';
```

5. You can create database tables manually by executing the following scripts.

   - To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

```sql
run '<PRODUCT_HOME>/dbscripts/derby.sql';
```

6. Restart the server.

You can create database tables automatically **when starting the product for the first time** by using the -Dsetup parameter as follows:

   - For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
   - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

The product is now configured to run using a remote Apache Derby database.

### Setting up H2

You can set up either an embedded H2 database or a remote H2 database using the instructions in the following topics:

- Setting up Embedded H2
- Setting up Remote H2

### Setting up Embedded H2

The following sections describe how to replace the default H2 databases with Embedded H2:
Setting up the database

Download and install the H2 database engine in your computer.

Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:
   `<PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar`
2. Find the JAR file of the new H2 database driver (`<H2_HOME>/bin/h2-*.jar`, where `<H2_HOME>` is the H2 installation directory) and copy it to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Embedded H2 database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

For instructions on installing DB2 Express-C, see H2 installation guide.
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>
            <driverClassName>org.h2.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and pass word</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
</tbody>
</table>
### validationQuery
The SQL query that will be used to validate connections from this pool before returning them to the caller.

### validationInterval
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

### defaultAutoCommit
This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

---

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
   <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

   ```xml
   <Configuration>
   <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in the H2 shell or web console:

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:
Follow the steps below to run the script in Web console:

1. Run the `./h2.sh` command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click Run.
5. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Changing the identity/storage databases**

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-dataSources.xml` file.
2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/h2.sql</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>For the storage database</td>
<td>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/storage/h2.sql</code></td>
</tr>
</tbody>
</table>

**Setting up Remote H2**

The following sections describe how to replace the default H2 databases with Remote H2:

- Setting up the remote H2 database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the identity/storage databases

**Setting up the remote H2 database**

Follow the steps below to set up a Remote H2 database.

1. Download and install the H2 database engine on your computer as follows.

   For instructions on installing, see the H2 installation guide.
1. Go to the `<H2_HOME>/bin/` directory and run the H2 network server starting script as follows, where `<H2_HOME>` is the H2 installation directory:

   ```bash
   ./.h2.sh
   ```

2. Run the H2 database server with the following commands:
   - For Linux:
     ```bash
     ./h2.sh
     ```
   - For Windows:
     ```bash
     h2.bat
     ```

   The script starts the database engine and opens a pop-up window.

3. Click **Start Browser** to open a web browser containing a client application, which you use to connect to a database. If a database does not already exist by the name you provided in the **JDBC URL** text box, H2 will automatically create a database.

### Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:
   ```
   <PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar
   ```

2. Find the JAR file of the new H2 database driver (`<H2_HOME>/bin/h2-*.*.jar`, where `<H2_HOME>` is the H2 installation directory) and copy it to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory.

### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Remote H2 database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default `WSO2_CARBON_DB` datasource

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:h2:tcp://localhost/~/registryDB;create=true</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>org.h2.Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</ datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
validationInterval  | The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

defaultAutoCommit  | This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file, see Tomcat JDBC Connection Pool.

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ registry.xml` file.

   ```
   <dbConfig name="wso2registry">
   <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

   ```
   <Configuration>
   <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in H2 shell or web console:

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

   ```
   <PRODUCT_HOME>/dbscripts/h2.sql
   ```
Follow the steps below to run the script in Web console:

1. Run the "./h2.sh" command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click Run.

5. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Changing the identity/storage databases**

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you set up a separate database for identity/storage related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.
2. Create the database tables using the following scripts:

   | For the identity database | Use `<PRODUCT_HOME>/dbscripts/identity/h2.sql` |
Setting up IBM Informix

The following sections describe how to replace the default H2 databases with IBM Informix:

- Prerequisites
- Creating the database
- Setting up Informix JDBC drivers
- Setting up datasource configurations
- Creating database tables
- Changing the identity database

Prerequisites

Download the latest version of IBM Informix and install it on your computer.

Creating the database

Create the database and users in Informix. For instructions on creating the database and users, see Informix product documentation.

Do the following changes to the default database when creating the Informix database.

- Use page size as 4K or higher when creating the dbspace as shown in the following command (i.e. denoted by -k 4):
  ```
  onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000
  ```
- Add the following system environment variables.
  ```
  export DB_LOCALE=en_US.UTF-8
  export CLIENT_LOCALE=en_US.UTF-8
  ```
- Create a sbspace other than the dbspace by executing the following command:
  ```
  onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000
  ```
- Add the following entry to the example sbspace name (i.e. testspace4) with your sbspace name: SBSPACENAME testspace4

Setting up Informix JDBC drivers

Download the Informix JDBC drivers and copy them to your WSO2 product's <PRODUCT_HOME>/repository/components/lib directory.

Use Informix JDBC driver version 3.70.JC8, 4.10.JC2 or higher.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM
Informix database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the \PRODUCT_HOME\repository\conf\datasources\master-datasources.xml file as shown below.

```
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <!-- IP ADDRESS AND PORT OF DB SERVER -->
      <url>jdbc:informix-sqli://localhost:1533/AM_DB;CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</url>
      <username>wso2carbon</username>
      <password>wso2carbon</password>
      <driverClassName>com.informix.jdbc.IfxDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000. You need to add the following configuration when specifying the connection URL as shown above: Add the following configuration to the connection URL when specifying it as shown above: CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</td>
</tr>
</tbody>
</table>
### Database Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from the pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS driver's default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file, see *Tomcat JDBC Connection Pool*.

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ registry.xml` file.

```xml
<dbConfig name="wso2registry">
    <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```
3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

**Creating database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

```
<PRODUCT_HOME>/dbscripts/informix.sql
```

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Changing the identity database**

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/informix.sql</code></th>
</tr>
</thead>
</table>

**Setting up Microsoft SQL**

The following sections describe how to replace the default H2 database with MS SQL:

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the identity database

**Setting up the database and users**
Follow the steps below to set up the Microsoft SQL database and users.

**Enable TCP/IP**

1. In the start menu, click **Programs** and launch **Microsoft SQL Server 2005**.
2. Click **Configuration Tools**, and then click **SQL Server Configuration Manager**.
3. Enable TCP/IP and disable **Named Pipes** from protocols of your Microsoft SQL server.
4. Double click **TCP/IP** to open the TCP/IP properties window, and set **Listen All** to **Yes** on the **Protocol** tab.
5. On the **IP Address** tab, disable **TCP Dynamic Ports** by leaving it blank and give a valid TCP port, so that Microsoft SQL server will listen on that port.

   The best practice is to use port 1433, because you can use it in order processing services.

6. Similarly, enable TCP/IP from **SQL Native Client Configuration** and disable **Named Pipes**. Also check whether the port is set correctly to 1433.
7. Restart Microsoft SQL Server.

Create the database and user

1. Open Microsoft SQL Management Studio to create a database and user.
2. Click **New Database** from the **Database** menu, and specify all the options to create a new database.
3. Click **New Login** from the **Logins** menu, and specify all the necessary options.

Grant permissions

Assign newly created users the required grants/permissions to log in, create tables, and insert, index, select, update, and delete data in tables in the newly created database, as the minimum set of SQL server permissions.

**Setting up the JDBC driver**

Download and copy the sqljdbc4 Microsoft SQL JDBC driver file to the WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory. Use `com.microsoft.sqlserver.jdbc.SQLServerDriver` as the `<driverClassName>` in your `datasource configuration` in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Microsoft SQL database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:sqlserver://<IP>:1433;databaseName=wso2greg</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. Change the <code>&lt;IP&gt;</code> with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
</tbody>
</table>
validationQuery | The SQL query that will be used to validate connections from this pool before returning them to the caller.

validationInterval | The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

defaultAutoCommit | This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file, see Tomcat JDBC Connection Pool.

---

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the WSO2_CARBON_DB datasource above to the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT_HOME>/repository/conf/registry.xml file.

```xml
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.

```xml
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

---

**Creating the database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:
<PRODUCT_HOME>/dbscripts/mssql.sql

2. Restart the server.

You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Changing the identity database**

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.
2. Create the database tables using the following scripts:

   | For the identity database | Use `<PRODUCT_HOME>/dbscripts/identity/mssql.sql` |

**Setting up MySQL**

The following sections describe how to replace the default H2 databases with MySQL:

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the registry/user management databases
- Changing the identity/storage databases

**Setting up the database and users**

Follow the steps below to set up a MySQL database:

1. Download and install MySQL on your computer using the following command:

   ```bash
   sudo apt-get install mysql-server mysql-client
   ```

   For instructions on installing MySQL on MAC OS, go to Homebrew.

2. Start the MySQL service using the following command:

   ```bash
   sudo /etc/init.d/mysql start
   ```
3. Log in to the MySQL client as the root user (or any other user with database creation privileges).
   
   mysql -u root -p

4. Enter the password when prompted.

   In most systems, there is no default root password. Press the Enter key without typing anything if you have not changed the default root password.

5. In the MySQL command prompt, create the database using the following command:
   
   create database regdb;

   For users of Microsoft Windows, when creating the database in MySQL, it is important to specify the character set as latin1. Failure to do this may result in an error (error code: 1709) when starting your cluster. This error occurs in certain versions of MySQL (5.6.x), and is related to the UTF-8 encoding. MySQL originally used the latin1 character set by default, which stored characters in a 2-byte sequence. However, in recent versions, MySQL defaults to UTF-8 to be friendlier to international users. Hence, you must use latin1 as the character set as indicated below in the database creation commands to avoid this problem. Note that this may result in issues with non-latin characters (like Hebrew, Japanese, etc.). The database creation command should be as follows:

   mysql> create database <DATABASE_NAME> character set latin1;

   For users of other operating systems, the standard database creation commands will suffice. For these operating systems, the database creation command should be as follows:

   mysql> create database <DATABASE_NAME>;

6. Give authorization of the database to the regadmin user as follows:

   GRANT ALL ON regdb.* TO regadmin@localhost IDENTIFIED BY "regadmin";

7. Once you have finalized the permissions, reload all the privileges by executing the following command:

   FLUSH PRIVILEGES;

8. Log out from the MySQL prompt by executing the following command:

   quit;

Setting up the drivers

Download the MySQL Java connector JAR file, and copy it to the $PRODUCT_HOME/repository/components/lib/ directory.

Tip: Be sure to use the connector version that is supported by the MySQL version you use. If you come across any issues due to version incompatibility, follow the steps below:

1. Shut down the server and remove all existing connectors from $PRODUCT_HOME/repository/components/lib and $PRODUCT_HOME/repository/components/dropins.
2. Download the connector JAR that is compatible with your current MySQL version.
3. Copy the JAR file only to $PRODUCT_HOME/repository/components/lib. Files will be copied automatically to the dropins folder at the server startup.
4. Start the server with the -Dsetup parameter as sh wso2server.sh -Dsetup.
Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the MySQL database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/regdb</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MySQL is 3306</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
</tbody>
</table>
maxActive | The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.

maxWait | The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.

minIdle | The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.

testOnBorrow | The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

validationQuery | The SQL query that will be used to validate connections from this pool before returning them to the caller.

validationInterval | The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

defaultAutoCommit | This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/`. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
    <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

```bash
mysql -u regadmin -p -Dregdb < '<PRODUCT_HOME>/dbscripts/mysql.sql';
```

You may have to enter the password for each command when prompted.

If you are using MySQL version 5.7 or later, use the following script instead:

```bash
mysql -u regadmin -p -Dregdb < '<PRODUCT_HOME>/dbscripts/mysql5.7.sql';
```

If you prefer to populate MySQL 5.7 or later with the -Dsetup command, rename mysql5.7.sql to mysql.sql.

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the -Dsetup parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

Changing the registry/user management databases

If you change the database that comes by default or set up a separate database for registry or user management related data, follow the below instructions.

1. Add the datasource to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

Changing the identity/storage databases

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you set up a separate database for identity/storage related data, the instructions are the same. In summary:
1. Add the datasource to the master-datasources.xml file.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use &lt;PRODUCT_HOME&gt;/dbscripts/identity/mysql.sql</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the storage database</td>
<td>Use &lt;PRODUCT_HOME&gt;/dbscripts/storage/mysql.sql</td>
</tr>
</tbody>
</table>

**Setting up MySQL Cluster**

For instructions on setting up any WSO2 product with a MySQL cluster, see this article, which is published in the WSO2 library.

**Setting up OpenEdge**

The following sections describe how to replace the default H2 databases with OpenEdge (OE):

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

**Setting up the database and users**

Follow the steps below to set up an OpenEdge (OE) database.

1. Download and install OpenEdge on you computer.
2. Go to the <OE_HOME>/bin/ directory and use the proenv script to set up the environment variables.
3. Add `<OE_HOME>/java/prosp.jar` to the CLASSPATH environment variable.
4. Create an empty database using the prodb script as follows. This script creates a database by copying an existing database provided with the installation.
   ```
   prodb CARBON_DB <OE-installation-directory>/empty8
   ```
5. Start the database using the proserve script as follows. Provide the database name and a port as arguments to this script using the -db and -S parameters.
   ```
   proserve -db CARBON_DB -S 6767
   ```
6. Use the sqlexp script to start the default SQL explorer that comes with the OpenEdge installation. Connect to the database you just created by using the -db and -S parameters as follows:
   ```
   sqlexp -db CARBON_DB -S 6767
   ```
7. Use the following commands to create a user and grant that user the required permissions to the database:
   ```
   CREATE USER 'wso2carbon','wso2carbon';
   GRANT dba,resource TO 'wso2carbon';
   COMMIT;
   ```
8. Log out from the SQL explorer by typing the following command: exit
Setting up the drivers

Copy the `<OE_HOME>/java/openedge.jar` file to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the OpenEdge database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default `WSO2_CARBON_DB` datasource**

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:datadirect:openedge://localhost:6767;databaseName=CARBON_DB</url>
         <username>regadmin</username>
         <password>regadmin</password>
         <driverClassName>com.ddtek.jdbc.openedge.OpenEdgeDriver</driverClassName>
         <maxActive>80</maxActive>
         <maxWait>60000</maxWait>
         <minIdle>5</minIdle>
         <testOnBorrow>true</testOnBorrow>
         <validationQuery>SELECT 1</validationQuery>
         <validationInterval>30000</validationInterval>
         <defaultAutoCommit>false</defaultAutoCommit>
       </configuration>
     </definition>
   </datasource>
   ```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the datasource</td>
</tr>
<tr>
<td>description</td>
<td>Description of the datasource</td>
</tr>
<tr>
<td>jndiConfig</td>
<td>JNDI configuration for the datasource</td>
</tr>
<tr>
<td>definition</td>
<td>Definition of the datasource type and configuration details</td>
</tr>
<tr>
<td>configuration</td>
<td>Configuration details for the datasource</td>
</tr>
</tbody>
</table>

The configuration parameters can be modified according to the requirements and the database being used.
| **url** | The URL of the database. The default port for a DB2 instance is 50000. |
| **username** and **password** | The name and password of the database user |
| **driverClassName** | The class name of the database driver |
| **maxActive** | The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections. |
| **maxWait** | The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely. |
| **minIdle** | The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none. |
| **testOnBorrow** | The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another. |
| **validationQuery** | The SQL query that will be used to validate connections from this pool before returning them to the caller. |
| **validationInterval** | The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. |
| **defaultAutoCommit** | This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction. |

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/make-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_Carbon_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/make-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.
3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

```xml
<Configuration>
    <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts:

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```sql
<PRODUCT_HOME>/dbscripts/openedge.sql
```

Follow the steps below to create the database tables by executing the scripts:

1. Modify the OpenEdge script provided with the product to create the tables manually. Make a backup of the `<PRODUCT_HOME>/dbscripts/openedge.sql` script under the name `openedge_manual.sql`.
2. Replace all the `"/"` symbols in the `openedge_manual.sql` script with the `"."` symbol.
3. At the end of the `openedge_manual.sql` script, add the following line and save the script:

   ```sql
   COMMIT;
   ```

4. Run the modified script using the SQL explorer as follows:

   ```bash
   sqlexp -db CARBON_DB -S 6767 -user wso2carbon -password wso2carbon <PRODUCT_HOME>/dbscripts/openedge_manual.sql
   ```

5. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### Setting up Oracle

The following sections describe how to replace the default H2 databases with Oracle:
Setting up the database and users

Follow the steps below to set up a Oracle database.

1. Create a new database by using the Oracle database configuration assistant (dbca) or manually.

2. Make the necessary changes in the Oracle tnsnames.ora and listener.ora files in order to define addresses of the databases for establishing connections to the newly created database.

3. After configuring the .ora files, start the Oracle instance using the following command:

   $ sudo /etc/init.d/oracle-xe restart

4. Connect to Oracle using SQL*Plus as SYSDBA as follows:

   $ ./$<ORACLE_HOME>/config/scripts/sqlplus.sh sysadm/password as SYSDBA

5. Connect to the instance with the username and password using the following command:

   $ connect

6. As SYSDBA, create a database user and grant privileges to the user as shown below:

   ```
   Create user <USER_NAME> identified by password account unlock;
   grant connect to <USER_NAME>;
   grant create session, create table, create sequence, create trigger to <USER_NAME>;
   alter user <USER_NAME> quota <SPACE_QUOTA_SIZE_IN_MEGABYTES> on '<TABLE_SPACE_NAME>';  
   commit;
   ```

7. Exit from the SQL*Plus session by executing the `quit` command.

Setting up the JDBC driver

1. Copy the Oracle JDBC libraries (for example, <ORACLE_HOME>/jdbc/lib/ojdbc14.jar) to the <PRODUCT_HOME>/repository/components/lib/ directory.

2. Remove the old database driver from the <PRODUCT_HOME>/repository/components/dropins/ directory.

   If you get a `timezone region not found` error when using the ojdbc6.jar with WSO2 servers, set the Java property as follows: `export JAVA_OPTS="-Duser.timezone='+05:30'"`

   The value of this property should be the GMT difference of the country. If it is necessary to set this property permanently, define it inside the `wso2server.sh` as a new `JAVA_OPTS` property.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used
to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB data source, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:oracle:thin:@SERVER_NAME:PORT/DB_NAME</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
</tbody>
</table>
maxWait | The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.

minIdle | The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.

testOnBorrow | The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

validationQuery | The SQL query that will be used to validate connections from this pool before returning them to the caller.

validationInterval | The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

defaultAutoCommit | This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

The default port for Oracle is 1521.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>

Creating the database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL*Plus:

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

   SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql

2. Restart the server.

You can create database tables automatically **when starting the product for the first time** by using the -Dsetup parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

Changing the identity/storage databases

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you set up a separate database for identity/storage related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.
2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/oracle.sql</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>For the storage database</td>
<td>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/storage/oracle.sql</code></td>
</tr>
</tbody>
</table>

Setting up Oracle RAC

The following sections describe how to replace the default H2 databases with Oracle RAC:

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables

Oracle Real Application Clusters (RAC) is an option for the Oracle Database for clustering and high availability in
Oracle database environments. In the Oracle RAC environment, some of the commands used in `oracle.sql` are considered inefficient. Therefore, the product has a separate SQL script `oracle_rac.sql` for Oracle RAC. The Oracle RAC-friendly script is located in the `dbscripts` folder together with other `.sql` scripts.

To test products on Oracle RAC, rename `oracle_rac.sql` to `oracle.sql` before running `-Dsetup`.

### Setting up the database and users

Follow the steps below to set up an Oracle RAC database.

1. Set environment variables `<ORACLE_HOME>`, `PATH`, and `ORACLE_SID` with the corresponding values `/oracle/app/oracle/product/11.2.0/dbhome_1`, `$PATH:$ORACLE_HOME/bin`, and `orcl1` as follows:

   ```
   [oracle@node1 ~]$ export ORACLE_HOME=/oracle/app/oracle/product/11.2.0/dbhome_1
   [oracle@node1 ~]$ export PATH=$PATH:$ORACLE_HOME/bin
   [oracle@node1 ~]$ export ORACLE_SID=orcl1
   ```

2. Connect to Oracle using SQL*Plus as SYSDBA.

   ```
   [oracle@node1 ~]$ sqlplus SYSDBA/1 as sysdba
   SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 18 18:10:42 2011
   Copyright (c) 1982, 2009, Oracle.  All rights reserved.
   Connected to:
   Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
   With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
   Data Mining and Real Application Testing options
   SQL> select 2+2 from dual;
   2+2
   -------
   4
   SQL> create user dbgreg identified by dbgreg account unlock;
   User created.
   SQL> grant connect to dbgreg;
   Grant succeeded.
   SQL> grant create session, dba to dbgreg;
   Grant succeeded.
   SQL> commit;
   Commit complete.
   SQL> quit
   Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
   With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
   Data Mining and Real Application Testing options
   [oracle@node1 ~]$ 
   ```

3. Create a database user and grant privileges to the user as shown below:
Create user `<USER_NAME>` identified by password account unlock;
grant connect to `<USER_NAME>`;
grant create session, create table, create sequence, create trigger to `<USER_NAME>`;
alter user `<USER_NAME>` quota `<SPACE_QUOTA_SIZE_IN_MEGABYTES>` on '<TABLE_SPACE_NAME>';
commit;

4. Exit from the SQL*Plus session by executing the `quit` command.

**Setting up the JDBC driver**

Copy the Oracle JDBC libraries (for example, the `<ORACLE_HOME>/jdbc/lib/ojdbc14.jar` file) to the `<PRODUCT_HOME>/repository/components/lib/` directory.

Remove the old database driver from the `<PRODUCT_HOME>/repository/components/dropins/` directory when you upgrade the database driver.

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle RAC database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on) (ADDRESS=(PROTOCOL=TCP)(HOST=racnode1) (PORT=1521)) (ADDRESS=(PROTOCOL=TCP)(HOST=racnode2) (PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=rac)))</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1 FROM DUAL</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
</tbody>
</table>
### validationQuery
The SQL query that will be used to validate connections from this pool before returning them to the caller.

### validationInterval
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

### defaultAutoCommit
This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

---

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/ master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">  
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>  
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.

   ```xml
   <Configuration>  
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>  
   </Configuration>
   ```

**Creating the database tables**

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL*Plus:

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:
2. Restart the server.

```sql
SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
```

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Setting up PostgreSQL**

The following sections describe how to replace the default H2 databases with PostgreSQL:

- Setting up the database and login role
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the identity database

**Setting up the database and login role**

Follow the steps below to set up a PostgreSQL database.

1. Install PostgreSQL on your computer as follows:
   ```bash
   client@wso2:~$ctb$ sudo apt-get install postgresql
   ```

2. Start the PostgreSQL service using the following command:
   ```bash
   client@wso2:~$ctb$ sudo /etc/init.d/postgresql start
   Starting PostgreSQL 8.4 database server: main.
   client@wso2:~$ctb$
   ```

3. Create a database and the login role from a GUI using the PGAdminIII tool.
4. To connect PGAdminIII to a PostgreSQL database server, locate the server from the object browser, right-click the client, and click **Connect**. This will show you the databases, tablespaces, and login roles as follows:
5. To create a database, click **Databases** in the tree (inside the object browser), and click **New Database**.
6. In the **New Database** dialog box, give a name to the database (for example: gregdb) and click **OK**.
7. To create a login role, click **Login Roles** in the tree (inside the object browser), and click **New Login Role**. Enter the role name and a password.

These values will be used in the product configurations as described in the following sections. In the sample configuration, **gregadmin** will be used as both the role name and the password.

8. Optionally enter other policies, such as the expiration time for the login and the connection limit.
9. Click **OK** to finish creating the login role.

**Setting up the drivers**

1. Download the **PostgreSQL JDBC4 driver**.
2. Copy the driver to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory.

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, **WSO2_CARBON_DB** datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the PostgreSQL database to replace the default H2 database, either change the default configurations of the **WSO2_CARBON_DB** datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.
1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:postgresql://localhost:5432/gregdb</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>org.postgresql.Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <defaultAutoCommit>true</defaultAutoCommit>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a PostgreSQL instance is 5432.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user.</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver.</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
</tbody>
</table>
### testOnBorrow
Whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.

### validationInterval
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

### defaultAutoCommit
Whether to commit database changes automatically or not. This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see **Tomcat JDBC Connection Pool**.

---

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the **WSO2_CARBON_DB datasource** above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.
1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

```sql
<PRODUCT_HOME>/dbscripts/postgresql.sql
```

2. Restart the server.

You can create database tables automatically **when starting the product for the first time** by using the -Dsetup parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Changing the identity database**

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.
2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use &lt;PRODUCT_HOME&gt;/dbscripts/identity/postgresql.sql</th>
</tr>
</thead>
</table>

**Setting up MariaDB**

The following sections describe how to replace the default H2 databases with MariaDB, which is a drop-in replacement for MySQL:

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the identity/storage databases

**Setting up the database and users**

Follow the steps below to set up MariaDB. See Tested DBMSs for information on the MariaDB versions that we have tested the WSO2 products with.

1. Download, install and start MariaDB on your computer. See [https://downloads.mariadb.org/](https://downloads.mariadb.org/).

You can install MariaDB standalone or as a galera cluster for high availability. Database clustering is independent of WSO2 product clustering. For more information on setting up a galera cluster, see the MariaDB Galera Cluster documentation.
2. Log in to MariaDB as the root user (or any other user with database creation privileges).
   mysql -u root -p
3. Enter the password when prompted.

   In most systems, there is no default root password. Press the Enter key without typing anything if you
   have not changed the default root password.

4. In the MySQL command prompt, create the database using the following command:
   create database regdb;
5. Give authorization of the database to the regadmin user as follows:
   GRANT ALL ON regdb.* TO regadmin@localhost IDENTIFIED BY "regadmin";
6. Once you have finalized the permissions, reload all the privileges by executing the following command:
   FLUSH PRIVILEGES;
7. Log out from the MySQL prompt by executing the following command:
   quit;

**Setting up the drivers**

Download the MySQL Java connector JAR file, and copy it to the `<PRODUCT_HOME>/repository/components/lib/` directory.

**Note** that you must use the MySQL connector that is compatible with your MariaDB version. For example, `mysql-connector-java-5.1.36-bin.jar` is compatible with MariaDB version 10.0.20. See Tested DBMSs for information on which version of a WSO2 product has been tested for compatibility with which version of MariaDB and MySQL connector.

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, `WSO2_CARBON_DB` datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the MariaDB database to replace the default H2 database, either change the default configurations of the `WSO2_CARBON_DB` datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

Follow the steps below to change the type of the default `WSO2_CARBON_DB` datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

   Do not change the datasource name `WSO2_CARBON_DB` in the below configuration.
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/regdb</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <defaultAutoCommit>false</defaultAutoCommit>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MariaDB is 3306</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
</tbody>
</table>
### validationQuery
The SQL query that will be used to validate connections from this pool before returning them to the caller.

### validationInterval
The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

### defaultAutoCommit
This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default.

When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.

---

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

---

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/` user-mgt.xml file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

---

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:
You may have to enter the password for each command when prompted.

```
mysql -u regadmin -p -Dregdb < '<PRODUCT_HOME>/dbscripts/mysql.sql';
```

2. Restart the server.

You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

### Changing the identity/storage databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you set up a separate database for identity/storage related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file.
2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the identity database</th>
<th>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/mysql.sql</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>For the storage database</td>
<td>Use <code>&lt;PRODUCT_HOME&gt;/dbscripts/storage/mysql.sql</code></td>
</tr>
</tbody>
</table>

### Managing Datasources

A datasource provides information that a server can use to connect to a database. Datasource management is provided by the following feature in the WSO2 feature repository:

**Name**: WSO2 Carbon - datasource management feature  
**Identifier**: org.wso2.carbon.datasource.feature.group

If datasource management capability is not included in your product by default, add it by installing the above feature, using the instructions given under the Feature Management section of this documentation.

Click **Data Sources** on the **Configure** tab of the product's management console to view, edit, and delete the datasources in your product instance.

You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** on the **Configure** tab of the product management console. However, you cannot edit or delete the default `<WSO2_CARBON_DB>` datasource.
Adding Datasources

If the datasource management feature is installed in your WSO2 product instance, you can add datasources that allow the server to connect to databases and other external data stores.

Use the following steps to add a datasource:

1. In the product management console, click **Data Sources** on the **Configure** tab.

2. Click **Add Data Source**.
3. Specify the required options for connecting to the database. The available options are based on the type of datasource you are creating:
   - Configuring a RDBMS Datasource
   - Configuring a Custom Datasource

After adding datasources, they will appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to `<PROD UCT_HOME>/repository/components/lib`.

Configuring an RDBMS Datasource

When adding a datasource, if you select RDBMS as the datasource type, the following screen appears:
This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the custom datasource option. Enter values for the following fields when using the default RDBMS datasource configuration:

- **Data Source Type**: RDBMS
- **Name**: Name of the datasource (must be a unique value)
- **Data Source Provider**: Specify the datasource provider.
- **Driver**: The class name of the JDBC driver to use. Make sure to copy the JDBC driver relevant to the database engine to the `<PRODUCT_HOME>/repository/components/lib/` directory. For example, if you are using MySQL, specify `com.mysql.jdbc.Driver` as the driver and copy `mysql-connector-java-5.xx-bin.jar` file to this directory. If you do not copy the driver to this directory when you create the datasource, you will get an exception similar to `Cannot load JDBC driver class com.mysql.jdbc.Driver`.
- **URL**: The connection URL to pass to the JDBC driver to establish the connection.
- **User Name**: The connection user name that will be passed to the JDBC driver to establish the connection.
- **Password**: The connection password that will be passed to the JDBC driver to establish the connection.
- **Expose as a JNDI Data Source**: Allows you to specify the JNDI datasource.
- **Data Source Configuration Parameters**: Allows you to specify the datasource connection pool parameters when creating a RDBMS datasource.

For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

After creating datasources, they appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

### Configuring the Datasource Provider

A datasource provider connects to a source of data such as a database, accesses its data, and returns the results of the access queries. When creating a RDBMS datasource, use the default provider or link to an external provider.

**Default datasource provider**

To use the default datasource provider, select **default**, and then enter the Driver, URL, User Name, and
Password connection properties as follows:

**External datasource provider**

If you need to add a datasource supported by an external provider class such as `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`, select **External Data Source**, click **Add Property**, and then enter the name and value of each connection property you need to configure. Following is an example datasource for an external datasource provider:

**Configuring a JNDI Datasource**

Java Naming and Directory Interface (JNDI) is a Java Application Programming Interface (API) that provides naming and directory functionality for Java software clients, to discover and look up data and objects via a name. It helps decoupling object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI look-up and use it.

When adding a datasource, to expose a RDBMS datasource as a JNDI datasource, click **Expose as a JNDI Data**
Source to display the JNDI fields as follows:

Following are descriptions of the JNDI fields:

- **Name**: Name of the JNDI datasource that will be visible to others in object look-up.
- **Use Data Source Factory**: To make the datasource accessible from an external environment, you must use a datasource factory. When this option is selected, a reference object will be created with the defined datasource properties. The datasource factory will create the datasource instance based on the values of the reference object when accessing the datasource from an external environment. In the datasource configuration, this is set as: `<jndiConfig useDataSourceFactory="true">`.
- **JNDI Properties**: Properties related to the JNDI datasource (such as password).
  When you select this option, set the following properties:
  - `java.naming.factory.initial`: Selects the registry service provider as the initial context.
  - `java.naming.provider.url`: Specifies the location of the registry when the registry is being used as the initial context.

**Configuring the Datasource Connection Pool Parameters**

When the server processes a database operation, it spawns a database connection from an associated datasource. After using this connection, the server returns it to the pool of connections. This is called datasource connection pooling. It is a recommended way to gain more performance/throughput in the system. In datasource connection pooling, the physical connection is not dropped with the database server, unless it becomes stale or the datasource connection is closed.

RDBMS datasources in WSO2 products use Tomcat JDBC connection pool (`org.apache.tomcat.jdbc.pool`). It is common to all components that access databases for data persistence, such as the registry, user management (if configured against a JDBC userstore), etc.

You can configure the datasource connection pool parameters, such as how long a connection is persisted in the
pool, using the datasource configuration parameters section that appears in the product management console when creating a datasource. Click and expand the option as shown below:
### Add New Data Source

<table>
<thead>
<tr>
<th>New Data Source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSource Id*</td>
<td>oracle-ds</td>
</tr>
<tr>
<td>Data Source Type*</td>
<td>JDBC (Type)</td>
</tr>
<tr>
<td>Database Engine*</td>
<td>Oracle</td>
</tr>
<tr>
<td>Driver Class*</td>
<td>oracle.jdbc.driver.OracleDriver</td>
</tr>
<tr>
<td>URL*</td>
<td>jdbc:oracle:[driverType]:[username/password]@[host][port]/[database]</td>
</tr>
<tr>
<td>User Name</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td></td>
</tr>
</tbody>
</table>

**Data Source Configuration Parameters**

<table>
<thead>
<tr>
<th>Transaction Isolation</th>
<th>TRANSACTION_UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Size</td>
<td></td>
</tr>
<tr>
<td>Max. Active</td>
<td></td>
</tr>
<tr>
<td>Max. Idle</td>
<td></td>
</tr>
<tr>
<td>Min. Idle</td>
<td></td>
</tr>
<tr>
<td>Max. Wait</td>
<td></td>
</tr>
<tr>
<td>Validation Query</td>
<td></td>
</tr>
<tr>
<td>Test On Return</td>
<td>false</td>
</tr>
<tr>
<td>Test On Borrow</td>
<td>true</td>
</tr>
<tr>
<td>Test While Idle</td>
<td>false</td>
</tr>
<tr>
<td>Time Between Eviction Runs MILS</td>
<td></td>
</tr>
<tr>
<td>Minimum Evictable Idle Time</td>
<td></td>
</tr>
<tr>
<td>Remove Abandoned</td>
<td>false</td>
</tr>
<tr>
<td>Remove Abandoned Timeout</td>
<td></td>
</tr>
<tr>
<td>Log Abandoned</td>
<td>false</td>
</tr>
<tr>
<td>Default Auto Commit</td>
<td>false</td>
</tr>
<tr>
<td>Default Read Only</td>
<td>false</td>
</tr>
<tr>
<td>Default Catalog</td>
<td></td>
</tr>
<tr>
<td>Validator Class Name</td>
<td></td>
</tr>
<tr>
<td>Connection Properties</td>
<td></td>
</tr>
<tr>
<td>Init SQL</td>
<td></td>
</tr>
<tr>
<td>JDBC Interceptors</td>
<td></td>
</tr>
<tr>
<td>Validation Interval</td>
<td></td>
</tr>
<tr>
<td>JMX Enabled</td>
<td>false</td>
</tr>
<tr>
<td>Fair Queue</td>
<td>false</td>
</tr>
<tr>
<td>Abandon When Percentage Full</td>
<td></td>
</tr>
<tr>
<td>Max Age</td>
<td></td>
</tr>
<tr>
<td>Use Equals</td>
<td>false</td>
</tr>
</tbody>
</table>
Following are descriptions of the parameters you can configure. For more details on datasource configuration parameters, see [Apache Tomcat JDBC Connection Pool guide](http://tomcat.apache.org/-doc/7.0/jdbc/jdbc-pool-config.html).

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction isolation</td>
<td>The default <code>TransactionIsolation</code> state of connections created by this pool are as follows:</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_NONE</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_READ_COMMITTED</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_READ_UNCOMMITTED</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_REPEATABLE_READ</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_SERIALIZABLE</td>
</tr>
<tr>
<td>Initial Size (int)</td>
<td>The initial number of connections created, when the pool is started. Default value is zero.</td>
</tr>
<tr>
<td>Max. Active (int)</td>
<td>Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.</td>
</tr>
<tr>
<td>Max. Idle (int)</td>
<td>Maximum number of connections that should be kept in the pool at all times. Default value is 8. Idle connections are checked periodically (if enabled), and connections that have been idle for longer than <code>minEvictableIdleTimeMillis</code> will be released. (also see <code>testWhileIdle</code>)</td>
</tr>
<tr>
<td>Min. Idle (int)</td>
<td>Minimum number of established connections that should be kept in the pool at all times. The connection pool can shrink below this number, if validation queries fail. Default value is zero. For more information, see <code>testWhileIdle</code>.</td>
</tr>
<tr>
<td>Max. Wait (int)</td>
<td>Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. Default value is 30000 (30 seconds).</td>
</tr>
<tr>
<td>Validation Query (String)</td>
<td>The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are <code>SELECT 1</code> (mysql), <code>select 1 from dual</code> (oracle), <code>SELECT 1</code> (MS Sql Server).</td>
</tr>
<tr>
<td>Test On Return (boolean)</td>
<td>Used to indicate if objects will be validated before returned to the pool. The default value is false.</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Test On Borrow (boolean)</td>
<td>Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. Default value is false.</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code>.</td>
</tr>
<tr>
<td>Test While Idle (boolean)</td>
<td>The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. The default value is false and this property has to be set in order for the pool cleaner/test thread to run. For more information, see <code>timeBetweenEvictionRunsMillis</code>.</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</td>
</tr>
<tr>
<td>Time Between Eviction Runs Mills (int)</td>
<td>Number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. This value should not be set under 1 second. It indicates how often we check for idle, abandoned connections, and how often we validate idle connections. The default value is 5000 (5 seconds).</td>
</tr>
<tr>
<td>Minimum Evictable Idle Time (int)</td>
<td>Minimum amount of time an object may sit idle in the pool before it is eligible for eviction. The default value is 60000 (60 seconds).</td>
</tr>
<tr>
<td>Remove Abandoned (boolean)</td>
<td>Flag to remove abandoned connections if they exceed the <code>removeAbandonedTimeout</code>. If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <code>removeAbandonedTimeout</code>. Setting this to true can recover database connections from applications that fail to close a connection. For more information, see <code>logAbandoned</code>. The default value is false.</td>
</tr>
<tr>
<td>Remove Abandoned Timeout (int)</td>
<td>Timeout in seconds before an abandoned (in use) connection can be removed. The default value is 60 (60 seconds). The value should be set to the longest running query that your applications might have.</td>
</tr>
<tr>
<td>Log Abandoned (boolean)</td>
<td>Flag to log stack traces for application code which abandoned a connection. Logging of abandoned connections, adds overhead for every connection borrowing, because a stack trace has to be generated. The default value is false.</td>
</tr>
<tr>
<td>Auto Commit (boolean)</td>
<td>The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the <code>setAutoCommit</code> method will not be called.</td>
</tr>
<tr>
<td>Default Read Only (boolean)</td>
<td>The default read-only state of connections created by this pool. If not set then the <code>setReadOnly</code> method will not be called. (Some drivers don't support read only mode. For example: Informix)</td>
</tr>
<tr>
<td>Default Catalog (String)</td>
<td>The default catalog of connections created by this pool.</td>
</tr>
<tr>
<td><strong>Validator Class Name (String)</strong></td>
<td>The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.ValidationError</code> interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a <code>Validator</code> instance, which is then used instead of any validation query to validate connections. The default value is null. An example value is <code>com.mycompany.project.SimpleValidator</code>.</td>
</tr>
<tr>
<td><strong>Connection Properties (String)</strong></td>
<td>Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]</code>*. The default value is null. The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.</td>
</tr>
<tr>
<td><strong>Init SQL</strong></td>
<td>Ability to run a SQL statement exactly once, when the connection is created.</td>
</tr>
<tr>
<td><strong>JDBC Interceptors</strong></td>
<td>Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.</td>
</tr>
<tr>
<td><strong>Validation Interval (long)</strong></td>
<td>To avoid excess validation, only run validation at most at this frequency - time in milliseconds. If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).</td>
</tr>
<tr>
<td><strong>JMX Enabled (boolean)</strong></td>
<td>Register the pool with JMX or not. The default value is true.</td>
</tr>
<tr>
<td><strong>Fair Queue (boolean)</strong></td>
<td>Set to true, if you wish that calls to <code>getConnection</code> should be treated fairly in a true FIFO fashion. This uses the <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue</code> implementation for the list of the idle connections. The default value is true. This flag is required when you want to use asynchronous connection retrieval. Setting this flag ensures that threads receive connections in the order they arrive. During performance tests, there is a very large difference in how locks and lock waiting is implemented. When <code>fairQueue=true</code>, there is a decision making process based on what operating system the system is running. If the system is running on Linux (<code>property os.name=Linux</code>), then to disable this Linux specific behavior and still use the fair queue, simply add the property <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true</code> to your system properties, before the connection pool classes are loaded.</td>
</tr>
<tr>
<td><strong>Abandon When Percentage Full (int)</strong></td>
<td>Connections that have been abandoned (timed out) will not get closed and reported up, unless the number of connections in use are above the percentage defined by <code>abandonWhenPercentageFull</code>. The value should be between 0-100. The default value is zero, which implies that connections are eligible for closure as soon as <code>removeAbandonedTimeout</code> has been reached.</td>
</tr>
<tr>
<td><strong>Max Age (long)</strong></td>
<td>Time in milliseconds to keep this connection. When a connection is returned to the pool, the pool will check to see if the current time when connected, is greater than the <code>maxAge</code> that has been reached. If so, it closes the connection rather than returning it to the pool. The default value is zero, which implies that connections will be left open and no age check will be done upon returning the connection to the pool.</td>
</tr>
<tr>
<td><strong>Use Equals (boolean)</strong></td>
<td>Set to true, if you wish the <code>ProxyConnection</code> class to use <code>String.equals</code>, and set to false when you wish to use <code>==</code> when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.</td>
</tr>
</tbody>
</table>
### Suspect Timeout (int)

Timeout value in seconds. Default value is zero. Similar to the `removeAbandonedTimeout` value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if `logAbandoned` is set to true. If this value is equal or less than zero, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than zero, and the connection was not abandoned, or if abandon check is disabled. If a connection is suspected, a warning message gets logged and a JMX notification will be sent.

### Alternate User Name Allowed (boolean)

By default, the `jdbc-pool` will ignore the `DataSource.getConnection(username, password)` call, and simply return a previously pooled connection under the globally configured properties `username` and `password`, for performance reasons.

The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the `DataSource.getConnection(username, password)` call, simply set the property `alternateUsernameAllowed`, to true. If you request a connection with the credentials user1/password1, and the connection was previously connected using different user2/password2, then the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per-schema level. The default value is false.

### Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

![New Data Source](image)

Following are descriptions of the custom datasource fields:

- **Data Source Type**: Custom
- **Custom Data Source Type**: Specify whether the data is in a table or accessed through a query as described below.
- **Name**: Enter a unique name for this datasource
- **Description**: Description of the datasource
- **Configuration**: XML configuration of the datasource

### Custom datasource type

When creating a custom datasource, specify whether the datasource type is `DS_CUSTOM_TABULAR` (the data is
stored in tables), or DS_CUSTOM_QUERY (non-tabular data accessed through a query). More information about each type are explained below.

**Custom tabular datasources**

Tabular datasources are used for accessing tabular data, that is, the data is stored in rows in named tables that can be queried later. To implement tabular datasources, the interface org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS is used. For more information, see a sample implementation of a tabular custom datasource at InMemoryDataSource.

A tabular datasource is typically associated with a SQL data services query. WSO2 products use an internal SQL parser to execute SQL against the custom datasource. For more information, see a sample data service descriptor at InMemoryDSSample. Carbon datasources also support tabular data with the org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS datasource reader implementation. If you have Data Services Server installed, for more information see the <PRODUCT_HOME>/repository/conf/datasources/custom-datasources.xml file, which is a sample Carbon datasource configuration.

**Custom query datasources**

Custom query-based datasources are used for accessing non-tabular data through a query expression. To implement query-based datasources, the org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS interface is used. You can create any non-tabular datasource using the query-based approach. Even if the target datasource does not have a query expression format, you can create and use your own. For example, you can support any NoSQL type datasource using this type of a datasource.

For more information, see a sample implementation of a custom query-based datasource at EchoDataSource, and a sample data service descriptor with custom query datasources in InMemoryDSSample. Carbon datasources also support query-based data with the org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryDataSourceReader datasource reader implementation. If you have Data Services Server installed, for more information, see the <PRODUCT_HOME>/repository/conf/datasources/custom-datasources.xml file, which is a sample Carbon datasource configuration.

In the init methods of all custom datasources, user-supplied properties will be parsed to initialize the datasource accordingly. Also, a property named __DATASOURCE_ID__, which contains a UUID to uniquely identify the current datasource, will be passed. This can be used by custom datasource authors to identify the datasources accordingly, such as datasource instances communicating within a server cluster for data synchronization.

Shown below is an example configuration of a custom datasource of type DS_CUSTOM_TABULAR:
After creating datasources, they will appear on the Data Sources page. You can edit and delete them as needed by clicking Edit or Delete links.

Registry

This chapter contains the following information:

- Introduction to Registry
- Managing the Registry
- Searching the Registry
- Sharing Registry Space Among Multiple Products

Introduction to Registry

A registry is a content store and a metadata repository. Various SOA artifacts such as services, WSDLs and configuration files can be stored in a registry, keyed by unique paths. A path is similar to a Unix file path. In WSO2 products, all configurations pertaining to modules, logging, security, data sources and other service groups are stored in the registry by default.

The Registry kernel of WSO2 Carbon provides the basic registry and repository functionality. Products based on Carbon use the services provided by the Registry kernel to establish their own registry space, which is utilized for storing data and persisting configuration. Here are some of the features provided by the WSO2 Registry interface:

- Provides the facility to organize resources into collections.
- Keeps multiple versions of resources.
- Manages social aspects such as rating of resources.
- AtomPub interfaces to publish, view and manage resources from remote or non-Java clients.

The Registry space provided to each Carbon product contains three major partitions.

- **Local Data Repository** - Used to store settings/metadata specific to the product. This registry is not intended to be shared among multiple servers.
• **Configuration Registry** - Used to store product-specific configurations. These configurations can be shared across multiple instances of the same product like a cluster.
• **Governance Registry** - Used to store user-specified metadata and resources and can be shared across an organization.

These registry instances are mounted to a single top level registry to provide a single, unified view. Mount points of the three registries are /_system/local, /_system/config and /_system/governance respectively. One could browse the contents of the registry used by the Carbon product through its management console.

**Managing the Registry**

Follow the instructions below to access the registry user interface.

1. Log on to the product’s Management Console and select *Browse* under *Registry*.

2. The *Browse* page appears.

**Components of Registry User Interface**

- **Breadcrumb** - Shows the current directory hierarchy.
- **Metadata** - Shows metadata for the resource/collection.
- **Properties** - Shows properties for the resource/collection.
- **Entries** - Shows the contents of the resource/collection.
- **Permissions** - Shows the defined role permissions to use the resource/collection.

**Managing Breadcrumb**

Use the breadcrumb to navigate backward in the current branch of the directory path by clicking on a directory name in the breadcrumb.

1. In the *Browse* window, click the *Tree View* tab to see the branch.
2. Click on a particular directory name to see its details in the *Detail view* tab.

**Note**

You can access the root of the directory path by clicking the *Root* icon in the breadcrumb.

### Managing Metadata

The *Metadata* panel displays the following properties of the resource or the collection:

- **Created** - Shows the time when a resource was created and the author of a resource/collection.
- **Last Updated** - Shows the time when a resource was updated and the author of alterations/collection.
- **Media Type** - An associated media type of the resource/collection.
- **Checkpoint** - Allows to create a checkpoint (URL for the permanent link) of a resource/collection.
- **Versions** - Allows to view versions of a resource/collection.
- **Description** - Description of the resource/collection.

For example,

![Metadata](image)

**Checkpoint Creation**

1. To create a checkpoint, click on the *Create Checkpoint* link.

![Metadata](image)

2. If the checkpoint was successfully created, a message will be displayed. Click *OK*.

![Checkpoint Created Successfully](image)

**Viewing Versions**
1. Click on the *View versions* link.

2. The *Versions* page appears.

   - **Version** - Shows the number of a resource/collection version.
   - **Last Modified Date** - Shows the last date of updating.
   - **Last Modified By** - Shows the author of alterations.
   - **Actions**
     - **Details** - Allows to get to the *Browse* page of a particular resource/collection version.
     - **Restore** - Allows to restore a resource/collection version.

**Editing Description**

1. To edit a description of a resource/collection, click on the *Edit* link.
2. Edit the description of a resource/collection in the text area and click Save.

Managing Properties

The Properties panel displays the properties of the currently selected resource or collection. New properties can be added, while existing properties can be edited or deleted.

Adding a Property

1. To add a property, click on the Add New Property link.

2. In the Add New Property panel, enter a unique name of a property and its value. Click Add.
Editing a Property

1. Click on the Edit link of a particular property in the Action column.

2. Edit the name and the value of a property in the active fields and click Save.

Deleting a Property

1. To delete a property, click on the Delete link of a certain property in the Action column.

2. Confirm your request by clicking Yes.
Managing Entries and Content

If the currently selected entity is a collection, the Content panel is called the Entries panel and shows the child entries under that collection. It provides details of each entry. An entry can be either another collection or a resource. Here you can also add a new resource, add a new collection and create links.

If the currently selected entity is a resource, the Content panel provides a user interface through which one can display, edit, upload, and download the content.

The Entries Panel

The following information is given as shown in the example screenshot below.

- Add Resource
- Add Collection
- Create Link
- Child Resources - The list of child entries provides the following information:
  - Name - The name of a child resource.
  - Created On - The date when a child resource was created.
  - Author - The author who created a child resource.

You can also see the detailed information about the resource by clicking on the Info icon. The following information is available as shown in the example screenshot below.

- Media Type
- Feed
- Rating

To see the available actions over a resource, click on the Actions icon.
The following actions over the resources are available:

- **Rename** - Allows to rename a resource.
- **Move** - Allows to move a resource to a new directory.
- **Delete** - Allows to delete a resource.
- **Copy** - Allows to copy a resource to a specified directory.

**Tip**

All these options are available not for all the resources.

### Renaming a Resource

1. To rename a resource, click *Rename* and enter a new name to the field.

2. Click on the *Rename* button to save a new name of a resource.

### Moving a Resource

1. To move a resource to a new directory, click *Move* and specify *Destination Path*.

2. Click *Move*.

### Deleting a Resource

1. To delete a resource, click *Delete* and confirm your request by clicking *Yes* in the message that appears if the resource is deleted successfully.
Copying a Resource

1. To copy a resource to some directory, click Copy and specify Destination Path.

2. Click Copy.

If the resource was successfully copied, a message appears. Click OK.

Adding a Collection

Follow the instructions below to add a new collection.

1. To add a new collection, click Add Collection.

2. Specify the following options:
   - **Name** - The unique name of a collection.
   - **Media Type** - Select media type of a collection from the drop-down menu:
     - application/vnd.wso2.esb
     - application/vnd.apache.synapse
     - application/vnd.apache.axis2
     - application/vnd.wso2.wsas
     - Other
3. Click Add.

Adding a Resource

You can add a resource to a certain collection for more convenient usage of the Registry.

Follow the instructions below to add a new child entry to a collection.

1. To add a new resource, click on the Add Resource link.

   ![Add Resource Link]

2. In the Add Resource panel, select Method from the drop-down menu.

   The following methods are available:

   - Upload content from file
   - Import content from URL
   - Create Text content
   - Create custom content

   ![Method Selection]

   Uploading Content from File

   1. If this method was selected, specify the following options:

      - File - The path of a file to fetch content (XML, WSDL, JAR etc.) Use the Browse button to upload a file.
      - Name - The unique name of a resource.
      - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
      - Description - The description of a resource.

   2. Click Add once the information is added as shown in the example below.
Importing Content from URL

1. If this method was selected, specify the following options:
   - URL - The full URL of the resource to fetch content from URL.
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
   - Description - The description of a resource.

2. Click Add once the information is added.

Text Content Creation

1. If this method was selected, specify the following options:
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
   - Description - The description of a resource.
   - Content - The resource content. You can use either Rich Text Editor or Plain Text Editor to enter.

2. Click Add once the information is added.
Custom Content Creation

1. If this method was selected, choose the Media Type from the drop-down menu and click Create Content.

Media Types

Each collection and resource created and stored on the repository has an associated media type. However, you also have the option to leave this unspecified enforcing the default media type. There are two main ways to configure media types for resources.

- The first method is by means of a one-time configuration, which can be done by modifying the "mime.types" file found in <CARBON_HOME>/repository/conf directory. This can be done just once before the initial start-up of the server.
- The second method is to configure the media types via the server administration console. The first method does not apply for collections, and the only available mechanism is to configure the media types via the server administration console.

Initially the system contains the media types defined in the mime.types file will be available for resources and a set of default media types will be available for collections.

Managing media types for resources can be done via the server administration console, by editing the properties of the /system/mime.types/index collection. This collection contains two resources, collection and custom.ui. To
manage media types of collections and custom user interfaces, edit the properties of these two resources.

**Link Creation**

Follow the instructions below to create a link on a resource/collection.

1. Symbolic links and Remote links can be created in a similar way to adding a normal resource. To add a link, click *Create Link* in the *Entries* panel.

2. Select a link to add from the drop-down menu.

   **A Symbolic Link**

   When adding a Symbolic link, enter a name for the link and the path of an existing resource or collection which is being linked. It creates a link to the particular resource.

   **A Remote Link**

   You can mount a collection in a remotely-deployed registry instance to your registry instance by adding a Remote link. Provide a name for the Remote link in the name field. Choose the instance to which you are going to mount and give the path of the remote collection which you need to mount for the path field, or else the root collection will be mounted.
Managing Role Permissions

The Permissions panel shows the defined role permissions, allows to add new role permissions and edit existing ones.

Adding New Role Permissions

1. In the New Role Permission pane, select a role to set a permission.

2. Select an action from the drop-down menu. The following actions are available:
   - Read
   - Write
   - Delete
   - Authorize
3. Select whether to allow the action the selected role or deny.

4. Click on the **Add Permission** button.

5. A new permission appears in the *Defined Role Permissions* list.

**Editing Role Permissions**

1. You can also edit the defined role permissions using the check boxes in the *Defined Role Permissions* list.

2. After editing the permissions, click on the **Apply All Permissions** button to save the alterations.

**Searching the Registry**

All resources found in the Registry can be searched through the product's Management Console. Search can be refined by resource name, created date range, updated date range, tags, comments, property name, property value, media type etc.
Follow the instructions below to find a necessary resource in the Registry.

1. Log on to the product's Management Console and select *Metadata* under *Registry*.

2. The *Search* window appears.
The search can be refined by:

- **Resource Name**
- **Created Date Range** - The date when a resource was created.
- **Updated Date Range** - The date when a resource was updated.
- **Update Author** - The date when a resource was updated.
- **Create Author** - The author of a resource creation.
- **Tags**
- **Comments**
- **Property Name**
- **Property Value**
- **Media Type**

**Tip**

Created or updated dates can be either entered in the format of MM/DD/YYYY or picked from the calendar interface provided.

3. Fill the search criteria and click on the *Search* button. The results are displayed in the Search Results window.

**Sharing Registry Space Among Multiple Products**

Any WSO2 Carbon-based product has the following options when configuring and using a registry space:

- Use the registry space shipped by default with the product.
- Use a remote registry instance/s for the registry partitions that can be shared across multiple WSO2 Carbon-based product instances.

This guide explains the second option using WSO2 Governance Registry as the remote registry instance.

WSO2 Carbon is the base platform for all WSO2 products and its Registry kernel provides the basic registry and repository functionality. Products based on Carbon use the services provided by the Registry kernel to establish their own registry spaces utilized for storing data and persisting configuration. The Registry space provided to each product contains three major partitions.

- **Local Repository** : Used to store configuration and runtime data that is local to the server. This partition is not to be shared with multiple servers and can be browsed under `/system/local` in the registry browser.
- **Configuration Repository** : Used to store product-specific configuration. This partition can be shared across multiple instances of the same product. (eg: sharing ESB configuration across a ESB cluster) and can be browsed under `/system/config` in the registry browser.
- **Governance Repository**: Used to store configuration and data that are shared across the whole platform. This typically includes services, service descriptions, endpoints or datasources and can be browsed under `/_system/governance` in the registry browser.

Two of these three partitions can be shared across multiple product instances in a typical production environment. Therefore, we can identify four main deployment strategies for the three partitions as follows.

- **All Partitions in a Single Server**
- **Config and Governance Partitions in a Remote Registry**
- **Governance Partition in a Remote Registry**
- **Config and Governance Partitions in Separate Nodes**

In all of the above four sections, any of the WSO2 Carbon-based products can be mounted to a remote WSO2 Governance Registry (G-Reg) instance. Examples discussed here use JDBC based configuration model as that is the recommended approach for a production deployment setup.

### All Partitions in a Single Server

**Strategy 1: Local Registry**

![Pattern #1: Local Registry](image)

Figure 1: All registry partitions in a single server instance.

The entire registry space is local to a single server instance and not shared. This is recommended for a stand-alone deployment of a single product instance, but can also be used if there are two or more instances of a product that do not require sharing data or configuration among them.

This strategy requires no additional configuration.

**Config and Governance Partitions in a Remote Registry**

In this deployment strategy, the configuration and governance spaces are shared among instances of a group/cluster. For example, two WSO2 Application Server instances that have been configured to operate in a clustered environment can have a single configuration and governance registry which is shared across each node of the cluster. A separate instance of the WSO2 Governance Registry (G-Reg) is used to provide the space used in common.
Figure 2: Config and governance partitions in the remote Governance Registry instance.

Configuration steps are given in the following sections.

- Creating the Database
- Configuring Governance Registry as the Remote Registry Instance
- Configuring Carbon Server Nodes

Creating the Database

In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named ‘registrydb’. Instructions are as follows:

1. Access MySQL using the command:

   mysql -u root -p

2. Enter the password when prompted.

3. Create ‘registrydb’ database.

   ```
   create database registrydb;
   ```

The MySQL database for G-Reg is now created.

Configuring Governance Registry as the Remote Registry Instance

Database configurations are stored in $CARBON_HOME/repository/conf/datasources/master-datasources.xml file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since Governance Registry in this example is using a MySQL database named ‘registrydb’, the master-datasources.xml file needs to be configured so that the datasource used for the registry and user manager in Governance Registry is the said MySQL database.

2. Navigate to $G-REG_HOME/repository/conf/datasources/master-datasources.xml file where $G-REG_HOME is the Governance Registry distribution home. Replace the existing WSO2_CARBON_DB datasource with the following configuration:

```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://x.x.x.x:3306/registrydb</url>
      <username>root</username>
      <password>root</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

Change the values of the following elements according to your environment.

- `<url>`: URL of the MySQL database.
- `<username>` and `<password>`: username and password of the MySQL database.
- `<validationQuery>`: Validate and test the health of the DB connection.
- `<validationInterval>`: specified time intervals at which the DB connection validations should run.

3. Navigate to $G-REG_HOME/repository/conf/axis2/axis2.xml file in all Carbon-based product instances to be connected with the remote registry, and enable tribes clustering with the following configuration.

```
<clustering
class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
enable="true"/>
```

The above configuration is required only when caching is enabled for the Carbon server instances and `<enableCache>` parameter is set to true. This provides cache invalidation at the event of any updates on the registry resources.


5. Start the Governance Registry server with -Dsetup so that all the required tables are created in the database. For example, in Linux
The Governance Registry server is now running with all required user manager and registry tables for the server also created in ‘registrydb’ database.

**Configuring Carbon Server Nodes**

Now that the shared registry is configured, let's take a look at the configuration of Carbon server nodes that use the shared, remote registry.

1. Download and extract the relevant WSO2 product distribution from the ‘Products’ menu of [https://wso2.com](https://wso2.com). In this example, we use two server instances (of any product) by the names CARBON-Node1 and CARBON-Node2.

2. We use the same datasource used for Governance Registry above as the registry space of Carbon-based product instances.

**Configure master-datasources.xml File**

3. Configure `$CARBON_HOME/repository/conf/datasource/master-datasources.xml` where `$CARBON_HOME` is the distribution home of any WSO2 Carbon-based product you downloaded in step 1. Then, add the following datasource for the registry space.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB_GREG</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB_GREG</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:mysql://x.x.x.x:3306/registrydb</url>
         <username>root</username>
         <password>root</password>
         <driverClassName>com.mysql.jdbc.Driver</driverClassName>
         <maxActive>50</maxActive>
         <maxWait>60000</maxWait>
         <testOnBorrow>true</testOnBorrow>
         <validationQuery>SELECT 1</validationQuery>
         <validationInterval>30000</validationInterval>
       </configuration>
     </definition>
   </datasource>
   ```

Change the values of the relevant elements accordingly.

**Configuring registry.xml File**

4. Navigate to `$CARBON_HOME/repository/conf/registry.xml` file and specify the following configurations for both server instances setup in step 1.
Add a new db config to the datasource configuration done in step 3 above. For example,

```
<dbConfig name="remote_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:

```
<remoteInstance url="https://x.x.x.x:9443/registry">
    <id>instanceid</id>
    <dbConfig>remote_registry</dbConfig>
    <cacheId>root@https://x.x.x.x:9443/registry</cacheId>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

Define the registry partitions using the remote Governance Registry instance. In this deployment strategy, we are mounting the config and governance partitions of the Carbon-based product instances to the remote Governance Registry instance. This is graphically represented in Figure 2 at the beginning.

```
<mount path="/_/system/config" overwrite="true">
    <instanceId>instanceid</instanceId>
    <targetPath>/_/system/nodes</targetPath>
</mount>
<mount path="/_/system/governance" overwrite="true">
    <instanceId>instanceid</instanceId>
    <targetPath>/_/system/governance</targetPath>
</mount>
```

- **mount path**: Registry collection of Carbon server instance that needs to be mounted
- **mount overwrite**: Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
  - true: The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - false: The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  - virtual: If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.
- **target path**: Path to the remote Governance Registry instance where the registry collection is mounted.

**Configuring axis2.xml File**
1. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file where `CARBON_HOME` is the distribution home of any WSO2 Carbon-based products to be connected with the remote registry. Enable carbon clustering by copying the following configuration to all Carbon server instances:

```xml
<clustering
    class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
    enable="true"/>
```


3. Start both servers and note the log entries that indicate successful mounting to the remote Governance Registry instance. For example,

```
[2012-12-14 14:02:31,389] INFO - CarbonCoreActivator Carbon Home : /home/gillian/Products/Mount/Node1
[2012-12-14 14:02:31,389] INFO - CarbonCoreActivator Java TempDir : /home/gillian/Products/Mount/Nodes/tmp
[2012-12-14 14:02:31,383] INFO - AgentDS Successfully deployed Agent Client
```

4. Navigate to the registry browser in the Carbon server's management console and note the config and governance partitions indicating successful mounting to the remote registry instance. For example,

```
Governance Partition in a Remote Registry
```
In this deployment strategy, only the governance partition is shared among instances of a group/cluster. For example, a WSO2 Application Server instance and a WSO2 ESB instance that have been configured to operate in a clustered environment can have a single governance registry which is shared across each node of the cluster. A separate instance of the WSO2 Governance Registry (G-Reg) is used to provide the space used in common.

![Governance partition in the remote Governance Registry instance](image)

Figure 3: Governance partition in the remote Governance Registry instance.

Configuration steps are given in the following sections.

- Creating the Database
- Configuring Governance Registry Instance
- Configuring Carbon Server Nodes

**Creating the Database**

In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named ‘registrydb’. Instructions are as follows:

1. Access MySQL using the command:

   ```
   mysql -u root -p
   ```

2. Enter the password when prompted.

3. Create ‘registrydb’ database.

   ```
   create database registrydb;
   ```

   The MySQL database for G-Reg is now created.

**Configuring Governance Registry Instance**

Database configurations are stored in `$CARBON_HOME/repository/conf/datasources/master-datasources.xml` file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since Governance Registry in this example is using a MySQL database named ‘registrydb’, the `master-datasources.xml` file needs to be configured so that the datasource used for the registry and user manager in Governance Registry is the said MySQL database.

2. Navigate to $G-REG_HOME/repository/conf/datasources/master-datasources.xml file where G-REG_HOME is the Governance Registry distribution home. Replace the existing WSO2_CARBON_DB datasource with the following configuration:

   ```xml
   <datasource>
   <name>WSO2_CARBON_DB</name>
   <description>The datasource used for registry and user manager</description>
   <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
   </jndiConfig>
   <definition type="RDBMS">
       <configuration>
           <url>jdbc:mysql://x.x.x.x:3306/registrydb</url>
           <username>root</username>
           <password>root</password>
           <driverClassName>com.mysql.jdbc.Driver</driverClassName>
           <maxActive>50</maxActive>
           <maxWait>60000</maxWait>
           <testOnBorrow>true</testOnBorrow>
           <validationQuery>SELECT 1</validationQuery>
           <validationInterval>30000</validationInterval>
       </configuration>
   </definition>
   </datasource>
   ```

   Change the values of the following elements according to your environment.
   - `<url>`: URL of the MySQL database.
   - `<username>` and `<password>`: username and password of the mySQL database.
   - `<validationQuery>`: Validate and test the health of the DB connection.
   - `<validationInterval>`: specified time intervals at which the DB connection validations should run.

3. Navigate to $G-REG_HOME/repository/conf/axis2/axis2.xml file in all Carbon-based product instances to be connected with the remote registry, and enable tribes clustering with the following configuration.

   ```xml
   <clustering>
       class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
       enable="true"/>
   ```

   The above configuration is required only when caching is enabled for the Carbon server instances and <enableCache> parameter is set to true. This provides cache invalidation at the event of any updates on the registry resources.


5. Start the Governance Registry server with -Dsetup so that all the required tables are created in the database. For example, in Linux
sh wso2server.sh -Dsetup

The Governance Registry server is now running with all required user manager and registry tables for the server also created in ‘registrydb’ database.

Configuring Carbon Server Nodes

Now that the shared registry is configured, let's take a look at the configuration of Carbon server nodes that use the shared, remote registry.

1. Download and extract the relevant WSO2 product distribution from the ‘Products’ menu of https://wso2.com. In this example, we use two server instances (of any product) by the names CARBON-Node1 and CARBON-Node2 and the configuration is given for one server instance. Similar steps apply to the other server instance as well.

2. We use the same datasource used for Governance Registry above as the registry space of Carbon-based product instances.

Configure master-datasources.xml File

3. Configure $CARBON_HOME/repository/conf/datasource/master-datasources.xml where CARBON_HOME is the distribution home of any WSO2 Carbon-based product you downloaded in step 1. Then, add the following datasource for the registry space.

```xml
<datasource>
  <name>WSO2_CARBON_DB_GREG</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB_GREG</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://x.x.x.x:3306/registrydb</url>
      <username>root</username>
      <password>root</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

Change the values of the relevant elements accordingly.

Configuring registry.xml File

4. Navigate to $CARBON_HOME/repository/conf/registry.xml file and specify the following configurations for both server instances setup in step 1.
Add a new db config to the datasource configuration done in step 3 above. For example,

```xml
<dbConfig name="remote_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:

```xml
<remoteInstance url="https://x.x.x.x:9443/registry">
    <id>instanceid</id>
    <dbConfig>remote_registry</dbConfig>
    <cacheId>root@https://x.x.x.x:9443/registry</cacheId>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

Define the registry partitions using the remote Governance Registry instance. In this deployment strategy, we are mounting the governance partition of the Carbon-based product instances to the remote Governance Registry instance. This is graphically represented in Figure 3 at the beginning.

```xml
<mount path="/system/governance" overwrite="true">
    <instanceId>instanceid</instanceId>
    <targetPath>/_system/governance</targetPath>
</mount>
```

- `mount path`: Registry collection of Carbon server instance that needs to be mounted
- `mount overwrite`: Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
  - `true`: The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - `false`: The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  - `virtual`: If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.
- `target path`: Path to the remote Governance Registry instance where the registry collection is mounted.

**Configuring axis2.xml File**

5. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file where CARBON_HOME is the distribution home of any WSO2 Carbon-based products to be connected with the remote registry. Enable carbon clustering by copying the following configuration to all Carbon server instances:
<clustering
class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
enable="true"/>

Note

The above configuration is needed only when caching is enabled in the server instances and <enableCache> parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.


7. Start both servers and note the log entries that indicate successful mounting to the remote Governance Registry instance. Also navigate to the registry browser in the Carbon server's management console and note the governance partition indicating successful mounting to the remote registry instance.

Config and Governance Partitions in Separate Nodes

In this deployment strategy, let's assume 2 clusters of Carbon-based product Foo and Carbon-based product Bar that share a governance registry space by the name G-Reg 1. In addition, the product Foo cluster shares a configuration registry space by the name G-Reg 2 and the product Bar cluster shares a configuration registry space by the name G-Reg 3.

Figure 4: Config and governance partitions in separate registry instances.

Configuration steps are given in the following sections.

- Creating the Database
- Configuring the Remote Registry Instances
- Configuring Foo Product Cluster
- Configuring Bar Product Cluster

Creating the Database
In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named ‘registrydb’. Instructions are as follows:

1. Access MySQL using the command:

   ```
   mysql -u root -p
   ```

   Enter the password when prompted.

2. Create ‘registrydb’ database.

   ```
   create database registrydb;
   ```

   The MySQL database for G-Reg 1 is now created. Similarly create ‘registrydb2’ and ‘registrydb3’ as the MySQL databases for G-Reg 2 and G-Reg 3 respectively.

   **Configuring the Remote Registry Instances**

   Database configurations are stored in `$CARBON_HOME/repository/conf/datasources/master-datasources.xml` file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since the Governance Registry nodes (G-Reg 1, G-Reg 2 and G-Reg 3) in this example are using MySQL databases (‘registrydb’, ‘registrydb2’ and ‘registrydb3’ respectively) the master-datasources.xml file of each node needs to be configured so that the datasources used for the registry, user manager and configuration partitions in Governance Registry are the said MySQL databases.


   2. First, navigate to `$G-REG_HOME/repository/conf/datasources/master-datasources.xml` file where $G-REG_HOME is the distribution home of Governance Registry of G-Reg 1. Replace the existing WSO2_CARBON_DB datasource with the following configuration:
Change the values of the following elements according to your environment.

- `<url>`: URL of the MySQL database.
- `<username>` and `<password>`: username and password of the MySQL database.
- `<validationQuery>`: Validate and test the health of the DB connection.
- `<validationInterval>`: specified time intervals at which the DB connection validations should run.

3. Similarly, replace the existing WSO2_CARBON_DB datasource in G-Reg 2 with the following:
4. Repeat the same for G-Reg 3 as follows.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

5. Navigate to `$G-REG_HOME/repository/conf/axis2/axis2.xml` file in all instances and enable tribes clustering with the following configuration.
<clustering
class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
enable="true"/>

The above configuration is required only when caching is enabled for the Carbon server instances and<br>
<enableCache> parameter is set to true. This provides cache invalidation at the event of any updates on the registry<br>
resources.

6. Copy the 'mySQL JDBC connector jar' (http://dev.mysql.com/downloads/connector/j/5.1.html) to<br>G-REG_HOME/repository/components/lib directory in G-Reg 1, G-Reg 2 and G-Reg 3.

7. Start the Governance Registry servers with -Dsetup so that all the required tables will be created in the<br>databases. For example, in Linux

   sh wso2server.sh -Dsetup

The Governance Registry server instances are now running with all required user manager and registry tables for<br>the server created in 'registrydb', 'registrydb1' and 'registrydb2' databases.

Configuring the Foo Product Cluster

Now that the shared registry nodes are configured, let's take a look at the configuration of Carbon server clusters<br>that share the remote registry instances. Namely, Foo product cluster shares G-Reg 1 and G-Reg 2 while Bar<br>product cluster shares G-Reg 1 and G-Reg 3.

Include the following configurations in the master node of Foo product cluster.

Configure master-datasources.xml File

1. Configure $CARBON_HOME/repository/conf/datasource/master-datasources.xml where CARBON_HOME is the<br>distribution home of any WSO2 Carbon-based product. Then, add the following datasource for the registry space.
<datasource>
    <name>WSO2_CARBON_DB_GREG</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB_GREG</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://10.20.30.41:3306/registrydb</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

<datasource>
    <name>WSO2_CARBON_DB_GREG_CONFIG</name>
    <description>The datasource used for configuration partition</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB_GREG_CONFIG</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

Change the values of the relevant elements according to your environment.

**Configuring registry.xml File**

2. Navigate to `$CARBON_HOME/repository/conf/registry.xml` file and specify the following configurations.

Add a new db config to the datasource configuration done in step 1 above. For example,
Specify the remote Governance Registry instance with the following configuration:

```xml
<remoteInstance url="https://10.20.30.41:9443/registry">
  <id>governanceRegistryInstance</id>
  <dbConfig>governance_registry</dbConfig>
  <cacheId>root@https://10.20.30.41:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

<remoteInstance url="https://10.20.30.42:9443/registry">
  <id>configRegistryInstance</id>
  <dbConfig>config_registry</dbConfig>
  <cacheId>root@https://10.20.30.42:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- **<remoteInstance url>** : URL of the remote G-Reg instance.
- **<dbConfig>** : The dbConfig name specified for the registry database configuration.
- **<cacheId>** : This provides information on where the cache resource resides.
- **<enableCache>** : Whether caching is enabled on the Carbon server instance.

**Note**

When adding the corresponding configuration to the registry.xml file of a slave node, set `<readOnly>true</readOnly>`. This is the only configuration change.

Define the registry partitions using the remote Governance Registry instance.
<mount path="/system/config" overwrite="true">
  <instanceId>configRegistryInstance</instanceId>
  <targetPath>/system/nodes</targetPath>
</mount>

<mount path="/system/governance" overwrite="true">
  <instanceId>governanceRegistryInstance</instanceId>
  <targetPath>/system/governance</targetPath>
</mount>

- **mount path**: Registry collection of Carbon server instance that needs to be mounted
- **mount overwrite**: Defines if an existing collection/resource at the given path should be overwritten or not.
  Possible values are:
  - `true`: The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - `false`: The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  - `virtual`: If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.
- **target path**: Path to the remote Governance Registry instance where the registry collection is mounted.

### Configuring axis2.xml File

3. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file and enable carbon clustering by copying the following configuration to all Carbon server instances:

```xml
<clustering
  class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
  enable="true"/>
```

**Note**

The above configuration is needed only when caching is enabled in the server instances and `<enableCache>` parameter set to `true`. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.


### Configuring the Bar Product Cluster

The instructions here are similar to that of the Foo product cluster discussed above. The difference is that Bar product cluster shares G-Reg 1 (Governance space) and G-Reg 3 (Config space) remote registry spaces whereas Foo product cluster shares G-Reg 1 and G-Reg 2 (Config space).

Include the following configurations in the master node of Foo product cluster.

#### Configure master-datasources.xml File

1. Configure `$CARBON_HOME/repository/conf/datasource/master-datasources.xml` where `CARBON_HOME` is the distribution home of any WSO2 Carbon-based product. Then, add the following datasource for the registry space.

```xml
Note

The above configuration is needed only when caching is enabled in the server instances and `<enableCache>` parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.
```
Change the values of the relevant elements according to your environment.

**Configuring registry.xml File**

2. Navigate to $CARBON_HOME/repository/conf/registry.xml file and specify the following configurations.

Add a new db config to the datasource configuration done in step 1 above. For example,
Specify the remote Governance Registry instance with the following configuration:

```xml
<remoteInstance url="https://10.20.30.41:9443/registry">
  <id>governanceRegistryInstance</id>
  <dbConfig>governance_registry</dbConfig>
  <cacheId>root@https://10.20.30.41:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

<remoteInstance url="https://10.20.30.43:9443/registry">
  <id>configRegistryInstance</id>
  <dbConfig>config_registry</dbConfig>
  <cacheId>root@https://10.20.30.43:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

**Note**

When adding the corresponding configuration to the registry.xml file of a slave node, set `<readOnly>true</readOnly>`. This is the only configuration change.

Define the registry partitions using the remote Governance Registry instance.
mount path : Registry collection of Carbon server instance that needs to be mounted
mount overwrite : Defines if an existing collection/resource at the given path should be overwritten or not.
Possible vales are:
  • true - The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  • false - The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  • virtual - If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.

target path : Path to the remote Governance Registry instance where the registry collection is mounted.

Configuring axis2.xml File

3. Navigate to $CARBON_HOME/repository/conf/axis2/axis2.xml file and enable carbon clustering by copying the following configuration to all Carbon server instances:

```
<clustering
   class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
   enable="true"/>
```

Note

The above configuration is needed only when caching is enabled in the server instances and <enableCache> parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.


5. Start both clusters and note the log entries that indicate successful mounting to the remote Governance Registry nodes.

6. Navigate to the registry browser in the Carbon server's management console of a selected node and note the config and governance partitions indicating successful mounting to the remote registry instances.

Using Asymmetric Encryption

WSO2 products uses asymmetric encryption by default for the purposes of authentication and data encryption. In asymmetric encryption, keystores (with key pairs and certificates) are created and stored for the product. It is possible to have multiple keystores so that the keys used for different use cases are kept unique. The following
topics explain more details on keystores.

- Understanding keystores
- Setting up keystores for WSO2 products
- Default keystore settings in WSO2 products
- Managing keystores

Understanding keystores

A keystore is a repository that stores cryptographic keys and certificates. You use these artifacts for security purposes such as encrypting sensitive information, and establishing trust between your server and the outside parties that connect to it. The usage of keys and certificates contained in a keystore are explained below.

Key pairs: According to public-key cryptography, a key pair (private key and the corresponding public key) is used for encrypting sensitive information and for authenticating the identity of parties that communicate with your server. For example, information that is encrypted in your server using the public key can only be decrypted using the corresponding private key. Therefore, if any party wants to decrypt this encrypted data, they should have the corresponding private key, which is usually kept as a secret (not publicly shared).

Digital certificate: When there is a key pair, it is also necessary to have a digital certificate to verify the identity of the keys. Typically, the public key of a key pair is embedded in this digital certificate, which also contains additional information such as the owner, validity, etc. of the keys. For example, if an external party wants to verify the integrity of data or validate the identity of the signer (by validating the digital signature), it is necessary for them to have this digital certificate.

Trusted certificates: To establish trust, the digital certificate containing the public key should be signed by a trusted certifying authority (CA). You can generate self-signed certificates for the public key (thereby creating your own certifying authority), or you can get the certificates signed by an external CA. Both types of trusted certificates can be effectively used depending on the sensitivity of the information that is protected by the keys. When the certificate is signed by a reputed CA, all the parties who trust this CA also trust the certificates signed by them.

Identity and Trust

The key pair and the CA-signed certificates in a keystore establishes two security functions in your server: The key pair with the digital certificate is an indication of identity and the CA-signed certificate provides trust to the identity. Since the public key is used to encrypt information, the keystore containing the corresponding private key should always be protected, as it can decrypt the sensitive information. Furthermore, the privacy of the private key is important as it represents its own identity and protects the integrity of data. However, the CA-signed digital certificates should be accessible to outside parties that require to decrypt and use the information.

To facilitate this requirement, the certificates must be copied to a separate keystore (called a Truststore), which can then be shared with outside parties. Therefore, in a typical setup, you will have one keystore for identity (containing the private key) that is protected, and a separate keystore for trust (containing CA certificates) that is shared with outside parties.

Setting up keystores for WSO2 products

In WSO2 products, public key encryption is used for the following purposes:

- Authenticating the communication over Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocols.
- Encrypting sensitive information such as plain text passwords in configuration files.
- Encrypting data such as scripts, configuration files, xml, xsd etc. into the registry.

You can set up several keystores with separate key pairs and certificates for the above use cases in your system. It is recommended to maintain the following keystores:

- Maintain a primary keystore for encrypting sensitive data such as admin passwords and certain registry data. By default, the primary keystore is also used for WS-Security and for authenticating Tomcat level connections.
- Maintain a separate keystore for authenticating the communication over SSL/TLS for Tomcat level connections.
- Optionally, you can set up separate keystores with key pairs and certificates for WS-Security.
- A separate keystore (truststore) for the purpose of storing the trusted certificates of public keys in your keystores.

For information on creating new keystores with the required certificates, see Creating New Keystores, and for information on how to update configuration files in your product with keystore information, see Configuring Keystores in WSO2 Products.

Default keystore settings in WSO2 products

All WSO2 products are shipped with two default keystore files stored in the `<PRODUCT_HOME>/repository/resources/security/` directory;

- `wso2carbon.jks`: This keystore contains a key pair and is used by default in your Carbon server for all of the purposes explained above.
- `client-truststore.jks`: This is the default trust store, which contains the trusted certificates of the keystore used in SSL communication.

It is recommended to replace this default keystore with a new keystore that has self-signed or CA signed certificates when the products are deployed in production environments. This is because `wso2carbon.jks` is available with open source WSO2 products, which means anyone can have access to the private key of the default keystore.

Managing keystores

WSO2 products provide the facility to add keystores using the Management Console or using an XML configuration, and to import certificates to the keystore using the Management Console. The WSO2 keystore management feature provides a UI and an API to add and manage keystores used for WS-Security scenarios. When you apply WS-Security to Web services using the Management Console, you can select a keystore from uploaded keystores for encryption/signing processes. The Management Console also allows you to view/delete keystores.

All the functions of keystore management are exposed via APIs. As a result, if you are writing a custom extension to a WSO2 product (e.g., for WSO2 ESB mediators), you can directly access configured keystores using the API. The API hides the underlying complexity, allowing you to easily use it in third-party applications to manage their keystores as well.

This functionality is bundled with the following feature that is installed in your product.

**Name**: WSO2 Carbon - Security Management Feature  
**Identifier**: org.wso2.carbon.security.mgt.feature.group

Note the following regarding WSO2 keystore management:

- You cannot import an existing private key for which you already have a certificate.
- You cannot delete the default `wso2carbon.jks` keystore.
Configuring Keystores in WSO2 Products

After you have created a new keystore and updated the client-truststore.jks file, you must update a few configuration files in order to make the keystore work the way it is intended to. Note that keystores are used for multiple functions in WSO2 products, which includes securing the servlet transport, encrypting confidential information in configuration files, etc. Therefore, you must update the specific configuration files with the relevant keystore information. For example, you may have separate keystores for the purpose of encrypting passwords in configuration files, and for securing the servlet transport.

The wso2carbon.jks keystore file, which is shipped with all WSO2 products, is used as the default keystore for all functions. However, in a production environment, it is recommended to create new keystores with keys and certificates.

If you want an easy way to locate all the configuration files that have references to keystores, you can use the grep command as follows:

1. Open a command prompt and navigate to the <PRODUCT_HOME>/repository/conf/ directory where your product stores all configuration files.
2. Execute the following command: `grep -nr ".jks"`.

The configuration files and the keystore files referred to in each file are listed out. See an example of this below.

```
./axis2/axis2.xml:260:
  <Location>repository/resources/security/wso2carbon.jks</Location>
./axis2/axis2.xml:431:
  <Location>repository/resources/security<wso2carbon.jks</Location>
./carbon.xml:316:
  <Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./carbon.xml:332:
  <Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./identity.xml:180:
  <Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./security/secret-conf.properties:21:#keystore.identity.location=repository/resources/security/wso2carbon.jks
```

Configuring the primary keystore

The primary keystore mainly stores the keys for encrypting administrator passwords as well as other confidential information. The Keystore element in the carbon.xml file, stored in the <PRODUCT_HOME>/repository/conf/ directory should be updated with details of the primary keystore. The default configuration is shown below.

```
<keystore type="JCEKS" provider="SunJCE" keystorePassword="admin" keystorePath="repository/resources/security/wso2carbon.jks" keyPassword="admin" keyPath="repository/resources/security/wso2carbon.jks"
```
You need to add in the following information:

- `<jks store password>`
- `<jks alias>`
- `<jks store password (same as the key password)>`

**Configuring a keystore for SSL connections**

The `catalina-server.xml` file stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory should be updated with the keystore used for certifying SSL connections to Carbon servers. Given below is the default configuration in the `catalina-server.xml` file, which points to the default keystore in your product.

```xml
<keystoreFile>${carbon.home}/repository/resources/security/wso2carbon.jks</keystoreFile>
<keystorePass>Wso2carbon</keystorePass>
```

**Configuring a keystore for Java permissions**

The `sec.policy` file stored in the `<PRODUCT_HOME>/repository/conf/` directory should be updated with details of the keystore used for enabling Java permissions for your server. The default configuration is shown below.

```xml
<keystore "file:${user.dir}/repository/resources/security/wso2carbon.jks", "JKS";>
```

**Configuring keystores for WS-Security**

If there are WS-Security scenarios implemented in your WSO2 product, you can use separate keystores for these scenarios.

**Configuring keystores for advanced transport handling**
To have more advanced transport handling functions using keystores, you need to update the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file and the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file.

## Creating New Keystores

WSO2 Carbon-based products are shipped with a default keystore named `wso2carbon.jks`, which is stored in the `<PRODUCT_HOME>/repository/resources/security` directory. This keystore comes with a private/public key pair that is used to encrypt sensitive information, for communication over SSL and for encryption/signature purposes in WS-Security. However, note that since `wso2carbon.jks` is available with open source WSO2 products, anyone can have access to the private key of the default keystore. It is therefore recommended to replace this with a keystore that has self-signed or CA signed certificates when the products are deployed in production environments.

- Creating a keystore using an existing certificate
- Creating a keystore using a new certificate
- Adding the public key to client-truststore.jks

### Creating a keystore using an existing certificate

Secure Sockets Layer (SSL) is a protocol that is used to secure communication between systems. This protocol uses a public key, a private key and a random symmetric key to encrypt data. As SSL is widely used in many systems, certificates may already exist that can be reused. In such situations, you can use the CA-signed certificates to generate a Java keystore using OpenSSL and the Java keytool.

1. First you must export certificates to the **PKCS12/PFX** format. Give strong passwords whenever required.

   ```
   In WSO2 products, it is a must to have same password for both the keystore and key.
   ```

   Execute the following command to export the certificates:

   ```sh
   openssl pkcs12 -export -in <certificate file>.crt -inkey <private>.key -name "<alias>" -certfile <additional certificate file> -out <pfx keystore name>.pfx
   ```

   2. Convert the **PKCS12** to a Java keystore using the following command:

   ```
   keytool -importkeystore -srckeystore <pkcs12 file name>.pfx -srcstoretype pkcs12 -destkeystore <JKS name>.jks -deststoretype JKS
   ```

   Now you have a keystore with CA-signed certificates.

### Creating a keystore using a new certificate

If there are no certificates signed by a Certification Authority, you can follow the steps in this section to create a keystore with keys and a new certificate. We will be using the keytool that is available with your JDK installation.

#### Step 1: Creating keystore with private key and public certificate

1. Open a command prompt and go to the `<PRODUCT_HOME>/repository/resources/security/` directory. All keystores should be stored here.
2. Create the keystore that includes the private key by executing the following command:
2. Execute the following command to generate the CSR:

```
keytool -certreq -alias certalias -file newcertreq.csr -keystore newkeystore.jks
```

As mentioned before, use the same alias that you used during the keystore creation process.

You will be asked to give the keystore password. Once the password is given, the command will output the newcertreq.csr file to the `<PRODUCT_HOME>/repository/resources/security/` directory. This is the CSR that you must submit to a CA.

2. You must provide this CSR file to the CA. For testing purposes, try the 90 days trial SSL certificate from Comodo.

It is preferable to have a wildcard certificate or multiple domain certificates if you wish to have multiple subdomains like `gateway.sampledomain.org`, `publisher.sampledomain.org`, `identity.sampledomain.org`, etc., for the deployment. For such requirements you must modify the CSR request by adding subject alternative names. Most of the SSL providers give instructions to generate the CSR in such cases.
3. After accepting the request, a signed certificate is provided along with several intermediate certificates (depending on the CA) as a bundle (.zip file).

**Sample certificates provided by the CA (Comodo)**

| The Root certificate of the CA: | AddTrustExternalCARoot.crt |
| Intermediate certificates: | COMODORSAAddTrustCA.crt, COMODORSAAddTrustSecureServerCA.crt |
|SSL Certificate signed by CA: | test_sampleapp_org.crt |

**Step 3: Importing CA-signed certificates to keystore**

1. Before importing the CA-signed certificate to the keystore, you must add the root CA certificate and the two intermediate certificates by executing the commands given below. Note that the sample certificates given above are used as examples.

   ```
   keytool -import -v -trustcacerts -alias ExternalCARoot -file AddTrustExternalCARoot.crt -keystore newkeystore.jks -storepass mypassword
   keytool -import -v -trustcacerts -alias TrustCA -file COMODORSAAddTrustCA.crt -keystore newkeystore.jks -storepass mypassword
   keytool -import -v -trustcacerts -alias SecureServerCA -file COMODORSAAddTrustSecureServerCA.crt -keystore newkeystore.jks -storepass mypassword
   ```

   Optionally we can append the `-storepass <keystore password>` option to avoid having to enter the password when prompted later in the interactive mode.

2. After you add the root certificate and all other intermediate certificates, add the CA-signed SSL certificate to the keystore by executing the following command:

   ```
   keytool -import -v -alias <certalias> -file <test_sampleapp_org.crt> -keystore newkeystore.jks -keypass myppassword -storepass mykpassword
   ```

   In this command, use the same alias that you used while creating the keystore.

Now you have a Java keystore including a CA-signed certificate that can be used in a production environment. Next, you must add its public key to the `client-truststore.jks` file to enable backend communication and inter-system communication via SSL.

**Adding the public key to client-truststore.jks**

In SSL handshake, the client needs to verify the certificate presented by the server. For this purpose, the client usually stores the certificates it trusts, in a trust store. All WSO2 products are shipped with the trust store named client-truststore.jks, which resides in the same directory as the keystore (`<PRODUCT_HOME>/repository/resources/security/`). Therefore, we need to import the new public certificate into this trust store for frontend and backend communication of WSO2 products to happen properly over SSL.

Note that we are using the default client-truststore.jks file in your WSO2 product as the trust store in this
To add the public key of the signed certificate to the client trust store:

1. Get a copy of the `client-truststore.jks` file from the `<PRODUCT_HOME>/repository/resources/security/` directory.
2. Export the public key from your .jks file using the following command:

   ```bash
   keytool -export -alias certalias -keystore newkeystore.jks -file <public key name>.pem
   ```

3. Import the public key you extracted in the previous step to the `client-truststore.jks` file using the following command:

   ```bash
   keytool -import -alias certalias -file <public key name>.pem -keystore client-truststore.jks -storepass wso2carbon
   ```

   Note that ‘wso2carbon’ is the keystore password of the default client-truststore.jks file.

Now, you have an SSL certificate stored in a Java keystore and a public key added to the `client-truststore.jks` file. Note that both these files should be in the `<PRODUCT_HOME>/repository/resources/security/` directory. You can now replace the default `wso2carbon.jks` keystore in your product with the newly created keystore by updating the relevant configuration files in your product. See the related links for information.

**Related links**

- See Configuring Keystores in WSO2 Products for more information on updating the various configuration files with the new keystore information.

**Managing Keystores with the UI**

If the WSO2 Security Management feature is installed in your product, you can manage the keystores using the Management Console. In order to do this, all the required keystore files should first be created and stored in the `<PRODUCT_HOME>/repository/resources/security/` directory. For information on how to create new keystore files, see Creating New Keystores, and for information on how to update configuration files in your product with keystore information, see Configuring Keystores in WSO2 Products.

The default `wso2carbon.jks` keystore cannot be deleted.

**Adding keystores**

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to manage multiple keystores. Follow the instructions below to add a new keystore to your product using the Management Console.

1. Log in to the WSO2 product with your user name and password.
2. Go to the **Configure** tab and click **Key Stores**.
3. The **Key Store Management** page appears. Click the **Add New Key store** link to open the following screen:

   Home > Configure > KeyStores > Add New KeyStore

## Add New KeyStore

### Step 1: Upload KeyStore File

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyStore File*</td>
<td>Choose File no file selected</td>
</tr>
<tr>
<td>KeyStore Password*</td>
<td>******</td>
</tr>
<tr>
<td>Provider</td>
<td>admin</td>
</tr>
<tr>
<td>KeyStore Type</td>
<td>JKS</td>
</tr>
</tbody>
</table>

Next > Cancel

4. Specify the **Provider** and the **Keystore Password**, which points to the password required to access the private key.
5. In the **Keystore Type** field, specify whether the keystore file you are uploading is JKS or PKCS12.
   - **JKS** (Java Key Store): Allows you to read and store key entries and certificate entries. However, the key entries can store only private keys.
   - **PKCS12** (Public Key Cryptography Standards): Allows you to read a keystore in this format and export the information from that keystore. However, you cannot modify the keystore. This is used to import certificates from different browsers into your Java Key store.
6. Click **Next** and on the next page, provide the **Private Key Password**.
7. Click **Finish** to add the new keystore to the list.

### Viewing keystores

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to view keystores using the Management Console. Follow the instructions below to view a keystore.

1. Log in to the WSO2 product with your user name and password.
2. Go to the **Configure** tab and click **Key Stores**.
3. The **Key Store Management** page appears. All the keystores that are currently added to the product will be listed here as follows:

   Home > Configure > KeyStores

## KeyStore Management

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>wso2carbon.jks</td>
<td>JKS</td>
<td>![Import Cert] ![View] ![Delete]</td>
</tr>
</tbody>
</table>

Add New KeyStore
4. Click View in the list of actions. The View Key Store screen shows information about the available certificates.

It also displays information about private key certificates:

Certificate of the Private Key

5. Click Finish to go back to the Key Store Management screen.

Importing certificates to keystore

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to import certificates for keystores. Follow the instructions below to import a certificate for a keystore.

1. Log in to the WSO2 product with your user name and password.
2. Go to the Configure tab and click Keystores.
3. The Keystore Management page appears. All the keystores that are currently added to the product will be listed here as follows:
4. Click Import Cert associated with the keystore for which you want to import a certificate.
5. The available certificates are already listed on the Import Certificates screen. Click Browse to find the location of the new certificate that you want to import.
6. Once you have selected the certificate, click Import.
7. Once a certificate is imported successfully, you will see the following confirmation:
Click **OK**.

8. The imported certificate appears in the list of **Available Certificates**. In the example shown below, the "GeoTrust_Global_CA" certificate was imported.

```
mycert
verisignclass3ca
entrustgsslca
```

Click **OK**.

8. The imported certificate appears in the list of **Available Certificates**. In the example shown below, the "GeoTrust_Global_CA" certificate was imported.

```
mycert
verisignclass3ca
entrustgsslca
```

## Fixing Security Vulnerabilities

A cipher is an algorithm for performing encryption or decryption. You can disable the weak ciphers in the Tomcat server by modifying the `cipher` attribute in the SSL Connector container, which is in the `catalina-server.xml` file. Enter the ciphers that you want your server to support in a comma-separated list. By default, all ciphers, whether they are strong or weak, will be enabled. However, if you do not add the `cipher` attribute or keep it blank, all SSL ciphers by JSSE will be supported by your server. This will enable the weak ciphers.

The steps below explain how to disable weak and enable strong ciphers in a product:

1. Take a backup of `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.
2. Stop the server.
3. Add the `cipher` attribute to the existing configuration in the `catalina-server.xml` file with the list of ciphers that you want your server to support as follows:

   ```xml
   ciphers="<cipher-name>,<cipher-name>"
   ```

   The example below shows how a connector looks after an example configuration is done:
    port="9443"
    bindOnInit="false"
    sslProtocol="TLS"
    maxHttpHeaderSize="8192"
    acceptorThreadCount="2"
    maxThreads="250"
    minSpareThreads="50"
    disableUploadTimeout="false"
    enableLookups="false"
    connectionUploadTimeout="120000"
    maxKeepAliveRequests="200"
    acceptCount="200"
    server="WSO2 Carbon Server"
    clientAuth="false"
    compression="on"
    scheme="https"
    secure="true"
    SSLEnabled="true"
    compressionMinSize="2048"
    noCompressionUserAgents="gozilla, traviata"
    compressableMimeTypes="text/html, text/javascript, application/xhtml+xml, text/css, application/xml, text/xsl, image/gif, image/jpeg"
    ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLSDHE_RSA_WITH_AES_128_CBC_SHA,TLSDHE_DSS_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA"
    keystoreFile="${carbon.home}/repository/resources/security/wso2carbon.jks"
    keystorePass="wso2carbon"
    URIEncoding="UTF-8"/>

4. Save the catalina-server.xml file.

5. Restart the server.

Logging

Logging is one of the most important aspects of a production-grade server. A properly configured logging system is vital in identifying errors, security threats and usage patterns. You can view system and application logs of a running WSO2 product instance in different ways as follows:
• Through the Management Console.
• Through the log files that are stored in `<PRODUCT_HOME>/repository/logs` folder. The folder contains current logs in a log file with a date stamp. Older logs are archived in `wso2carbon.log` file.
• Through the command prompt/shell terminal that opens when running the `wso2server.bat/wso2server.sh` files to start the server.

WSO2 products use a log4j-based logging mechanism through Apache Commons Logging facade library. The `log4j.properties` file, which governs how logging is performed by the server is in `<PRODUCT_HOME>/repository/conf` folder. There are two ways to configuring log4j:

• Manually editing the `log4j.properties` file
• **Logging Configuration** through the management console. Changes apply at run time.

We recommend the second approach because you do not have to restart the server for the configuration changes to apply. When you change the parameters using the Management Console, first, the server stores new values in the database and then changes the appropriate components in the logging framework, enabling logging properties to be updated immediately. All changes made to Log4j through the management console are persisted in the WSO2 Registry and are available after server restarts. Any changes to the logging configuration you make through the management console get priority over `log4j.properties` file settings. However, if you modify `log4j.properties` and restart the server, the earlier log4j configuration that persisted in the registry will be overwritten. There is also an option in the management console to restore the original Log4j configuration from the `log4j.properties` file.

WSO2 products store logs per service. You cannot drill down service-level logs further to filter operational or query logs. We also do not provide database level logs. However, if you get SQL errors (e.g., SQL violations in your queries), you can see those errors in ERROR logs. You can also use application logs to in case of an issue to figure out the cause.

**Logging configuration**

There are three main components in log4j as Loggers, Appenders, and Layouts. You can change these parameters both globally and individually, at run time.

Follow the steps below to configure logging properties using the management console.

1. Log in to the product's management console and select **Configure > Logging**.
2. The **Logging Configuration** page appears as follows:
It has the following configuration options:

**Persist All Configuration Changes**: Allows you to persist all modifications, which will be available even after the server restarts.

**Global Log4J Configuration**: This section allows you to assign a single log level and log pattern to all loggers.

- **Log Level**: Severity of the message. Reflects a minimum level for the logger. You can view the hierarchy of levels.
- **Log Pattern**: Defines the output format of the log file. This is the layout pattern that describes the log message format.

The Restore Defaults button allows you to overwrite the Registry with the logging configurations specified in the `log4j.properties` file.

**Configure Log4J appenders**: This section allows you to configure appenders individually. Log4j allows logging requests to print to multiple destinations. These output destinations are called Appenders. You can attach several appenders to one logger.

- **Name**: The name of an appender. Following log appenders are configured by default:
  - CARBON_CONSOLE - Logs to the console when the server is running.
  - CARBON_LOGFILE - Writes the logs to AS_HOME/repository/logs/wso2carbon.log.
  - CARBON_MEMORY
  - CARBON_SYS_LOG - Allows separation of the software that generates messages from the system that stores them and the software that reports and analyzes them.
  - CARBON_TRACE_LOGFILE

- **Log pattern**: Defines the output format of the log file.
- **Sys Log Host**: The IP address of the system log server. The syslog server is a dedicated log server for many applications. It runs in a particular TCP port in a separate machine, which can be identified by an IP address.
- Facility - The log message type sent to the system log server.
- Threshold - Filters log entries based on their level. For example, threshold set to "WARN" will allow log entry to pass into appender if its level is "WARN," "ERROR" or "FATAL," other entries will be discarded. This is the minimum log level at which you can log a message.

The available categories of logs you can view are:

- TRACE - Designates fine-grained informational events than the DEBUG.
- DEBUG - Designates fine-grained informational events that are most useful to debug an application.
- INFO - Designates informational messages that highlight the progress of the application at coarse-grained level.
- WARN - Designates potentially harmful situations.
- ERROR - Designates error events that might still allow the application to continue running.
- FATAL - Designates very severe error events that will presumably lead the application to abort.

Configure Log4J Loggers: A Logger is an object used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace and have a "child-parent" relationship. For example, the logger named "root.sv" is a parent of the logger named "root.sv.sf" and a child of "root."

When the server starts for the first time, all the loggers initially listed in the log4j.properties file appear on the logger name list. This section allows you to browse through all these loggers, define a log level and switch on/off additivity to any of them. After editing, the logging properties are read only from the database.

- Logger - The name of a logger.
- Parent Logger - The name of a parent logger.
- Level - Allows to select level (threshold) from the drop-down menu. After you specify the level for a certain logger, a log request for that logger will only be enabled if its level is equal or higher to the logger's one. If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. Refer to hierarchy of levels above.
- Additivity - Allows to inherit all the appenders of the parent Logger if set as True.

In this section, loggers can be filtered by the first characters (use the Starts With button) or by a combination of characters (use the Contains button).

Enabling data service DEBUG logs

To receive data service level debug logs, set the following list of loggers to DEBUG using either the UI or the log4j.properties file. Their logger level is set to INFO by default. Changing through the UI does not require a server restart.

- org.wso2.carbon.dataservices.core.DBDeployer
- org.wso2.carbon.dataservices.core.DBinOnlyMessageReceiver
- org.wso2.carbon.dataservices.core.DBinOutMessageReceiver
- org.wso2.carbon.dataservices.core.DBUtils
- org.wso2.carbon.dataservices.core.admin.DataServiceAdmin
- org.wso2.carbon.dataservices.core.admin.DataServiceFileUploader
- org.wso2.carbon.dataservices.core.custom.datasource.AbstractCustomDataSourceReader
- org.wso2.carbon.dataservices.core.custom.datasource.EchoDataSource
- org.wso2.carbon.dataservices.core.description.config.SQLConfig
- org.wso2.carbon.dataservices.core.description.query.SQLQuery
- org.wso2.carbon.dataservices.core.engine.DataService
- org.wso2.carbon.dataservices.core.engine.ParamValue
- org.wso2.carbon.dataservices.core.internal.DSAxis2ConfigurationContextObserver
- org.wso2.carbon.dataservices.core.internal.DataServicesDSComponent
- org.wso2.carbon.dataservices.task.DSTaskAdmin
If you use the UI, in the Logging Configuration page, search for loggers that contain the name dataservice and turn their level to DEBUG. For example,

![Configure Log4J Loggers](image)

**Feature Management**

This section contains the following information:

- Introduction to Feature Management
- Installing and Managing Features
- Recovering from Unsuccessful Feature Installation

For more information on installing features of any WSO2 component to WSO2 CEP to extend its functionality, see the Feature Management section of the WSO2 Carbon Documentation.

**Introduction to Feature Management**

Each WSO2 product is a collection of reusable software units called features where a single feature is a list of components and/or other feature. A component in WSO2 products is a single or a collection of OSGi bundles. Similar to a standard JAR file in Java, a bundle is the modularization unit in OSGi. This component-based architecture of WSO2 gives developers flexibility to build efficient and lean products that best suit their unique business needs, simply by adding and removing components.

Components add functionality to the products. For example, the statistics component enables users to monitor system and service-level statistics. This component contains two bundles. One is the back-end bundle that collects, summarizes and stores statistics. The other is the front-end bundle that presents data to the user through a user-friendly interface.

**What is software provisioning**

Provisioning software is the act of placing an individual software application or a complete software stack onto a target system. What we mean by provisioning WSO2 products is installing/updating/uninstalling features to/from WSO2 Carbon, which is the base platform on top of which the entire WSO2 product stack is developed. It is also possible to revert to a previous feature configuration using provisioning support.

You can easily install features to any WSO2 product using the WSO2 Component Manager, which comes with the products. Component manager is powered by Equinox P2 and allows you to connect to a remote or local P2 repository and get any feature installed into the product’s runtime. P2 can be used as a provisioning platform for any OSGi-based application. It enables easy provisioning capabilities and increases the user-friendliness in building
customized SOA products using the Carbon platform. Users can download the WSO2 Carbon platform or any other WSO2 product and extend their functionality by simply installing features. WSO2 Feature Manager provides a convenient user interface to perform common provisioning operations and related repository management functions.

You can also manually provision Carbon by dropping bundles and configuration files that belong to a feature. This method is not recommended because if you do not find the exact set of components and dependencies, it can lead to issues. Features/components can have many dependencies with other features/components and some even depend on specific versions of other components. Therefore, we recommend you to use WSO2 component manager as explain in the next section.

**Installing and Managing Features**

As explained in the Introduction, the recommended way to install features is using the component manager. You can also manually provision Carbon by dropping bundles and configuration files that belong to a feature. This method is not recommended because it is complex and error prone. WSO2 has Equinox P2 integrated with its products. It enables user-friendly provisioning capabilities using the component manager explained below.

If you are on Windows, be sure to point the `-Dcarbon.home` property in the product's startup script (`wso2server.bat`) to the product's distribution home (e.g., `-Dcarbon.home=C:\Users\VM\Desktop\wso2ce\p-4.0.0`). Then, restart the server. If not, you might not be able to install features through the management console.

The steps below explain how to **add a feature repository**, **disable a repository**, **install features** from the repository and **turn your server to an exclusive back-end/front-end server**.

1. Log in to the management console and select **Features** from the **Configure** menu.
2. The **Feature Management** page opens.

**Adding a feature repository**

3. First step is to add a feature repository. If you already have one, skip to **Installing a feature**. Else, go to the **Repository Management** tab and click **Add Repository**.
4. Provide a name and repository location and click **Add**. For example,

   ![Add Repository](image)

   WSO2 features are available in the Equinox P2 repository, which you can access from the **Release Matrix page** on the WSO2 Website (see the Links column for the platform release corresponding to the product version you are running).

   Feature manager is unable to add a remote repository when it is **behind a proxy**. In that case, download the remote repository to your environment and add it by selecting the **local** option.
5. After adding, you can change the repository name using the **Edit** link associated with it.

   You cannot change the repository URL after adding it. To change the URL, you must remove the old repository and add a new one.

6. By default, all repositories are enabled. You can disable a repository using the **Disable** link associated with it.

   ![Manage Repositories](image)

   When you perform a provisioning operation, metadata and artifacts are searched only from the enabled repositories.

**Installing features**

7. In the **Feature Management** page, click **Available Features** tab. Then, select a repository from the drop-down menu.

   ![Feature Management](image)

   The following options can be selected.

   **Show only the latest versions**

   Some repositories contain multiple versions of features. If you are only interested in the latest versions, click the **Show only the latest versions** option.

   **Group features by category**

   A feature category is a logical grouping of the features that constitute a particular WSO2 product. Categorizing logically related features makes it easier for users to search and install related features together. You can select the entire list of features of a particular product at once. Under these product based feature categories, there are other feature categories based on the product features. If you un-check this option when finding features, you will see an uncategorized, flat feature list from which individual features can be selected.
separately. For example, the features required to install WSO2 Data Services Server is grouped under the Data Service Server feature category as shown below.

8. Once the repository and options are selected, click the Find Features button.

To find a particular feature, you can use the search box. Search only returns available, uninstalled features. It excludes the ones that are already installed.

9. From the list of features that appear, select the ones you want to add and click Install.

10. The Install Details page appears. Verify the provided information and click Next.

11. Read and accept the terms of license agreement.

12. The installation process starts. It may take a few minutes to download the necessary components.

13. Once the installation process is complete, click Finish and restart the server for the changes to take effect.

14. Go to the Installed Features tab to browse through the list of installed features.

Turning your product to a back-end/front-end server

15. WSO2 products support back-end, front-end separation where you can manage multiple back-end servers using a single front-end server. You can convert a given product either to a back-end server or to a front-end server by removing the irrelevant features.

For example, if you want to get only a back-end server, you have to uninstall all the front-end features. To do that, select Front-end from the drop down menu as follows:
This lists all the front-end features that are currently installed in the system.

16. Select the features you want to remove and click **Uninstall**.

Unsuccessful feature installation can cause server startup failures. See [Recovering from Unsuccessful Feature Installation](#).

## Recovering from Unsuccessful Feature Installation

After installing features, if you encounter server issues or startup failures, you can revert the current configuration by restoring a previous one using either the management console or the command line. The latter is recommended if you cannot start the server.

Use the following steps to check your feature installation history and revert the server back to a previous installation. In this recovery process, some features might get installed and some uninstalled.

- Restoring using the management console
- Restoring using the command line

### Restoring using the management console

1. Log in to the management console and select **Features** from the **Configure** menu.
2. In the **Feature Management** page, go to the **Installation History** tab.
3. This tab lists the history of provisioning operations performed on the system. For example,

   ![Feature Management](image)

   ![Installation History](image)

4. Click on a configuration to view its details. For example,
Previous configurations can be identified as previous states of the system. It is a set of installed features. When you perform a provisioning operation such as installing/uninstalling of features, a system state/configuration change occurs.

5. Verify if the state is where you want to revert to and click **Revert**.

**Restoring using the command line**

If you cannot start the server after an unsuccessful feature installation, use the following steps to restore to a previous installation.

1. Start the product with `-DosgiConsole` system property.
2. Once the server is started, type the command `osgi> getInstallationHistory`.
3. A list of previous server states appears. For example,

   
   1376883697814 August 19, 2013 at 09:11:37 IST  
   1376883697957 August 19, 2013 at 09:11:37 IST  
   1376883700725 August 19, 2013 at 09:11:40 IST  
   1376883704884 August 19, 2013 at 09:11:44 IST  
   ...

4. You can check what features are installed and uninstalled in a given state by entering the following command:

   `osgi> getInstallationHistory <timestamp>`

   **Example:**
   
   `osgi> getInstallationHistory 1376933879416`

   The output gives you details similar to the following:
WSO2 Complex Event Processor, version 3.1.0

-- Installed features in this configuration

-- Uninstalled features in this configuration
WSO2 Carbon - Service Management Feature 4.2.0
WSO2 Stratos - Deployment Features 2.2.0
WSO2 Stratos - Common Composite Feature 2.2.0
WSO2 Stratos - Usage Agent Feature 2.2.0
WSO2 Stratos - Throttling Agent Feature 2.2.0
...

5. Decide to which state you want to revert the system and enter the following command:

```
.osgi> revert <timestamp>
```

For example:

```
.osgi> revert 1376933879416
```

The output will be similar to the following:

Successfully reverted to 1376933879416
Changes will get applied once you restart the server.

Transports

This section provides the following information:

- Introduction to Transports
- Server Transports
- Configuring Transports Globally
- Configuring Transport Level Security

Introduction to Transports

WSO2 Carbon is the base platform on which all WSO2 products are developed. Built on OSGi, WSO2 Carbon encapsulates all major SOA functionality. It supports a variety of transports, which make WSO2 products capable of receiving and sending messages over a multitude of transport and application-level protocols. This functionality is implemented mainly in the Carbon core, which combines a set of transport-specific components to load, enable, manage and persist transport related functionality and configurations.

All transports currently supported by WSO2 Carbon are directly or indirectly based on the Apache Axis2 transports framework. This framework provides two main interfaces for each transport implementation.

- **org.apache.axis2.transport.TransportListener** - Implementations of this interface must specify how incoming messages are received and processed before handing them over to the Axis2 engine for further processing.
- **org.apache.axis2.transport.TransportSender** - Implementations of this interface should specify how a message can be sent out from the Axis2 engine.

Each transport implementation generally contains a transport receiver/listener and a transport sender, since they use the interfaces above. The Axis2 transport framework enables the user to configure, enable and manage transport listeners and senders independent to each other, without having to restart the server. For example, one may enable only the JMS transport sender without having to enable JMS transport listener.
The transport management capability of WSO2 Carbon is provided by the following feature in the WSO2 feature repository:

**Name**: WSO2 Carbon - Transport Management Feature  
**Identifier**: org.wso2.carbon.transport.mgt.feature.group

If transport management capability is not included in your product by default, you can add it by installing the above feature using the instructions given in section Feature Management.

### Server Transports

WSO2 CEP supports the following transports, which make it capable of receiving and sending messages over a multitude of transport and application protocols.

- HTTP Servlet Transport
- HTTPS Servlet Transport
- Local Transport
- TCP Transport

#### HTTP Servlet Transport

The transport receiver implementation of the HTTP transport is available in the Carbon core component. The transport sender implementation comes from the Apache Axis2 transport module. This transport is shipped with WSO2 Carbon and all WSO2 Carbon-based products, which use this transport as the default transport, except WSO2 ESB. The two classes which implement the listener and sender APIs are `org.wso2.carbon.core.transports.http.HttpTransportListener` and `org.apache.axis2.transport.http.CommonsHTTPTransportSender` respectively.

- This is a blocking HTTP transport implementation, meaning that I/O threads get blocked while received messages are processed completely by the underlying Axis2 engine.
- In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

### Transport receiver parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The port number on which this transport receiver should listen for incoming messages.</td>
<td>Yes</td>
<td>A positive integer less than 65535</td>
<td></td>
</tr>
</tbody>
</table>
proxyPort

When used, this transport listener will accept messages arriving through a HTTP proxy server which listens on the specified proxy port. Apache mod_proxy should be enabled in the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.

No A positive integer less than 65535

HTTP servlet transport should be configured in the $PRODUCT_HOME/repository/conf/tomcat/catalina-server.xml file. The transport class that should be specified in the catalina-server.xml file is as follows:

```xml
```

This servlet transport implementation can be further tuned up using the following parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The port over which this transport receiver listens for incoming messages.</td>
<td>Yes</td>
<td>A positive integer less than 65535</td>
<td></td>
</tr>
<tr>
<td>proxyPort</td>
<td>When used, this transport listener will accept messages arriving through a HTTP proxy server which listens on the specified proxy port. Apache mod_proxy should be enabled on the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.</td>
<td>No</td>
<td>A positive integer less than 65535</td>
<td></td>
</tr>
<tr>
<td>maxHttpHeaderSize</td>
<td>The maximum size of the HTTP request and response header in bytes.</td>
<td>No</td>
<td>A positive integer</td>
<td>4096</td>
</tr>
<tr>
<td>maxThreads</td>
<td>The maximum number of worker threads created by the receiver to handle incoming requests. This parameter largely determines the number of concurrent connections that can be handled by the transport.</td>
<td>No</td>
<td>A positive integer</td>
<td>40</td>
</tr>
<tr>
<td>enableLookups</td>
<td>Use this parameter to enable DNS lookups in order to return the actual host name of the remote client. Disabling DNS lookups at transport level generally improves performance.</td>
<td>No</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>parameter</td>
<td>description</td>
<td>required</td>
<td>default</td>
<td>allowed values</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>disableUploadTimeout</td>
<td>This flag allows the servlet container to use a different, longer connection timeout while a servlet is being executed, which in the end allows either the servlet a longer amount of time to complete its execution, or a longer timeout during data upload.</td>
<td>No</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>clientAuth</td>
<td>Set to true if you want the SSL stack to require a valid certificate chain from the client before accepting a connection. Set to want if you want the SSL stack to request a client Certificate, but not fail if one is not present. A false value (which is the default) will not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.</td>
<td>No</td>
<td>true, false, want</td>
<td>false</td>
</tr>
<tr>
<td>maxKeepAliveRequests</td>
<td>The maximum number of HTTP requests which can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting this to -1 will allow an unlimited amount of pipelined or keep-alive HTTP requests.</td>
<td>No</td>
<td>-1 or any positive integer</td>
<td>100</td>
</tr>
<tr>
<td>acceptCount</td>
<td>The maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full will be refused.</td>
<td>No</td>
<td>A positive integer</td>
<td>10</td>
</tr>
<tr>
<td>compression</td>
<td>Use this parameter to enable content compression and save server bandwidth.</td>
<td>No</td>
<td>on, off, force</td>
<td>off</td>
</tr>
<tr>
<td>noCompressionUserAgents</td>
<td>Indicate a list of regular expressions matching user-agents of HTTP clients for which compression should not be used, because these clients, although they do advertise support for the feature, have a broken implementation.</td>
<td>No</td>
<td>A comma separated list of regular expressions</td>
<td>empty string</td>
</tr>
</tbody>
</table>
compressableMimeType | Use this parameter to indicate a list of MIME types for which HTTP compression may be used. | No | A comma separated list of valid mime types | text/html,text/xml,text/plain

This is only a subset of all the supported parameters. The servlet HTTP transport uses the `org.apache.catalina.connector.Connector` implementation from Apache Tomcat. So the servlet HTTP transport actually accepts any parameter accepted by the connector implementation. Please refer to Apache Tomcat's connector configuration reference (http://tomcat.apache.org/tomcat-5.5-doc/config/http.html) for more information and a complete list of supported parameters.

**Transport sender parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROTOCOL</td>
<td>The version of HTTP protocol to be used for outgoing messages.</td>
<td>No</td>
<td>HTTP/1.0, HTTP/1.1</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>Transfer-Encoding</td>
<td>Effective only when the HTTP version is 1.1 (i.e. the value of the PROTOCOL parameter should be HTTP/1.1). Use this parameter to enable chunking support for the transport sender.</td>
<td>No</td>
<td>chunked</td>
<td>Not Chunked</td>
</tr>
<tr>
<td>SocketTimeout</td>
<td>The socket timeout value in milliseconds, for out bound connections.</td>
<td>No</td>
<td>A positive integer</td>
<td>60000 ms</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>The connection timeout value in milliseconds, for out bound connections.</td>
<td>No</td>
<td>A positive integer</td>
<td>600000 ms</td>
</tr>
<tr>
<td>OmitSOAP12Action</td>
<td>Set this parameter to “true” if you need to disable the soapaction for SOAP 1.2 messages.</td>
<td>No</td>
<td>true, false</td>
<td>false</td>
</tr>
</tbody>
</table>

**HTTPS Servlet Transport**

Similar to the HTTP transport, the HTTPS transport consists of a receiver implementation which comes from the Carbon core component and a sender implementation which comes from the Apache Axis2 transport module. In fact, this transport uses exactly the same transport sender implementation as the HTTP transport. So the two classes that should be specified in the configuration are `org.wso2.carbon.core.transports.http.HttpsTransportListener` and `org.apache.axis2.transport.http.CommonsHTTPTransportSender` for the receiver and sender in the specified order. The configuration parameters associated with the receiver and the sender are the same as in HTTP transport. This is also a blocking transport implementation.

However, when using the following class as the receiver implementation, we need to specify the servlet HTTPS transport configuration in the transport's XML file.

- `org.wso2.carbon.core.transports.http.HttpsTransportListener`

The class that should be specified as the transport implementation is `org.wso2.carbon.server.transports.http.HttpsTransport`. In addition to the configuration parameters supported by the HTTP servlet transport, HTTPS servlet transport supports the following configuration parameters:
**Note:** In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslProtocol</td>
<td>Transport level security protocol to be used.</td>
<td>No</td>
<td>TLS, SSL</td>
<td>TLS</td>
</tr>
<tr>
<td>keystore</td>
<td>Path to the keystore which should be used for encryption/decryption.</td>
<td>Yes</td>
<td>A valid file path to a keystore file</td>
<td></td>
</tr>
<tr>
<td>keypass</td>
<td>Password to access the specified keystore.</td>
<td>Yes</td>
<td>A valid password</td>
<td></td>
</tr>
</tbody>
</table>

Similar to the servlet HTTP transport, this transport is also based on Apache Tomcat's connector implementation. Please refer Tomcat connector configuration reference for a complete list of supported parameters.

**Local Transport**

Apache Axis2's local transport implementation is used to make fast, in-VM service calls and transfer data within proxy services. The transport does not have a receiver implementation. The following class implements the sender API:

- `org.apache.axis2.transport.local.NonBlockingLocalTransportSender`

To use this transport, configure an endpoint with the `local://` prefix. For example, to make an in-VM call to the `HelloService`, use `local://HelloService`. Note that the local transport cannot be used to send REST API calls, which require the HTTP/S transports.

**Configuring the Local Transport**

By default, WSO2 provides `CarbonLocalTransportSender` and `CarbonLocalTransportReceiver`, which are used for internal communication among Carbon components and are not suitable for service invocation. To enable the local transport for service invocation, follow these steps.

1. In the `carbon.xml` file at location `<PRODUCT_HOME>/repository/conf`, an endpoint is available as follows by default.

   ```xml
   <ServerURL>local://services/</ServerURL>
   ``

   Replace it with

   ```xml
   <ServerURL>https://${carbon.local.ip}:${carbon.management.port}${carbon.context}/services/</ServerURL>
   ``

2. In the `axis2.xml` file at location `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml`, there is a transport sender and receiver named 'local' specified as follows in two different places:
TCP Transport

The TCP transport allows you to send and receive SOAP messages over TCP. The TCP transport is included with the WSO2 product distribution but must be enabled before use. To enable the TCP transport, open the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file in a text editor and add the following transport receiver configuration and sender configuration:

```xml
<transportReceiver name="tcp"
    class="org.apache.axis2.transport.tcp.TCPTransportListener">
    <parameter name="transport.tcp.port">6060</parameter>
</transportReceiver>
<transportSender name="tcp"
    class="org.apache.axis2.transport.tcp.TCPTransportSender"/>
```

If you want to use the sample Axis2 client to send TCP messages, uncomment the TCP transport sender configuration in the following file:

`samples/axis2Client/client_repo/conf/axis2.xml`

### Transport receiver parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The port on which the TCP server should listen for incoming messages</td>
<td>No</td>
<td>A positive integer less than 65535</td>
<td>8000</td>
</tr>
<tr>
<td>hostname</td>
<td>The host name of the server to be displayed in WSDLs, etc.</td>
<td>No</td>
<td>A valid host name or an IP address</td>
<td></td>
</tr>
</tbody>
</table>

### Configuring Transports Globally

You can configure and enable transports in a service level or in a global level using either of the following methods.

Remove both these lines and add following line.

```xml
<transportSender name="local"
    class="org.apache.axis2.transport.local.NonBlockingLocalTransportSender"/>
```
Globally enabled and configured transports effect all services deployed in a running WSO2 product instance.

- Using the axis2.xml file
- Using catalina-server.xml file

**Using the axis2.xml file**

WSO2 products come with a configuration file named axis2.xml in `<PRODUCT_HOME>/repository/conf/axis2` directory. This is similar to the axis2.xml file that comes with Apache Axis2 and Apache Synapse. It contains the global configuration of WSO2 products. The axis2.xml configuration generally includes configuration details for modules, phases, handlers, global configuration parameters and transports. The elements `<transportReceiver>` and `<transportSender>` are used to configure transport listeners and senders respectively. In the axis2.xml file that comes with WSO2 products, some transports are already configured and enabled by default, including the HTTP and HTTPS transports.

WSO2 products do not use the HTTP/S servlet transport configurations that are in axis2.xml file. Instead, they use Tomcat-level servlet transports, which are used by the management console in `<PRODUCT_HOME>/repository/conf/tomcat` file.

Given below is an example JMS transport receiver configuration in the axis2.xml file.
<transportReceiver name="jms"
  class="org.apache.axis2.transport.jms.JMSListener">
  <parameter name="myTopicConnectionFactory">
    <parameter name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url">tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</parameter>
  </parameter>
  <parameter name="myQueueConnectionFactory">
    <parameter name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url">tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName">QueueConnectionFactory</parameter>
  </parameter>
  <parameter name="default">
    <parameter name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url">tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName">QueueConnectionFactory</parameter>
  </parameter>
</transportReceiver>

<transportSender>
  element has the following attributes and sub elements:

  - **name** - A mandatory attribute which indicates a unique name for the transport receiver.
  - **class** - A mandatory attribute which indicates the transport receiver implementation class.
  - **parameters** - Configuration parameters for the transport receiver. It should be included as child elements of the <transportReceiver> element.

Similarly use <transportSender> element to configure and enable transport senders in WSO2 products.

- The axis2.xml file is loaded to memory only during server startup. Therefore, you must restart the server to apply any changes you make to the file while the server is up and running.
- Simply having <transportReceiver> and <transportSender> elements in the axis2.xml file causes those transports to be loaded and activated during server startup. Therefore, you must include any dependency JARs required by those transport implementations in the server classpath to prevent the server from running into exceptions at startup. In addition to that, an inaccurate transport
configuration (for example, a wrong parameter value) might cause the transport to be not enabled properly.

Using catalina-server.xml file

In addition to the above, transport receivers can be configured globally using the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file. The default HTTP/S configuration specified in the catalina-server.xml file is given below:

```
<!-- optional attributes:
 proxyPort="80" -->

            port="9763"
            bindOnInit="false"
            maxHttpHeaderSize="8192"
            acceptorThreadCount="2"
            maxThreads="250"
            minSpareThreads="50"
            disableUploadTimeout="false"
            connectionUploadTimeout="120000"
            maxKeepAliveRequests="200"
            acceptCount="200"
            server="WSO2 Carbon Server"
            compression="on"
            compressionMinSize="2048"
            noCompressionUserAgents="gozilla, traviata"
            compressableMimeType="text/html, text/javascript, application/x-javascript, application/javascript, application/xml, text/css, application/xslt+xml, text/xsl, image/gif, image/jpg, image/jpeg" URIEncoding="UTF-8" />

<!-- optional attributes:proxyPort="443" -->

            port="9443"
            bindOnInit="false"
            sslProtocol="TLS"
            maxHttpHeaderSize="8192"
            acceptorThreadCount="2"
            maxThreads="250"
            minSpareThreads="50"
            disableUploadTimeout="false"
            enableLookups="false"
            connectionUploadTimeout="120000"
            maxKeepAliveRequests="200"
            acceptCount="200"
            server="WSO2 Carbon Server"
            clientAuth="false"
            URIEncoding="UTF-8" />
```
compression="on"
scheme="https"
secure="true"
SSLEnabled="true"
compressionMinSize="2048"
noCompressionUserAgents="gozilla, traviata"
compressableMimeType="text/html, text/javascript, application/x-javascript, application/javascript, application/xml, text/css, application/xslt+xml, text/xsl, image/gif, image/jpg, image/jpeg"
At the moment, you can configure only the default servlet transports of WSO2 using `catalina-server.xml` file.

For more details on config parameters, see [http://tomcat.apache.org/tomcat-7.0-doc/config/http.html](http://tomcat.apache.org/tomcat-7.0-doc/config/http.html).

### Configuring Transport Level Security

The transport level security protocol of the Tomcat server is configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file. Note that the `sslProtocol` attribute is set to TLS (Transport Layer Security) by default.

See the following topics for configuration:

- Disabling SSL version 3
- Disabling the weak ciphers

#### Disabling SSL version 3

It is necessary to disable SSL version 3 in WSO2 products because of a bug ([Poodle Attack](http://tomcat.apache.org/tomcat-7.0-doc/config/http.html)) in the SSL version 3 protocol that could expose critical data encrypted between clients and servers. The Poodle Attack makes the system vulnerable by telling the client that the server does not support the more secure TLS protocol. This forces the server to connect via SSL 3.0. You can mitigate the effect of this bug by disabling SSL version 3 protocol in your server.

Follow the steps below to disable SSL 3.0 support.

1. Make a backup of the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file and stop the server.
2. Find the connector configuration that is corresponding to TLS (usually, this connector has the port set to 9443 and the `sslProtocol` as TLS).
   - If you are using JDK 1.6, remove the `sslProtocol="TLS"` attribute from the configuration and replace it with `sslEnabledProtocols="TLSv1"` as shown below.

```xml
    port="9443"
    bindOnInit="false"
    sslEnabledProtocols="TLSv1"
>
```

   - If you are using JDK 1.7, remove the `sslProtocol="TLS"` attribute from the above configuration and replace it with `sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"` as shown below.

```xml
    port="9443"
    bindOnInit="false"
    sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"
>
3. Start the server.

To test if SSL version 3 is disabled:

1. Download TestSSLServer.jar from here.
2. Execute the following command to test the transport:

   ```
   java -jar TestSSLServer.jar localhost 9443
   ```

3. The output of the command before and after disabling SSL version 3 is shown below.

**Before SSL version 3 is disabled:**

```plaintext
Supported versions: SSLv3 TLSv1.0
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
SSLv3
   RSA_EXPORT_WITH_RC4_40_MD5
   RSA_WITH_RC4_128_MD5
   RSA_WITH_RC4_128_SHA
   RSA_EXPORT_WITH_DES40_CBC_SHA
   RSA_WITH_DES_CBC_SHA
   RSA_WITH_3DES_EDE_CBC_SHA
   DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
   DHE_RSA_WITH_DES_CBC_SHA
   DHE_RSA_WITH_3DES_EDE_CBC_SHA
   RSA_WITH_AES_128_CBC_SHA
   DHE_RSA_WITH_AES_128_CBC_SHA
   RSA_WITH_AES_256_CBC_SHA
   DHE_RSA_WITH_AES_256_CBC_SHA
   (TLSv1.0: idem)
```

**After SSL version 3 is disabled:**
Supported versions: TLSv1.0
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
  TLSv1.0
    RSA_EXPORT_WITH_RC4_40_MD5
    RSA_WITH_RC4_128_MD5
    RSA_WITH_RC4_128_SHA
    RSA_EXPORT_WITH_DES40_CBC_SHA
    RSA_WITH_DES_CBC_SHA
    RSA_WITH_3DES_EDE_CBC_SHA
    DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
    DHE_RSA_WITH_DES_CBC_SHA
    DHE_RSA_WITH_3DES_EDE_CBC_SHA
    RSA_WITH_AES_128_CBC_SHA
    DHE_RSA_WITH_AES_128_CBC_SHA
    RSA_WITH_AES_256_CBC_SHA
    DHE_RSA_WITH_AES_256_CBC_SHA

Disabling the weak ciphers

A cipher is an algorithm for performing encryption or decryption. When the sslProtocol is set to TLS, only the TLS and default ciphers are enabled. However, the strength of the ciphers will not be considered when they are enabled. Therefore, to disable the weak ciphers, you enter only the ciphers that you want the server to support in a comma-separated list in the ciphers attribute. Also, if you do not add this cipher attribute or keep it blank, all SSL ciphers by JSSE will be supported by your server. This will enable the weak ciphers.

1. Make a backup of the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file and stop the server (same as for disabling SSL version 3).
2. Add the cipher attribute to the existing configuration in the `catalina-server.xml` file by adding the list of ciphers that you want your server to support as follows:
   
   ```xml
   ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,
   TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,
   SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA"
   ```

3. Start the server.

Enabling Java Security Manager

The Java Security Manager is used to define various security policies that prevent untrusted code from manipulating your system. Enabling the Java Security Manager for WSO2 products activates the Java permissions that are in the `<PRODUCT_HOME>/repository/conf/sec.policy` file. You modify this file to change the Java security permissions as required.

The steps below show how to enable the Java Security Manager for WSO2 products.
Before you begin, ensure that you have Java 1.8 installed.

1. Download the WSO2 product to any location (e.g., `<HOME>/user/<product-pack>` folder).

2. To sign the JARs in your product, you need a key. Generate it using the `keytool` command as follows:

   ```bash
   keytool -genkey -alias signFiles -keyalg RSA -keystore signkeystore.jks -validity 3650 -dname "CN=Sanjeewa,OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK"
   Enter keystore password:
   Re-enter new password:
   Enter key password for (RETURN if same as keystore password)
   ```

   The default keystore of the WSO2 products is `wso2carbon.jks`, which is in the `<PRODUCT_HOME>/repository/resources/security` folder. It is used for signing JARs.

3. Import the `signFiles` public key certificate that you created earlier to `wso2carbon.jks`. The sample below shows the security policy file referring the signer certificate from the `wso2carbon.jks` file:

   ```bash
   $ keytool -export -keystore signkeystore.jks -alias signFiles -file sign-cert.cer
   $ keytool -import -alias signFiles -file sign-cert.cer -keystore repository/resources/security/wso2carbon.jks
   Enter keystore password:
   Owner: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK
   Issuer: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK
   Serial number: 5486f3b0
   Valid from: Tue Dec 09 18:35:52 IST 2014 until: Fri Dec 06 18:35:52 IST 2024
   Certificate fingerprints:
   Signature algorithm name: SHA1withRSA
   Version: 3
   Trust this certificate? [no]: yes
   Certificate was added to keystore
   ```

4. Prepare the scripts to sign the JARs and grant them the required permission. For example, the `signJar.sh` script given below can be used to sign each JAR file separately or you can use the `signJars.sh` script, which runs a loop to read all JARs and sign them.

   ```bash
   ```
signJar.sh script

#!/bin/bash
set -e
jarfile=$1
keystore_file="signkeystore.jks"
keystore_keyalias='signFiles'
keystore_storepass='wso2123'
keystore_keypass='wso2123'
signjar="${JAVA_HOME}/bin/jarsigner -sigalg MD5withRSA -digestalg SHA1 -keystore $keystore_file -storepass $keystore_storepass -keypass $keystore_keypass"
verifyjar="${JAVA_HOME}/bin/jarsigner -keystore $keystore_file -verify"
echo "Signing $jarfile"
signjar $jarfile $keystore_keyalias
echo "Verifying $jarfile"
verifyjar $jarfile
# Check whether the verification is successful.
if [ $? -eq 1 ]
then
echo "Verification failed for $jarfile"
fi

signJars.sh script

#!/bin/bash
if [[ ! -d $1 ]]; then
echo "Please specify a target directory"
exit 1
fi
for jarfile in `find . -type f -iname \.jar`
do
  ./signJar.sh $jarfile
done

5. Execute the following commands to sign the JARs in your product:

./signJars.sh /HOME/user/<product-pack>

Every time you add an external JAR to the WSO2 product, sign them manually using the above instructions for the Java Security Manager to be effective. You add external JARs to the server when extending the product, applying patches etc.

6. Open the startup script in the <PRODUCT_HOME>/bin folder. For Linux, it is wso2server.sh.

7. Add the following system properties to the startup script and save the file:
8. Create a sec.policy file with the required security policies in the <PRODUCT_HOME>/repository/conf folder and start the server. Starting the server makes the Java permissions defined in the sec.policy file take effect.

An example of a sec.policy file is given below. It includes mostly WSO2 Carbon-level permissions.

```java
grant {
    // Allow socket connections for any host
    permission java.net.SocketPermission "*:1-65535", "connect,resolve";

    // Allow to read all properties. Use -Ddenied.system.properties in wso2server.sh to restrict properties
    permission java.util.PropertyPermission "+", "read";
    permission java.lang.RuntimePermission "getClassLoader";

    // CarbonContext APIs require this permission
    permission java.lang.management.ManagementPermission "control";

    // Required by any component reading XMLs. For example:
    org.wso2.carbon.databridge.agent.thrift:4.2.1.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind.v2.runtime.reflect";

    // Required by org.wso2.carbon.ndatasource.core:4.2.0. This is only necessary after adding above permission.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind";
};
```

**Multitenancy**

The goal of multitenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single server/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multitenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

You can register tenant domains using the Management Console of WSO2 products.
This section covers the following topics:

- Managing Tenants

**Managing Tenants**

You can add a new tenant in the management console and then view it by following the procedure below. In order to add a new tenant, you should be logged in as a super user.

1. Click **Add New Tenant** in the **Configure** tab of your product's management console.

2. Enter the tenant information in **Register A New Organization** screen as follows, and click **Save**.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>The domain name for the organization, which should be unique (e.g., abc.com). This is used as a unique identifier for your domain. You can use it to log into the admin console to be redirected to your specific tenant. The domain is also used in URLs to distinguish one tenant from another.</td>
</tr>
<tr>
<td>Select Usage Plan for Tenant</td>
<td>The usage plan defines limitations (such as number of users, bandwidth etc.) for the tenant.</td>
</tr>
<tr>
<td>First Name/Last Name</td>
<td>The name of the tenant admin.</td>
</tr>
<tr>
<td>Admin Username</td>
<td>The login username of the tenant admin. The username always ends with the domain name (e.g., <a href="mailto:admin@abc.com">admin@abc.com</a>)</td>
</tr>
<tr>
<td>Admin Password</td>
<td>The password used to log in using the admin username specified.</td>
</tr>
<tr>
<td>Admin Password (Repeat)</td>
<td>Repeat the password to confirm.</td>
</tr>
<tr>
<td>Email</td>
<td>The email address of the admin.</td>
</tr>
</tbody>
</table>

3. After saving, the newly added tenant appears in the **Tenants List** page as shown below. Click **View Tenants** in the **Configure** tab of the management console to see information of all the tenants that currently exist in
the system. If you want to view only tenants of a specific domain, enter the domain name in the **Enter the Tenant Domain** parameter and click **Find**.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Email</th>
<th>Created Date</th>
<th>Active</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>wso2.com</td>
<td><a href="mailto:frankie.axlon@gmail.com">frankie.axlon@gmail.com</a></td>
<td>2014/12/17 12:03:06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abc.com</td>
<td><a href="mailto:dalek.martin@gmail.com">dalek.martin@gmail.com</a></td>
<td>2014/12/17 13:43:46</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Integration

- Installing R to work with WSO2 CEP
- Supporting Different Transports
- Installing Machine Learner Features
- Installing WSO2 GPL Features

#### Installing R to work with WSO2 CEP

**Configuring R in Linux Environment**

Follow steps below to configure R.

1. Install R. For more information see *The R Project for Statistical Computing*.

   This guide uses R version 3.2.2.

2. You can verify whether the R installation is successful by typing command "R" in a new terminal. Upon successful installation you will get a log as shown below.
2. Install JRI. For more information see Java/R Interface Documentation.

4. Set path accordingly for $R_HOME and $JRI_HOME

   Eg: $R_HOME=/usr/lib/R
       $JRI_HOME=<Root folder>/R/x86_64-pc-linux-gnu-library/3.2/rJava/jri

5. Add a symbolic link which acts as a pointer to link libjri.so file in rJava/jri/libjri.so to the installation path of R

   Eg: ln -s /home/ubuntu/R/x86_64-pc-linux-gnu-library/3.2/rJava/jri/libjri.so /usr/lib

**Supporting Different Transports**

Follow the steps mentioned below to add support for each transport type

- MQTT Transport
- Kafka Transport
- SMS Transport
- JMS Transport

**MQTT Transport**

1. Download MQTT client library (mqtt-client-0.4.0.jar).
2. Add the JAR to the <PRODUCT_HOME>/repository/components/lib/ directory.
Kafka Transport

1. Download Apache Kafka server.

   This guide uses Kafka 2.10-0.8.2.1 version.

2. Copy the following client JAR files from `<KAFKA_HOME>/lib/` directory to `<PRODUCT_HOME>/repository/components/lib/` directory.
   - kafka_2.10-0.8.2.1.jar
   - zkclient-0.3.jar
   - scala-library-2.10.4.jar
   - zookeeper-3.4.6.jar
   - metrics-core-2.2.0.jar
   - kafka-clients-0.8.2.1.jar

   The above jars are the client jars for Kafka 2.10-0.8.1. If you want to use the client jars for Kafka_2.10-0.9.0.1 or Kafka_2.11-0.9.0.1, the following needs to be done.

   1. Copy the following client JAR files from `<KAFKA_HOME>/lib/` directory to `<PRODUCT_HOME>/repository/components/lib/` directory.
      - kafka_2.11-0.9.0.1.jar
      - kafka-clients-0.9.0.1.jar
      - metrics-core-2.2.0.jar
      - scala-library-2.11.7.jar
      - scala-parser-combinators_2.11-1.0.4.jar
      - zkclient-0.7.jar
      - zookeeper-3.4.6.jar
   
   2. Download the jass.conf file and save it in the `<PRODUCT_HOME>/repository/conf/identity` directory.

Kafka_2.10-0.9.0.1 is backward compatible. Therefore, you can use Kafka_2.10-0.8.2.1 client jars to connect with Kafka_2.10-0.9.0.1.

SMS Transport

1. Download and copy following libraries to `<PRODUCT_HOME>/repository/components/lib/` directory.
   - axis2-transport-sms-1.0.0.jar
   - jsmpp-2.1.0.jar

JMS Transport

Follow the steps to configure Apache ActiveMQ message broker

1. Install Apache ActiveMQ JMS.

   This guide uses ActiveMQ versions 5.7.0 - 5.9.0. If you want to use a later version, for instructions on the necessary changes to the configuration steps, go to Apache ActiveMQ Documentation.
2. Add the following ActiveMQ JMS-specific JAR files to the `<PRODUCT_HOME>/repository/components/lib/` directory.
   - `<ACTIVEMQ_HOME>/lib/geronimo-j2ee-management_1.1_spec-1.0.1.jar`
   - `<ACTIVEMQ_HOME>/lib/activemq-core-x.x.x.jar` (for 5.7.0 and below)
   - `<ACTIVEMQ_HOME>/lib/hawtbuf-1.9.jar` (for 5.8.0 and above)
   - `<ACTIVEMQ_HOME>/lib/activemq-client-x.x.x.jar` (for 5.8.0 and above)

Follow the steps to configure **Apache Qpid** message broker

1. Install **JMS-Qpid Broker** and **JMS-Qpid Client**.

   This guide uses Apache Qpid version 0.32. For more instructions on Apache Qpid go to Qpid documentation.

2. Add the following Qpid JMS-specific JAR files to `<PRODUCT_HOME>/repository/components/lib/` directory.
   - `<QPID-CLIENT_HOME>/lib/geronimo-jms_1.1_spec-1.1.1.jar`
   - `<QPID-CLIENT_HOME>/lib/qpid-client-0.32.jar`
   - `<QPID-CLIENT_HOME>/lib/qpid-common-0.32.jar`

Follow the steps to configure **WSO2 Message Broker (MB)**

1. Download and install WSO2 Message Broker. For instructions on WSO2 MB, go to Message Broker documentation.

   This guide uses WSO2 Message Broker (MB) version 3.1.0.

2. Add the following JMS-specific JAR files to `<PRODUCT_HOME>/repository/components/lib/` directory.
   - `<MB_HOME>/client-lib/andes-client-3.1.1.jar`
   - `<MB_HOME>/client-lib/log4j-1.2.13.jar`
   - `<MB_HOME>/client-lib/slf4j-1.5.10.wso2v1.jar`
   - `<MB_HOME>/client-lib/geronimo-jms_1.1_spec-1.1.0.wso2v1.jar`

Follow the steps to configure **TIBCO EMS Server**

1. Download and install TIBCO Enterprise Message Service. For more information on installing, go to TIBCO documentation.

   This guide uses TIBCO EMS Server version 8.2.0 V7

2. Add the following JMS-specific JAR files to `<PRODUCT_HOME>/repository/components/lib/` directory.
   - `<TIBCO_EMS_HOME>/lib/jms-2.0.jar`
   - `<TIBCO_EMS_HOME>/lib/tibjms.jar`

   You need to remove the line `javax.jms` in `launch.ini` file which is located in `<CEP_HOME>/repository/conf/etc to avoid OSGi package path conflicts as below.
Installing Machine Learner Features

- Introduction
- Installing required features in WSO2 CEP

Introduction

As explained in WSO2 Admin Guide - Working with Features, each WSO2 product is a collection of reusable software units called features. A single feature is a list of components and/or other features. This section describes how to install WSO2 Machine Learner features in WSO2 CEP.

Installing required features in WSO2 CEP

Follow the steps below to install the required features in WSO2 CEP.

1. Log in to the WSO2 CEP management console using admin/admin credentials and the following URL: https://<CEP_HOME>:<CEP_PORT>/carbon/
2. Click Configure, and then click Features.
3. Click Repository Management, and then click Add Repository.
4. Enter the details as shown below to add the Carbon P2 repository.

Feature Management

5. Click Add.
6. Click Available Features tab, and select the repository added in the previous step.
7. Deselect the Group features by category option.
8. Click Find Features. It can take a while to list out all the available features in the feature repository. Once listed, select the following features.

   - Machine Learner Core
   - Machine Learner Commons
• Machine Learner Database Service
• ML Siddhi Extension
• Analytics Core Server
• Analytics Data Source

If you cannot see this feature, retry with one of the following suggestions:

- Try adding a more recent P2 repository. The repository you added could be deprecated.
- Check the Installed Features tab to see whether the feature is already installed.

9. Once the features are selected, click Install to proceed with the installation.

10. Click Next, and then select I accept the terms of the license agreement.
11. Once the installation is completed, click Restart Now, and click Yes in the message which appears.

When installing ML features in an Apache Storm cluster, it is recommended to use a pom file instead of the Management Console. The Management Control only allows the features to be installed in the default profile instead of in all the profiles of the CEP nodes, and as a result, an exception occurs when events are sent to nodes that do not use the default profile. For more information, see Installing Features using pom Files.

When you run WSO2 CEP in a distributed mode, the following needs to be carried out `<CEP_HOME>/samples/cep/utils/storm-dependencies.jar/pom.xml`

1. The following dependencies should be uncommented in the `<CEP_HOME>/samples/cep/utils/storm-dependencies.jar/pom.xml` file as shown below.
<dependency>
  <groupId>org.wso2.carbon.ml</groupId>
  <artifactId>org.wso2.carbon.ml.siddhi.extension</artifactId>
  <version>${carbon.ml.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.carbon.ml</groupId>
  <artifactId>org.wso2.carbon.ml.core</artifactId>
  <version>${carbon.ml.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.carbon.ml</groupId>
  <artifactId>org.wso2.carbon.ml.database</artifactId>
  <version>${carbon.ml.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.carbon.ml</groupId>
  <artifactId>org.wso2.carbon.ml.commons</artifactId>
  <version>${carbon.ml.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.carbon.metrics</groupId>
  <artifactId>org.wso2.carbon.metrics.manager</artifactId>
  <version>${carbon.metrics.version}</version>
</dependency>

<!-- Dependencies for Spark --&gt;
<dependency>
  <groupId>org.wso2.orbit.org.apache.spark</groupId>
  <artifactId>spark-core_2.10</artifactId>
  <version>${spark.core.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.orbit.org.apache.spark</groupId>
  <artifactId>spark-sql_2.10</artifactId>
  <version>${spark.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.orbit.org.apache.spark</groupId>
  <artifactId>spark-mllib_2.10</artifactId>
  <version>${spark.version}</version>
</dependency>

<dependency>
  <groupId>org.wso2.orbit.org.apache.spark</groupId>
  <artifactId>spark-streaming_2.10</artifactId>
  <version>${spark.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.org.scalanlp</groupId>
  <artifactId>breeze_2.10</artifactId>
  <version>${breeze.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.jblas</groupId>
  <artifactId>jblas</artifactId>
  <version>${jblas.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.spire-math</groupId>
  <artifactId>spire_2.10</artifactId>
  <version>${spire.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>${hadoop.client.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.uncommons.maths</groupId>
  <artifactId>uncommons-maths</artifactId>
  <version>${uncommons.maths.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.json4s</groupId>
  <artifactId>json4s-jackson_2.10</artifactId>
  <version>${json4s.jackson.version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>${slf4j.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.github.fommil.netlib</groupId>
  <artifactId>core</artifactId>
  <version>${fommil.netlib.version}</version>
</dependency>
<dependency>
  <groupId>org.wso2.orbit.sourceforge.f2j</groupId>
  <artifactId>arpack_combined</artifactId>
  <version>${arpack.combined.version}</version>
</dependency>
<dependency>
  <groupId>org.scala-lang</groupId>
  <artifactId>scala-library</artifactId>
  <version>${scala.version}</version>
</dependency>
<groupId>org.apache.commons</groupId>
<dependency>
  <artifactId>commons-csv</artifactId>
  <version>${commons.csv.version}</version>
</dependency>

<!-- ML extension dependencies -->

<include>org.wso2.orbit.org.apache.spark:spark-core_2.10</include>
<include>org.wso2.orbit.org.apache.spark:spark-sql_2.10</include>
<include>org.wso2.orbit.org.apache.spark:spark-mllib_2.10</include>
<include>org.wso2.orbit.org.apache.spark:spark-streaming_2.10</include>
<include>org.wso2.orbit.org.scalanlp:breeze_2.10</include>
<include>org.wso2.orbit.jblas:jblas</include>
<include>org.wso2.orbit.spire-math:spire_2.10</include>
<include>org.wso2.orbit.org.apache.hadoop:hadoop-client</include>
<include>org.wso2.uncommons.maths:uncommons-maths</include>
<include>org.wso2.json4s:json4s-jackson_2.10</include>
<include>org.slf4j:slf4j-api</include>
<include>org.wso2.orbit.github.fommil.netlib:core</include>
<include>org.wso2.orbit.sourceforge.f2j:arpack_combined</include>
<include>org.scalalang:scala-library</include>
<include>org.apache.commons:commons-csv</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.core</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.database</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.commons</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.siddhi.extension</include>
<include>org.wso2.carbon.metrics:org.wso2.carbon.metrics.manager</include>
2. Run the following command from the `<CEP_HOME>/samples/cep/utils/storm-dependencies` directory.

```
jar mvn clean install
```

This will generate a jar in the `target` directory.

Queries are collectively processed by all the nodes in a CEP cluster. Therefore, make sure that the ML model is located in the same path in all the nodes. This allows all the nodes to access the model when events are sent to a specific node.

---

## Installing WSO2 GPL Features

- **Introduction**
- **Getting GPL P2 repositories (Optional)**
- **Building the P2 repositories (Optional)**
- **Installing required features in WSO2 CEP**

### Introduction

As explained in Feature Management, each WSO2 product is a collection of reusable software units called features. A single feature is a list of components and/or other features. This section describes how to install WSO2 GPL features in WSO2 CEP.

### Getting GPL P2 repositories (Optional)

1. Download the latest p2 repository from [https://github.com/wso2-gpl/carbon-event-processing/releases/download/v2.0.4/p2-repo.zip](https://github.com/wso2-gpl/carbon-event-processing/releases/download/v2.0.4/p2-repo.zip)

2. Unzip the p2 repository (p2-repo.zip) into a local directory in your machine.

### Building the P2 repositories (Optional)

1. Download the repositories and build them using `mvn clean install` command in the following order.
   
   1. orbit
   2. siddhi
   3. carbon-event-processing

2. Copy the generated P2 repository in `carbon-event-processing/repository/target/p2-repo` into a local directory in your machine.

### Installing required features in WSO2 CEP

Follow the steps below to install the required features in WSO2 CEP.

1. Download WSO2 CEP, and start the server. For instructions, see Getting Started.
2. Log in to the CEP management console.
3. Click `Configure`, and then click `Features`.
4. Click `Repository Management`, and then click `Add Repository`.
5. Enter the details as shown below to add the created P2 repository.
Parameter Name | Value
---|---
Name | WSO2 GPL Feature Repository
Local | /carbon-event-processing/repository/target/p2-repo

6. Click **Add** to add the repository.
7. Click the **Available Features** tab.
8. In the **Repository** parameter, select the WSO2 GPL Feature Repository repository you previously added.
9. Click **Find Features**. Once all the features are listed, select the following features.
   - GPL - Siddhi Eval Script Extension
   - GPL - Siddhi Geo Extension
   - GPL - Siddhi NLP Extension
   - GPL - Siddhi PMML Extension
   - GPL - Siddhi R Extension

If you cannot see this feature, retry with one of the following suggestions:
- Clear the **Group features by category** check box, and then click **Find Features** again.
- Try adding a more recent P2 repository. The repository you added could be deprecated.
- Check the **Installed Features** to see whether the feature is already installed.

10. Once the features are selected, click **Install** to proceed with the installation.
11. Once the installation is completed, restart WSO2 CEP.
Reference Guide

The following topics provide reference information for working with WSO2 CEP.

- Calling Admin Services from Apps

Calling Admin Services from Apps

All WSO2 products are built on top of the WSO2 Carbon kernel. WSO2 Carbon is managed internally using SOAP Web services known as admin services. All WSO2 products come with a management console UI, which communicates with these admin services underneath to facilitate administration capabilities through the UI.

However, there can be instances where you want to call these backend Web services directly. For information on how to discover the admin services of WSO2 API Manager and invoke them, see Calling Admin Services from Apps section in WSO2 Carbon documentation.
## Getting Support

In addition to this documentation, there are several ways to get help as you work on WSO2 products.

| ![Resource Icon] | **Explore learning resources**: For tutorials, articles, whitepapers, webinars, and other learning resources, look in the Resources menu on the WSO2 website. For training materials, click WSO2 Training on the Support & Training menu. In products that have a visual user interface, click the Help link in the top right-hand corner to get help with your current task. |
| ![Support Icon] | **Try our support options**: WSO2 offers a variety of development and production support programs, ranging from web-based support during normal business hours to premium 24x7 phone support. For support information, see [http://wso2.com/support/](http://wso2.com/support/). |
| ![Forum Icon] | **Ask questions in the user forums** at [http://stackoverflow.com](http://stackoverflow.com). Ensure that you tag your question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you can't find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at [http://wso2.org/mail](http://wso2.org/mail). |
| ![Report Icon] | **Report issues**, submit enhancement requests, track and comment on issues using our public bug-tracking system, and contribute samples, patches, and tips & tricks (see the WSO2 Contributor License Agreement). |
FAQ

Does CEP 4.0.0 support JMS queue and topic

- **Event Processing**
  - Can Siddhi be used outside WSO2 CEP?
  - What are the event processing engines supported by the CEP?
  - Does WSO2 provide support for Esper and Drools Fusion?
  - Is distributed processing supported in CEP?
  - Is it possible to match event patterns in Siddhi?
  - Can I refer to an RDBMS table in my execution plan?
  - What are the domain specific operators available in Siddhi?
  - Is it possible to write extensions to Siddhi? If yes how?
  - Is it possible to do time series analysis with CEP?

- **Event Receiving and Publishing**
  - How can CEP receive events from an external event source?
  - How can CEP publish events to an external event sink?
  - Is it possible to write custom Transport Adapters to receive and send events?
  - Does CEP support JMS queue and topic?
  - What are the JMS brokers supported by the CEP?
  - What are the ways to send notifications from the CEP?

- **Other**
  - Is there any way to trace the events that are going through the CEP?
  - Is it possible to visualize how the events are processed in CEP?
  - Is there any "playground" that I could use to learn or try-out Siddhi?
  - Can I create query templates in CEP?

**Event Processing**

*Can Siddhi be used outside WSO2 CEP?*

Yes, Siddhi is just a Java Library that can be embedded to any system. See Writing queries with Siddhi 3.0.x | An introduction blog post that explains how to use Siddhi as a Java Library.

*What are the event processing engines supported by the CEP?*

WSO2 CEP only supports Siddhi as the event processing engine.

*Does WSO2 provide support for Esper and Drools Fusion?*

No. WSO2 only provides support for Siddhi as the event processing engine for WSO2 CEP.

*Is distributed processing supported in CEP?*

Yes, this is supported. For more information, see Distributed mode deployment.

*Is it possible to match event patterns in Siddhi?*

Yes, it is possible to correlate events over time and detect event patterns based on the order of event arrival. For more information, see Patterns.

*Can I refer to an RDBMS table in my execution plan?*

Yes, you could use RDBMS event tables to achieve this. An event table is a structured representation of stored data, allowing stored data to be accessed and manipulated at runtime.

*What are the domain specific operators available in Siddhi?*
Siddhi supports a range of Time Series, Mathematical, String Geo REGEX Time NLP PMML Based Predictive Analytics Machine Learning Kalman Map Reorder and Markov Models functions.

**Is it possible to write extensions to Siddhi? If yes how?**

Yes, it is possible to extend the capabilities of Siddhi, according to custom requirements. This can be done by writing extensions to Siddhi. See Writing Extensions to Siddhi.

**Is it possible to do time series analysis with CEP?**

Yes. This is possible with the time series extension.

**Event Receiving and Publishing**

**How can CEP receive events from an external event source?**

Based on the transport that the event source is available on, create an event receiver. To select a receiver to match your requirement, see Event Receiver Types.

**How can CEP publish events to an external event sink?**

Based on the transport that the event sink is available on, create an event publisher. To select a publisher to match your requirement, see Event Publisher Types.

**Is it possible to write custom Transport Adapters to receive and send events?**

Yes, for creating custom transports to receive events, see Building Custom Event Receivers. For creating custom transports to publish events, see Building Custom Event Publishers.

**Does CEP support JMS queue and topic?**

Yes, CEP can be integrated with JMS queues and topics from version 3.0.0 onwards. For information on receiving events via JMS transport, see JMS Event Receiver. For information on publishing events via JMS transport, see JMS Event Publisher.

**What are the JMS brokers supported by the CEP?**

WSO2 CEP supports any JMS broker. CEP 4.0.0 has been tested with WSO2 Message Broker, ActiveMQ (versions 5.8.0, 5.9.0 and 5.10.0), IBM WebSphere MQ, Qpid and Tibco EMS.

For more information on how to configure WSO2 CEP with the JMS brokers mentioned above in order to receive events, see JMS Event Receiver.

For more information on how to configure WSO2 CEP with the JMS brokers mentioned above in order to publish events, see JMS Event Publisher.

**What are the ways to send notifications from the CEP?**

Notifications can be sent out in the form of Email and/or SMS. Use Email Event Publisher to send notifications via Email and SMS Event Publisher to send notifications via SMS.

**Other**

**Is there any way to trace the events that are going through the CEP?**

Yes, the Event Tracer traces the events as they get transferred between the components inside the CEP. For more information, see CEP Event Tracer.
Is it possible to visualize how the events are processed in CEP?

Yes. To visualize the event flow, use the Event Flow.

Is there any "playground" that I could use to learn or try-out Siddhi?

Yes. The CEP has a TryIt feature that you can use to try out Siddhi queries against events. For more information, see Siddhi Try It Tool.

Can I create query templates in CEP?

Yes. The CEP allows you to create query templates that other users can customize later in a user-friendly dashboard. This makes it easy for non-technical users to use WSO2 CEP. For more information, see Managing CEP Artifacts Via the Template Manager.
Site Map

Use this site map to quickly find the topic you're looking for by searching for a title on this page using your browser's search feature. You can also use the search box in the left navigation panel of this window to search for a word or phrase in all the pages in this documentation.
Glossary

Latency | Throughput

---

**Latency**

The time taken by the event flow to process a single event. A lower latency indicates better performance in terms of the efficiency with which the events are processed. Latency is affected by the receiver/publisher adapter type used as well as the performance configurations (see **Performance Tuning Recommendations** for further information). Reducing latency may involve achieving a lower throughput because the load handled by an event flow may need to be reduced in order to increase the event processing speed.

---

**Throughput**

The number of events handled by an event flow per second. A higher throughput indicates better performance in terms of event processing capacity of the CEP server. The throughput is affected by the performance configurations (see **Performance Tuning Recommendations** for further information). Increasing the throughput may involve increasing the latency because an increased load of events handled by an event flow can reduce the event processing speed.